

digital  
microcomputer  
handbook

1611

A  
0169

digital

21-11549-01  
7618

1621

2004 C  
0768

digital

23-001C3-00  
7626

1631

3010  
0709

digital

23-001B5-00  
7701

1631

3007  
0097

digital

23-002B5-00  
7701

# digital

**DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard,  
Massachusetts 01754, Telephone: (617) 897-5111**

## **SALES AND SERVICE OFFICES**

**UNITED STATES—ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) •**

**INTERNATIONAL—ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Djakarta • IRELAND, Dublin • ITALY, Milan and Turin • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Edinburgh, Leeds, London, Manchester and Reading • VENEZUELA, Caracas •**

**SECTION 1 LSI-11 FAMILY HARDWARE**



**SECTION 2 OPERATION**



**SECTION 3 PROCESSOR**



**SECTION 4 SYSTEM SOFTWARE**



**SECTION 5 DEC SERVICES**



**APPENDICES**





**digital**

# **microcomputer handbook**

**digital equipment corporation**

Copyright © 1976 by Digital Equipment Corporation

The material in this manual is for informational purposes and is subject to change without notice.

Digital Equipment Corporation assumes no responsibility for any errors which may appear in this handbook.

The following are trademarks of  
Digital Equipment Corporation, Maynard, Massachusetts:

DEC

PDP

FLIP CHIP

FOCAL

DIGITAL

DECUS

# CONTENTS

## SECTION I LSI-11 FAMILY HARDWARE

|   |      |
|---|------|
| <b>CHAPTER 1 INTRODUCTION</b>             | 1-1  |
| 1.1 GENERAL                               | 1-1  |
| 1.2 LSI-11 SYSTEM COMPONENTS              | 1-1  |
| 1.2.1 The LSI-11 Microcomputer            | 1-1  |
| 1.2.2 I/O Bus Concept                     | 1-7  |
| 1.2.3 Memory Options                      | 1-8  |
| 1.2.4 Peripheral Interface Options        | 1-9  |
| 1.2.5 Backplane Options                   | 1-17 |
| 1.2.6 Expansion Boxes                     | 1-21 |
| 1.2.7 Bus Accessory Options               | 1-21 |
| 1.3 THE PDP-11/03 SYSTEM                  | 1-22 |
| 1.4 THE PDP-11V03 SYSTEM                  | 1-22 |
| <br>                                      |      |
| <b>CHAPTER 2 SPECIFICATIONS</b>           | 2-1  |
| 2.1 GENERAL                               | 2-1  |
| 2.2 LSI-11 SYSTEM COMPONENTS              | 2-1  |
| 2.2.1 Modules                             | 2-1  |
| 2.2.2 Backplane Options                   | 2-1  |
| 2.2.3 BA11-ME and BA11-MF Expansion Boxes | 2-6  |
| 2.3 PDP-11/03 SYSTEM                      | 2-6  |
| 2.3.1 General                             | 2-6  |
| 2.3.2 H780 Power Supply                   | 2-7  |
| 2.3.3 Environmental Specifications        | 2-8  |
| 2.3.4 Mechanical Specifications           | 2-8  |
| 2.4 PDP-11V03 SYSTEMS                     | 2-11 |
| 2.4.1 General                             | 2-11 |
| 2.4.2 Specifications                      | 2-11 |
| <br>                                      |      |
| <b>CHAPTER 3 THE LSI-11 BUS</b>           | 3-1  |
| 3.1 CHOOSING AN I/O TRANSFER TYPE         | 3-1  |
| 3.2 DEVICE PRIORITY                       | 3-1  |
| 3.3 MODULE CONTACT FINGER IDENTIFICATION  | 3-2  |
| 3.4 BUS SIGNALS                           | 3-4  |
| 3.5 BUS CYCLES                            | 3-9  |
| 3.5.1 General                             | 3-9  |
| 3.5.2 Input Operations                    | 3-9  |
| 3.5.3 Output Operations                   | 3-9  |
| 3.6 DMA OPERATIONS                        | 3-11 |
| 3.7 INTERRUPTS                            | 3-15 |
| 3.8 BUS INITIALIZATION                    | 3-15 |
| 3.9 POWER-UP POWER-DOWN SEQUENCE          | 3-17 |
| 3.10 HALT MODE                            | 3-17 |
| 3.11 MEMORY REFRESH                       | 3-17 |
| 3.12 BUS SPECIFICATIONS                   | 3-18 |
| 3.13 BUS CONFIGURATIONS                   | 3-18 |
| 3.14 BUS SIGNAL TIMING                    | 3-21 |

|                  |   |      |
|------------------|---|------|
| <b>CHAPTER 4</b> | <b>LSI-11 MODULE DESCRIPTIONS</b>                       | 4-1  |
| 4.1              | GENERAL   | 4-1  |
| 4.2              | KD11-F MICROCOMPUTER                                    | 4-2  |
|                  | 4.2.1 General   | 4-2  |
|                  | 4.2.2 Basic Microcomputer Functions                     | 4-2  |
|                  | 4.2.3 KD11-F Resident Memory                            | 4-15 |
|                  | 4.2.4 DC-DC Power Inverter                              | 4-16 |
| 4.3              | MMV11-A 4K BY 16-BIT CORE MEMORY                        | 4-16 |
|                  | 4.3.1 General   | 4-16 |
|                  | 4.3.2 Functional Description                            | 4-18 |
| 4.4              | MRV11-AA 4K BY 16-BIT READ-ONLY MEMORY                  | 4-30 |
|                  | 4.4.1 General   | 4-30 |
|                  | 4.4.2 Functional Description                            | 4-31 |
| 4.5              | MSV11-B 4K BY 16-BIT SEMICONDUCTOR<br>READ/WRITE MEMORY | 4-33 |
|                  | 4.5.1 General   | 4-33 |
|                  | 4.5.2 Functional Description                            | 4-34 |
| 4.6              | DLV11 SERIAL LINE UNIT                                  | 4-36 |
|                  | 4.6.1 General   | 4-36 |
|                  | 4.6.2 Functional Description                            | 4-37 |
| 4.7              | DRV11 PARALLEL LINE UNIT                                | 4-42 |
|                  | 4.7.1 General   | 4-42 |
|                  | 4.7.2 Functional Description                            | 4-43 |
| 4.8              | DRV11-B DMA INTERFACE                                   | 4-48 |
| 4.9              | DRV11-P LSI-11 BUS FOUNDATION MODULE                    | 4-48 |
| 4.10             | LSI-11 BUS ACCESSORY OPTIONS                            | 4-48 |
|                  | 4.10.1 General  | 4-48 |
|                  | 4.10.2 Terminations                                     | 4-48 |
|                  | 4.10.3 Bootstrap ROM Logic                              | 4-50 |
|                  | 4.10.4 DMA Refresh Logic                                | 4-51 |
| 4.11             | H780 POWER SUPPLY                                       | 4-56 |
|                  | 4.11.1 General  | 4-56 |
|                  | 4.11.2 Specifications                                   | 4-57 |
|                  | 4.11.3 Functional Description                           | 4-59 |
|                  | 4.11.4 H780 Connections                                 | 4-67 |
| <br>             |   |      |
| <b>CHAPTER 5</b> | <b>CONFIGURING LSI-11 MODULES</b>                       | 5-1  |
| 5.1              | GENERAL   | 5-1  |
| 5.2              | PROCESSOR MODULE  | 5-2  |
|                  | 5.2.1 General   | 5-2  |
|                  | 5.2.2 Processor Module Jumpers                          | 5-2  |
|                  | 5.2.3 Installation                                      | 5-7  |
|                  | 5.2.4 Using the LSI-11 Microcomputer                    | 5-8  |
|                  | 5.2.5 Initialization and Power Fail                     | 5-9  |
| 5.3              | MSV11-B READ/WRITE MEMORY                               | 5-9  |
|                  | 5.3.1 General   | 5-9  |
|                  | 5.3.2 Address Jumpers                                   | 5-9  |
|                  | 5.3.3 Reply to Refresh Jumpers                          | 5-9  |
|                  | 5.3.4 Refresh Requirements                              | 5-10 |
| 5.4              | MMV11-A CORE MEMORY                                     | 5-11 |
|                  | 5.4.1 General   | 5-11 |
|                  | 5.4.2 Switch-Selected Addressing                        | 5-12 |
|                  | 5.4.3 Backplane Jumpers                                 | 5-13 |

|   |   |            |
|---|---|------------|
| 5.5                                     | MRV11-AA READ-ONLY MEMORY . . . . .                                     | 5-14       |
| 5.5.1                                   | General . . . . .   | 5-14       |
| 5.5.2                                   | Chip Type Jumpers . . . . .   | 5-15       |
| 5.5.3                                   | Address and Reply Jumpers . . . . .                                     | 5-15       |
| 5.5.4                                   | PROM Chips . . . . .  | 5-17       |
| 5.5.5                                   | Programming PROM Chips . . . . .  | 5-19       |
| 5.5.6                                   | I/O Timing and Bus Restrictions . . . . .                               | 5-19       |
| 5.6                                     | DLV11 SERIAL LINE UNIT . . . . .  | 5-19       |
| 5.6.1                                   | General . . . . .   | 5-19       |
| 5.6.2                                   | Jumper-Selected Addressing, Vectors,<br>and Module Operations . . . . . | 5-19       |
| 5.6.3                                   | Installation . . . . .  | 5-24       |
| 5.6.4                                   | Interfacing with 20 mA Current Loop Devices . . . . .                   | 5-26       |
| 5.6.5                                   | Interfacing with EIA-Compatible Devices . . . . .                       | 5-26       |
| 5.6.6                                   | Programming . . . . .   | 5-26       |
| 5.6.7                                   | Console Device . . . . .  | 5-28       |
| 5.7                                     | DRV11 PARALLEL LINE UNIT . . . . .                                      | 5-30       |
| 5.7.1                                   | General . . . . .   | 5-30       |
| 5.7.2                                   | Jumper-Selected Addressing and Vectors . . . . .                        | 5-30       |
| 5.7.3                                   | Installation . . . . .  | 5-33       |
| 5.7.4                                   | Interfacing to the User's Device . . . . .                              | 5-34       |
| 5.7.5                                   | Programming . . . . .   | 5-37       |
| 5.8                                     | DRV11-B DIRECT MEMORY ACCESS (DMA)<br>INTERFACE . . . . .               | 5-40       |
| 5.8.1                                   | General . . . . .   | 5-40       |
| 5.8.2                                   | Registers . . . . .   | 5-41       |
| 5.8.3                                   | Device and Vector Address Selection . . . . .                           | 5-44       |
| 5.8.4                                   | Functions . . . . .   | 5-47       |
| 5.8.5                                   | Device Cables and Signals . . . . .                                     | 5-47       |
| 5.9                                     | LSI-11 BUS FOUNDATION MODULE . . . . .                                  | 5-49       |
| 5.9.1                                   | General . . . . .   | 5-49       |
| 5.9.2                                   | Functions . . . . .   | 5-50       |
| 5.9.3                                   | Component Mounting Area . . . . .                                       | 5-53       |
| <b>CHAPTER 6 INSTALLATION . . . . .</b> |   | <b>6-1</b> |
| 6.1                                     | GENERAL . . . . .   | 6-1        |
| 6.2                                     | CONFIGURATION CHECKLIST . . . . .                                       | 6-1        |
| 6.3                                     | DEVICE PRIORITY . . . . .   | 6-2        |
| 6.3.1                                   | General . . . . .   | 6-2        |
| 6.3.2                                   | Priority Selection Using the H9270 Backplane . . . . .                  | 6-2        |
| 6.3.3                                   | H9270 Backplane/MMV11-A Configuration . . . . .                         | 6-3        |
| 6.3.4                                   | DDV11-B Expanded Backplane Configuration . . . . .                      | 6-4        |
| 6.4                                     | MODULE INSERTION AND REMOVAL . . . . .                                  | 6-5        |
| 6.5                                     | I/O CABLING . . . . .   | 6-6        |
| 6.6                                     | PDP-11/03 INSTALLATION PROCEDURE . . . . .                              | 6-7        |
| 6.6.1                                   | Packaging and Mounting . . . . .  | 6-7        |
| 6.6.2                                   | Power Requirements . . . . .  | 6-7        |
| 6.6.3                                   | Environmental Requirements . . . . .                                    | 6-7        |
| 6.7                                     | LSI-11 SYSTEM INSTALLATION . . . . .                                    | 6-7        |
| 6.7.1                                   | General . . . . .   | 6-7        |
| 6.7.2                                   | Mounting the Backplane . . . . .  | 6-7        |
| 6.7.3                                   | DC Power Connections . . . . .  | 6-8        |
| 6.7.4                                   | Backplane Ground Connection . . . . .                                   | 6-10       |

|                  |   |            |
|------------------|---|------------|
| 6.7.5            | Environmental Requirements . . . . .                | 6-10       |
| 6.7.6            | Externally Generated Bus Signals . . . . .          | 6-10       |
| 6.8              | USING LSI-11 BUS ACCESSORY OPTIONS . . . . .        | 6-15       |
| 6.8.1            | General . . . . .                                   | 6-15       |
| 6.8.2            | Using the REV11-A . . . . .                         | 6-16       |
| 6.8.3            | Using the REV11-C . . . . .                         | 6-17       |
| 6.8.4            | Using the TEV11 . . . . .                           | 6-17       |
| 6.8.5            | Using the BCV1-B . . . . .                          | 6-17       |
| 6.8.6            | Using the BCV1-A . . . . .                          | 6-18       |
| 6.9              | USING BA11-ME AND BA11-MF EXPANSION BOXES . . . . . | 6-19       |
| <b>CHAPTER 7</b> | <b>USING PROMS . . . . .</b>                        | <b>7-1</b> |
| 7.1              | GENERAL . . . . .                                   | 7-1        |
| 7.2              | PROM TYPES . . . . .                                | 7-1        |
| 7.3              | PROGRAMMING NOTES . . . . .                         | 7-1        |
| 7.4              | LOADING PROMS . . . . .                             | 7-3        |
| 7.4.1            | General . . . . .                                   | 7-3        |
| 7.4.2            | Word Format . . . . .                               | 7-3        |
| 7.4.3            | Addressing . . . . .                                | 7-3        |
| 7.5              | PROM FORMATTING USING THE QJV11 PROGRAM . . . . .   | 7-3        |
| 7.5.1            | General . . . . .                                   | 7-3        |
| 7.5.2            | Loading QJV11 . . . . .                             | 7-4        |
| 7.5.3            | Entering Parameters . . . . .                       | 7-5        |
| 7.5.4            | QJV11 Operation . . . . .                           | 7-8        |
| 7.6              | INSTALLING PROMS . . . . .                          | 7-10       |
| <b>CHAPTER 8</b> | <b>USER-DESIGNED INTERFACES . . . . .</b>           | <b>8-1</b> |
| 8.1              | GENERAL . . . . .                                   | 8-1        |
| 8.2              | BUS RECEIVER AND DRIVER CIRCUITS . . . . .          | 8-1        |
| 8.3              | PROGRAMMED INTERFACE . . . . .                      | 8-3        |
| 8.4              | INTERRUPT LOGIC . . . . .                           | 8-6        |
| 8.5              | DMA INTERFACE LOGIC . . . . .                       | 8-7        |
| <b>CHAPTER 9</b> | <b>MAINTENANCE . . . . .</b>                        | <b>9-1</b> |
| 9.1              | GENERAL . . . . .                                   | 9-1        |
| 9.2              | OPERATIONAL CHECKLIST . . . . .                     | 9-1        |
| 9.2.1            | General . . . . .                                   | 9-1        |
| 9.2.2            | LSI-11 System . . . . .                             | 9-1        |
| 9.2.3            | PDP-11/03 System . . . . .                          | 9-2        |
| 9.3              | USING PAPER TAPE DIAGNOSTICS . . . . .              | 9-3        |
| 9.3.1            | General . . . . .                                   | 9-3        |
| 9.3.2            | Loading Diagnostic Program Tapes . . . . .          | 9-4        |
| 9.3.3            | Program Modification Procedure . . . . .            | 9-4        |
| 9.3.4            | Program Starting and Execution . . . . .            | 9-5        |
| 9.3.5            | Diagnostic Program Results . . . . .                | 9-6        |
| 9.4              | USING RXDP DIAGNOSTICS . . . . .                    | 9-6        |
| 9.4.1            | RXDP Diagnostics . . . . .                          | 9-6        |
| 9.4.2            | Program Modification and Execution . . . . .        | 9-8        |
| 9.4.3            | Single Instruction Execution . . . . .              | 9-10       |
| 9.4.4            | Diagnostic Program Results . . . . .                | 9-11       |
| 9.4.5            | Running a Chain of Diagnostics . . . . .            | 9-11       |
| 9.4.6            | Making a Duplicate of the Diskette . . . . .        | 9-13       |
| 9.4.7            | Diagnostic Program Changes . . . . .                | 9-13       |
| 9.4.8            | Creating a Unique RXDP Diskette . . . . .           | 9-14       |

## SECTION II OPERATION

|  |     |
|--|-----|
| <b>CHAPTER 1 GENERAL PROCEDURES</b> . . . . .                            | 1-1 |
| 1.1 GENERAL . . . . .  | 1-1 |
| 1.2 THE CONSOLE DEVICE . . . . .   | 1-1 |
| 1.3 POWER-UP RESPONSE . . . . .  | 1-2 |
| 1.3.1 PDP-11/03 Power-On . . . . .                                       | 1-2 |
| 1.3.2 LSI-11 Initial Power-On . . . . .                                  | 1-4 |
| 1.3.3 ASCII Character Console Printout Program . . . . .                 | 1-4 |
| <b>CHAPTER 2 USING CONSOLE ODT COMMANDS</b> . . . . .                    | 2-1 |
| 2.1 THE HALT MODE . . . . .  | 2-1 |
| 2.2 ODT COMMANDS . . . . .   | 2-2 |
| <b>CHAPTER 3 USING REV11-A AND REV11-C COMMANDS</b> . . . . .            | 3-1 |
| 3.1 GENERAL . . . . .  | 3-1 |
| 3.2 REV11-A AND REV11-C COMMAND SET . . . . .                            | 3-1 |
| <b>CHAPTER 4 PAPER TAPE SYSTEM OPERATION</b> . . . . .                   | 4-1 |
| 4.1 GENERAL . . . . .  | 4-1 |
| 4.2 LOADING THE ABSOLUTE LOADER . . . . .                                | 4-1 |
| 4.3 LOADING PROGRAM TAPES . . . . .                                      | 4-3 |
| 4.3.1 General . . . . .  | 4-3 |
| 4.3.2 Normal Loading Procedure . . . . .                                 | 4-3 |
| 4.3.3 Relocated Loading Procedure . . . . .                              | 4-4 |
| 4.3.4 Self-Starting Programs . . . . .                                   | 4-4 |
| 4.4 PROGRAM STARTING AND EXECUTION . . . . .                             | 4-4 |
| 4.5 PAPER TAPE SOFTWARE . . . . .  | 4-6 |
| <b>CHAPTER 5 RXV11 FLOPPY DISK-BASED SYSTEM OPERATION</b> . . . . .      | 5-1 |
| 5.1 GENERAL . . . . .  | 5-1 |
| 5.2 BOOTSTRAPPING THE RXV11 . . . . .                                    | 5-1 |
| 5.2.1 General . . . . .  | 5-1 |
| 5.2.2 Bootstrapping the System Using the<br>REV11-A or REV11-C . . . . . | 5-1 |
| 5.2.3 Booting the System Via the Console Device . . . . .                | 5-2 |
| 5.2.4 Incorrect Loading . . . . .  | 5-3 |

## SECTION III PROCESSOR

|   |     |
|---|-----|
| <b>CHAPTER 1 INTRODUCTION</b> . . . . .         | 1-1 |
| 1.1 PROCESSOR HARDWARE . . . . .                | 1-1 |
| 1.1.1 General Registers . . . . .               | 1-1 |
| 1.1.2 The Processor Status Word (PSW) . . . . . | 1-3 |
| 1.1.3 Instruction Set . . . . .                 | 1-4 |
| 1.2 LSI-11 MEMORY ORGANIZATION . . . . .        | 1-5 |

|  |      |
|--|------|
| <b>CHAPTER 2 ADDRESSING MODES</b>            | 2-1  |
| 2.1 SINGLE OPERAND ADDRESSING                | 2-3  |
| 2.2 DOUBLE OPERAND ADDRESSING                | 2-3  |
| 2.3 DIRECT ADDRESSING                        | 2-5  |
| 2.3.1 Register Mode                          | 2-5  |
| 2.3.2 Autoincrement Mode                     | 2-7  |
| 2.3.3 Autodecrement Mode                     | 2-8  |
| 2.3.4 Index Mode                             | 2-9  |
| 2.4 DEFERRED (INDIRECT) ADDRESSING           | 2-11 |
| 2.5 USE OF THE PC AS A GENERAL REGISTER      | 2-13 |
| 2.5.1 Immediate Mode                         | 2-14 |
| 2.5.2 Absolute Addressing                    | 2-15 |
| 2.5.3 Relative Addressing                    | 2-16 |
| 2.5.4 Relative Deferred Addressing           | 2-16 |
| 2.6 USE OF STACK POINTER AS GENERAL REGISTER | 2-17 |
| 2.7 SUMMARY OF ADDRESSING MODES              | 2-17 |
| 2.7.1 General Register Addressing            | 2-17 |
| 2.7.2 Program Counter Addressing             | 2-19 |
| <br>   |      |
| <b>CHAPTER 3 INSTRUCTION SET</b>             | 3-1  |
| 3.1 INTRODUCTION                             | 3-1  |
| 3.2 INSTRUCTION FORMATS                      | 3-2  |
| 3.3 LIST OF INSTRUCTIONS                     | 3-4  |
| 3.4 SINGLE OPERAND INSTRUCTIONS              | 3-6  |
| 3.5 PS WORD OPERATORS                        | 3-17 |
| 3.6 DOUBLE OPERAND INSTRUCTIONS              | 3-18 |
| 3.7 PROGRAM CONTROL INSTRUCTIONS             | 3-24 |
| 3.8 MISCELLANEOUS                            | 3-24 |
| 3.9 RESERVED INSTRUCTIONS                    | 3-47 |
| 3.10 CONDITION CODE OPERATORS                | 3-49 |
| <br>   |      |
| <b>CHAPTER 4 EXTENDED ARITHMETIC OPTION</b>  | 4-1  |
| 4.1 GENERAL                                  | 4-1  |
| 4.2 FIXED POINT ARITHMETIC (EIS)             | 4-1  |
| 4.3 FLOATING POINT ARITHMETIC (FIS)          | 4-6  |
| <br>   |      |
| <b>CHAPTER 5 PROGRAMMING TECHNIQUES</b>      | 5-1  |
| 5.1 THE STACK                                | 5-1  |
| 5.2 SUBROUTINE LINKAGE                       | 5-5  |
| 5.2.1 Subroutine Calls                       | 5-5  |
| 5.2.2 Argument Transmission                  | 5-6  |
| 5.2.3 Subroutine Return                      | 5-9  |
| 5.2.4 LSI-11 Set Subroutine Advantages       | 5-9  |
| 5.2.5 Trap Subroutine Calls                  | 5-9  |
| 5.3 INTERRUPTS                               | 5-10 |
| 5.3.1 General Principles                     | 5-10 |
| 5.3.2 Nesting                                | 5-11 |
| 5.4 PROGRAMMING PERIPHERALS                  | 5-13 |
| 5.5 DEVICE REGISTERS                         | 5-14 |

## SECTION IV SYSTEM SOFTWARE

|   |      |
|---|------|
| <b>CHAPTER 1 INTRODUCTION</b>   | 1-1  |
| 1.1 SOFTWARE SYSTEMS  | 1-1  |
| 1.2 OPERATING SYSTEMS   | 1-1  |
| 1.3 LANGUAGES AND LANGUAGE PROCESSORS                                   | 1-3  |
| <b>CHAPTER 2 OPERATING SYSTEMS</b>                                      | 2-1  |
| 2.1 COMPONENTS AND FUNCTIONS  | 2-1  |
| 2.2 PROCESSING METHODS  | 2-3  |
| 2.3 DATA MANAGEMENT   | 2-4  |
| 2.3.1 Physical and Logical Units of Data                                | 2-4  |
| 2.3.2 Data Storage and Transfer Modes                                   | 2-6  |
| 2.3.3 I/O Devices and Physical Data<br>Access Characteristics           | 2-7  |
| 2.3.4 Physical Device Characteristics and<br>Logical Data Organizations | 2-10 |
| 2.3.5 File Structures and Access Methods                                | 2-12 |
| 2.3.6 Directories and Directory Access Techniques                       | 2-14 |
| 2.3.7 File Naming   | 2-15 |
| 2.4 USER INTERFACES   | 2-16 |
| 2.4.1 Special Terminal Commands   | 2-17 |
| 2.4.2 I/O Commands  | 2-17 |
| 2.4.3 Monitor and Command Language Commands                             | 2-19 |
| 2.5 PROGRAMMED SYSTEM SERVICES  | 2-20 |
| 2.6 SYSTEM UTILITIES  | 2-20 |
| <b>CHAPTER 3 LANGUAGE PROCESSORS</b>                                    | 3-1  |
| 3.1 LANGUAGE TRANSLATION SYSTEMS  | 3-1  |
| 3.2 PDP-11 ASSEMBLERS AND THE FORTRAN<br>COMPILERS                      | 3-4  |
| 3.3 INCREMENTAL COMPILERS   | 3-7  |
| <b>CHAPTER 4 FOREGROUND/BACKGROUND OPERATING<br/>  SYSTEM RT-11</b>     | 4-1  |
| 4.1 OPERATING SYSTEM FUNCTIONS AND FEATURES                             | 4-1  |
| 4.2 MONITOR ORGANIZATION  | 4-2  |
| 4.2.1 Monitor Components  | 4-3  |
| 4.2.2 General Memory Layout and Component Sizes                         | 4-4  |
| 4.2.3 I/O System Design and Operation                                   | 4-6  |
| 4.2.4 Batch Processing  | 4-7  |
| 4.2.5 Switching Between Single-Job and<br>Foreground/Background         | 4-7  |
| 4.3 SYSTEM CONVENTIONS  | 4-8  |
| 4.3.1 Physical Device Names   | 4-8  |
| 4.3.2 Filenames and Extensions  | 4-8  |
| 4.3.3 Data Formats  | 4-9  |
| 4.3.4 File Structure  | 4-9  |
| 4.4 COMMANDS  | 4-10 |
| 4.4.1 Keyboard Communication  | 4-10 |
| 4.4.2 Entering I/O Information Using the CSI                            | 4-14 |
| 4.4.3 BATCH Job Control Language  | 4-14 |

|                  |   |            |
|------------------|---|------------|
| 4.5              | MONITOR PROGRAMMED REQUESTS . . . . .                   | 4-15       |
| 4.5.1            | Summary of Programmed Requests . . . . .                | 4-17       |
| 4.5.2            | Program Environment Control . . . . .                   | 4-20       |
| 4.5.3            | Resource and System Interrogation . . . . .             | 4-22       |
| 4.5.4            | Command Interpretation . . . . .                        | 4-22       |
| 4.5.5            | File Operations . . . . .                               | 4-23       |
| 4.5.6            | Input/Output . . . . .                                  | 4-24       |
| 4.5.7            | Interjob Communications . . . . .                       | 4-26       |
| 4.5.8            | Timer Support . . . . .                                 | 4-26       |
| 4.5.9            | Program Termination or Suspension . . . . .             | 4-27       |
| 4.5.10           | Interrupt Service . . . . .                             | 4-27       |
| 4.6              | SYSTEM PROGRAMS . . . . .                               | 4-27       |
| 4.6.1            | EDIT Interactive Editor . . . . .                       | 4-28       |
| 4.6.2            | LINK Linker . . . . .                                   | 4-28       |
| 4.6.3            | LIBR Librarian . . . . .                                | 4-28       |
| 4.6.4            | ODT On-Line Debugger . . . . .                          | 4-29       |
| 4.6.5            | PATCH Code Patch Utility . . . . .                      | 4-29       |
| 4.6.6            | PATCHO Object Patch Utility . . . . .                   | 4-30       |
| 4.6.7            | PIP Peripheral Interchange Program Utility . . . . .    | 4-30       |
| 4.6.8            | SRCCOM Source Compare Utility . . . . .                 | 4-30       |
| 4.6.9            | FILEX File Exchange Utility . . . . .                   | 4-30       |
| 4.6.10           | DUMP File Dump Utility . . . . .                        | 4-30       |
| 4.7              | LANGUAGES . . . . .                                     | 4-30       |
| 4.7.1            | MACRO Assembler . . . . .                               | 4-30       |
| 4.7.2            | EXPAND Macro Expander and<br>ASEMBL Assembler . . . . . | 4-31       |
| 4.7.3            | FORTTRAN . . . . .                                      | 4-31       |
| 4.7.4            | FOCAL . . . . .   | 4-32       |
| 4.7.5            | Single-User BASIC . . . . .                             | 4-33       |
| 4.7.6            | Multi-User BASIC . . . . .                              | 4-34       |
| <b>CHAPTER 5</b> | <b>REAL-TIME MULTIPROGRAMMING RSX-11S . . . . .</b>     | <b>5-1</b> |
| 5.1              | FUNCTIONS AND FEATURES . . . . .                        | 5-1        |
| 5.2              | RSX-11S OPERATING SYSTEM CONCEPTS . . . . .             | 5-1        |
| 5.3              | SYSTEM ORGANIZATION AND GENERATION . . . . .            | 5-6        |
| 5.4              | SYSTEM CONVENTIONS . . . . .                            | 5-9        |
| 5.4.1            | Devices . . . . .                                       | 5-9        |
| 5.4.2            | MCR Operator Commands and<br>Terminal Control . . . . . | 5-9        |
| 5.5              | SYSTEM DIRECTIVES . . . . .                             | 5-11       |
| 5.6              | LANGUAGES . . . . .                                     | 5-14       |
| 5.6.1            | MACRO . . . . .   | 5-14       |
| 5.6.2            | FORTTRAN IV . . . . .                                   | 5-14       |
| <b>CHAPTER 6</b> | <b>MACRO . . . . .</b>                                  | <b>6-1</b> |
| 6.1              | FUNCTIONS AND FEATURES . . . . .                        | 6-1        |
| 6.2              | LANGUAGE . . . . .                                      | 6-1        |
| 6.2.1            | Symbols and Symbol Definitions . . . . .                | 6-2        |
| 6.2.2            | Directives . . . . .                                    | 6-4        |
| 6.3              | ASSEMBLER OPERATION . . . . .                           | 6-10       |
| 6.4              | ASSEMBLER ENVIRONMENTS . . . . .                        | 6-14       |

|                                       |      |
|---------------------------------------|------|
| <b>CHAPTER 7 FORTRAN IV</b>           | 7-1  |
| 7.1 FUNCTIONS AND FEATURES            | 7-1  |
| 7.2 LANGUAGE                          | 7-3  |
| 7.3 COMPILER OPERATION                | 7-9  |
| 7.4 FORTRAN IV OPERATING ENVIRONMENTS | 7-14 |

|   |      |
|---|------|
| <b>CHAPTER 8 BASIC</b>                                    | 8-1  |
| 8.1 FUNCTIONS AND FEATURES                                | 8-1  |
| 8.2 LANGUAGE  | 8-1  |
| 8.2.1 BASIC Files   | 8-7  |
| 8.2.2 Creating, Modifying and<br>Executing BASIC Programs | 8-7  |
| 8.3 COMPILER OPERATION                                    | 8-8  |
| 8.4 BASIC ENVIRONMENTS                                    | 8-11 |

## SECTION V DEC SERVICES

|                                       |     |
|---------------------------------------|-----|
| <b>CHAPTER 1 EDUCATIONAL SERVICES</b> | 1-1 |
| 1.1 GENERAL                           | 1-1 |
| 1.2 CATALOG COURSES                   | 1-1 |
| 1.3 CUSTOM COURSES                    | 1-1 |
| 1.4 ON-SITE INSTRUCTION               | 1-1 |
| 1.5 AUDIO-VISUAL COURSES              | 1-1 |
| 1.6 LSI-11, PDP-11/03 RELATED COURSES | 1-1 |

|                        |     |
|------------------------|-----|
| <b>CHAPTER 2 DECUS</b> | 2-1 |
|------------------------|-----|

|                              |     |
|------------------------------|-----|
| <b>CHAPTER 3 MAINTENANCE</b> | 3-1 |
| 3.1 GENERAL                  | 3-1 |
| 3.2 ON-SITE SERVICE          | 3-1 |
| 3.3 OFF-SITE SERVICE         | 3-1 |

## APPENDICES

|   |     |
|---|-----|
| <b>APPENDIX A GENERAL REFERENCE DATA</b>                                  | A-1 |
| <b>APPENDIX B INSTRUCTION TIMING</b>                                      | B-1 |
| <b>APPENDIX C LSI-11, PDP-11 PROGRAMMING/HARDWARE<br/>DIFFERENCE LIST</b> | C-1 |
| <b>APPENDIX D LSI-11, PDP-11/03 ENGINEERING BULLETIN</b>                  | D-1 |
| <b>APPENDIX E PERIPHERALS</b>   | E-1 |
| <b>APPENDIX F INTEGRATED CIRCUIT DIAGRAMS</b>                             | F-1 |
| <b>APPENDIX G ABSOLUTE LOADER FORMAT</b>                                  | G-1 |



The LSI-11 Family

## PREFACE

DIGITAL introduced the first PDP-11 processor in 1970. Since then, a family of PDP-11 computer products has been constantly evolving—not just a family of processors, but a family of peripherals, software, and services. Today, the PDP-11 family is the broadest family of compatible computer products on the market, with one of the latest additions being the LSI-11.

The LSI-11 is a 16-bit microcomputer with the speed and instruction set of a minicomputer. The LSI-11 was introduced in 1975; volume deliveries started during the fall of 1975. Due to its size and unique capabilities, it is being designed into many instrumentation, data processing, and controller applications.

Three LSI-11 system configurations comprise the LSI-11 portion of the PDP-11 family: LSI-11 system components, the PDP-11/03, and the PDP-11V03.

LSI-11 component systems include individual modules, backplane, etc., ordered as separate items. The user purchases only those items required for a specific application.

The PDP-11/03, a boxed version of the LSI-11, is designed for those that need a packaged microcomputer system. It consists of an LSI-11 microcomputer and 4K memory, a modular power supply, and a mounting box. It is an easy to use LSI-11-based microcomputer for system development or dedicated applications.

The PDP-11V03 is the latest addition to the LSI-11 family. It is a mass storage-based system, including the PDP-11/03, the RXV11 floppy disk system, a system cabinet and power distribution panel, either a VT52 DECscope or LA36 DECwriter terminal, RT-11 system software, and system diagnostics.

The remainder of this handbook contains detailed information for LSI-11 and PDP-11/03 systems. System specifications for the PDP-11V03 are also included.

This handbook contains all of the information previously contained in the *LSI-11, PDP-11/03 Processor Handbook* and *LSI-11, PDP-11/03 User's Manual*. Information has been expanded to include the latest LSI-11 options and module configurations. In addition, a section on system software is included. This information is those portions of the *PDP-11 Software Handbook* that are applicable to LSI-11 system applications. A brief description of the various sections comprising this handbook is provided below:

| Section | Description   |
|---------|---|
| I       | LSI-11 Family Hardware—Chapters containing a detailed description of LSI-11 system hardware specifications, installation, |

system configuration, and maintenance information. Instructions for loading programmable read only memory integrated circuits is also included.

- II Operation—Chapters describing the use of console mode ODT commands, REV11 (optional) commands, paper tape system operation, and RXV11 floppy disk-based system operation.
- III Processor—Chapters describing the processor's addressing modes, basic instruction set, optional EIS/FIS instruction set, and general programming techniques.
- IV System Software—Chapters containing a listing of software options, a description of operating systems, and floppy disk system software available for use on the LSI-11 system.
- V DIGITAL Services—Chapters on LSI-11 training, DECUS and maintenance services.



section 1  
lsi 11  
family  
hardware

**CHAPTER 1 INTRODUCTION**

**CHAPTER 2 SPECIFICATIONS**

**CHAPTER 3 THE LSI-11 BUS**

**CHAPTER 4 LSI-11 MODULE DESCRIPTIONS**

**CHAPTER 5 CONFIGURING LSI-11 MODULES**

**CHAPTER 6 INSTALLATION**

**CHAPTER 7 USING PROMs**

**CHAPTER 8 USER-DESIGNED INTERFACES**

**CHAPTER 9 MAINTENANCE**

## CHAPTER 1

# INTRODUCTION

### 1.1 GENERAL

Three basic LSI-11 microcomputer system configurations are available from DIGITAL: LSI-11 system components, the PDP-11/03 boxed LSI-11 system, and the PDP-11V03 system. The three basic system configurations are described below.

LSI-11 system components (Figure 1-1) include separate modules (printed circuit boards), backplanes, cables, etc., which are available separately. This allows the user to purchase only the system components required for a specific application. Original equipment manufacturers (OEM's), for example, can purchase LSI-11 system components to function in a specific computer-controlled system environment. In this application, the LSI-11 serves as low-cost, compact, flexible solution to the OEM's computer requirement. All of the essential processor hardware architecture (instruction set, addressing modes, registers, etc.) and software features of the PDP-11/40 computer are retained in the LSI-11 processor.

The PDP-11/03 (Figure 1-2) is a packaged version of the LSI-11 microcomputer. It includes a rack-mountable enclosure containing the LSI-11 processor, 4K memory, an LSI-11 bus-structured backplane, a power supply for the processor and options contained in the box, and a control panel (part of the power supply assembly) containing three indicators and three switches. All LSI-11 component system options can be used in PDP-11/03 systems. The PDP-11/03 is particularly useful in prototype system development.

The PDP-11V03 system (Figure 1-3) is a complete, ready-to-use floppy disk-based system that includes all necessary hardware factory-configured and installed, a terminal (VT52 or LA36), and RT-11 operating system software. RT-11 software includes single job and foreground/background operating system monitors. Included in the standard RT-11 software are all of the software tools normally required for assembly language programming. Floppy disk system diagnostics are also included with the system. Software options are described in Section IV.

### 1.2 LSI-11 SYSTEM COMPONENTS

#### 1.2.1 The LSI-11 Microcomputer

In general, all LSI-11 and PDP-11/03 systems include the KD11-F or KD11-J microcomputer. The KD11-F is a single 8.5 by 10 inch module that contains the LSI-11 microprocessor and a 4K by 16-bit semiconductor read/write memory (Figure 1-4). The KD11-J (Figure 1-5) uses the same microcomputer module as the KD11-F; however, it is supplied with the MMV11-A 4K by 16-bit core memory (Figure 1-7) instead of the semiconductor memory.

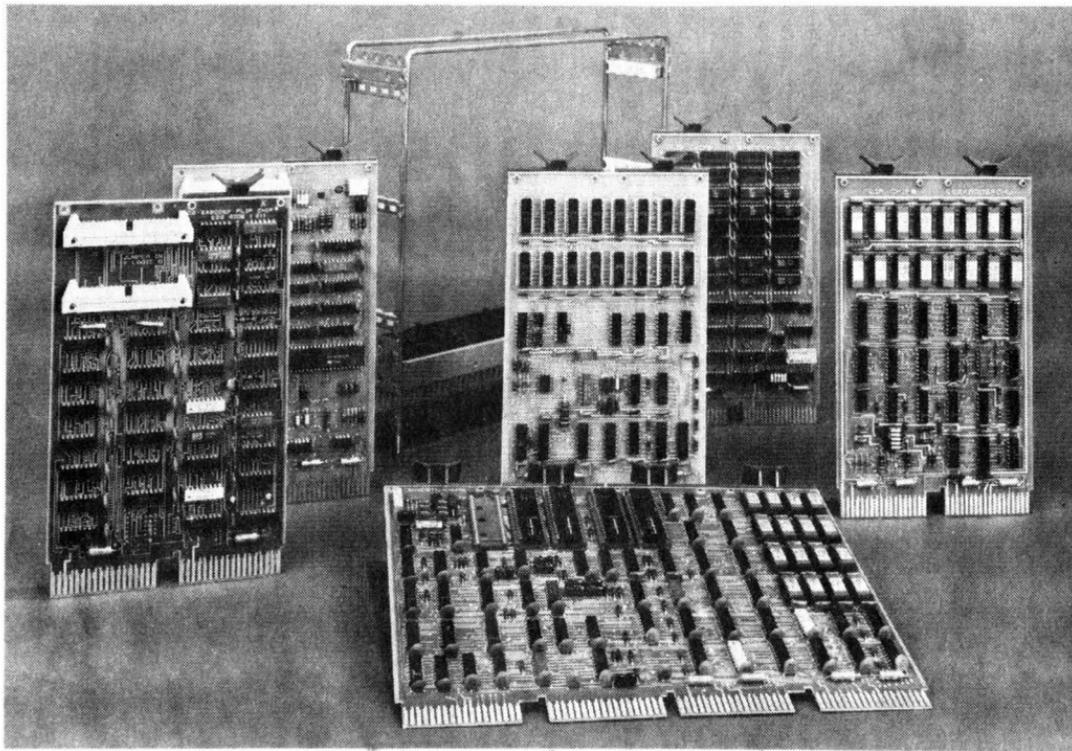


Figure 1-1 LSI-11 System Components

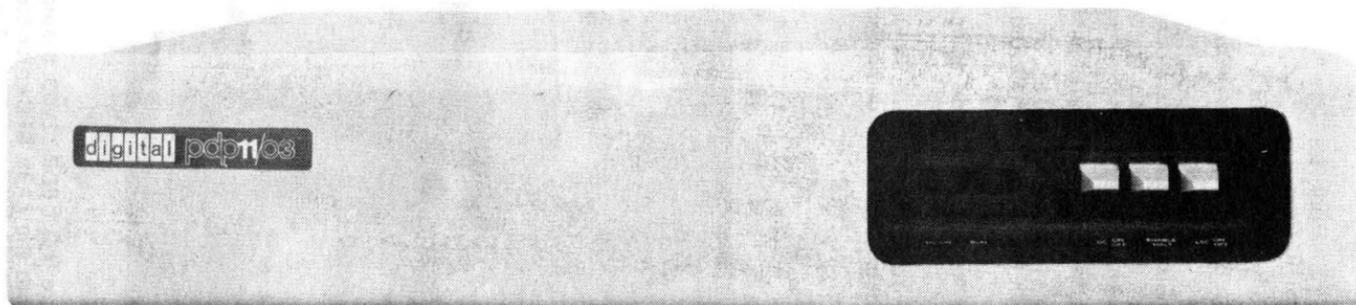


Figure 1-2 PDP-11/03 System

Each KD11-F features:

- A low-cost, powerful processor for integration into any small- or medium-sized computer system.
- Direct addressing of 32K 16-bit words or 64K 8-bit bytes ( $K = 1024$ ).
- Efficient processing of 8-bit characters without the need to rotate, swap, or mask.
- Asynchronous operation that allows system components to run at their highest possible speed; replacement with faster devices means faster operation without other hardware or software changes.
- A modular component design that provides ease and flexibility in configuring systems.
- Hardware memory stack for handling structured data, subroutines, and interrupts.
- Direct memory access for high data rate devices inherent in the bus architecture.
- Eight general-purpose registers that are available for data storage, pointers, and accumulators. Two are dedicated: SP and PC.

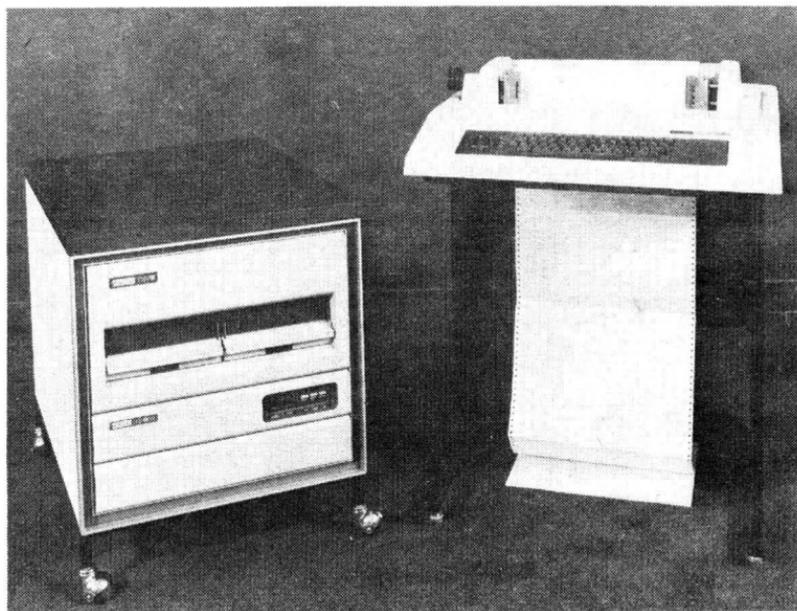


Figure 1-3 PDP-11V03 System  
(shown with the LA36 DECwriter terminal)

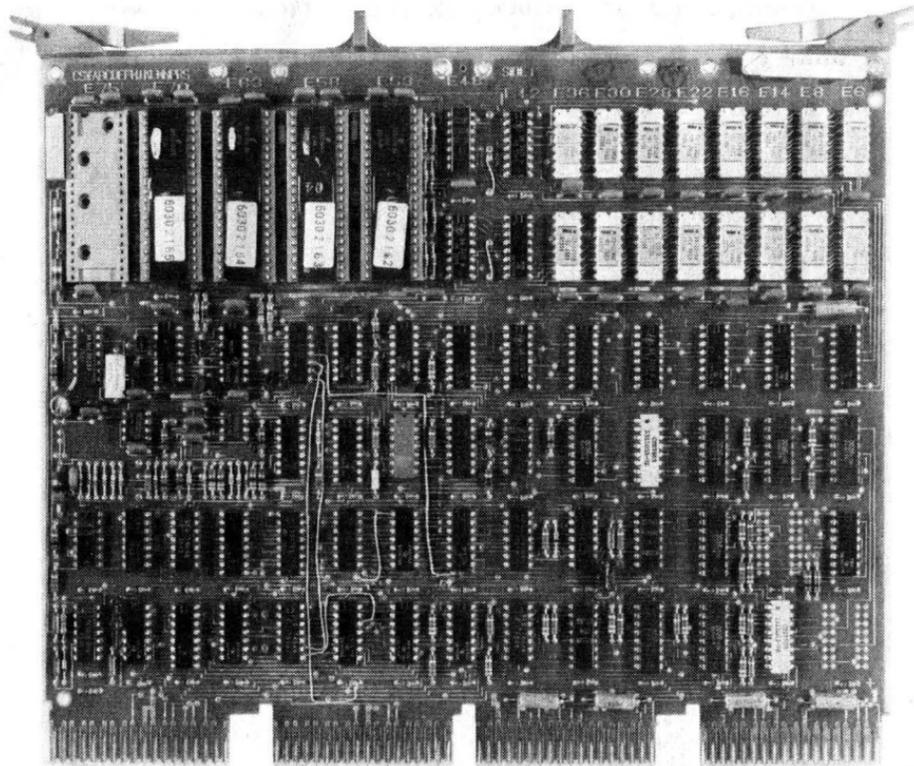


Figure 1-4 KD11-F LSI-11 Microcomputer and 4K Memory

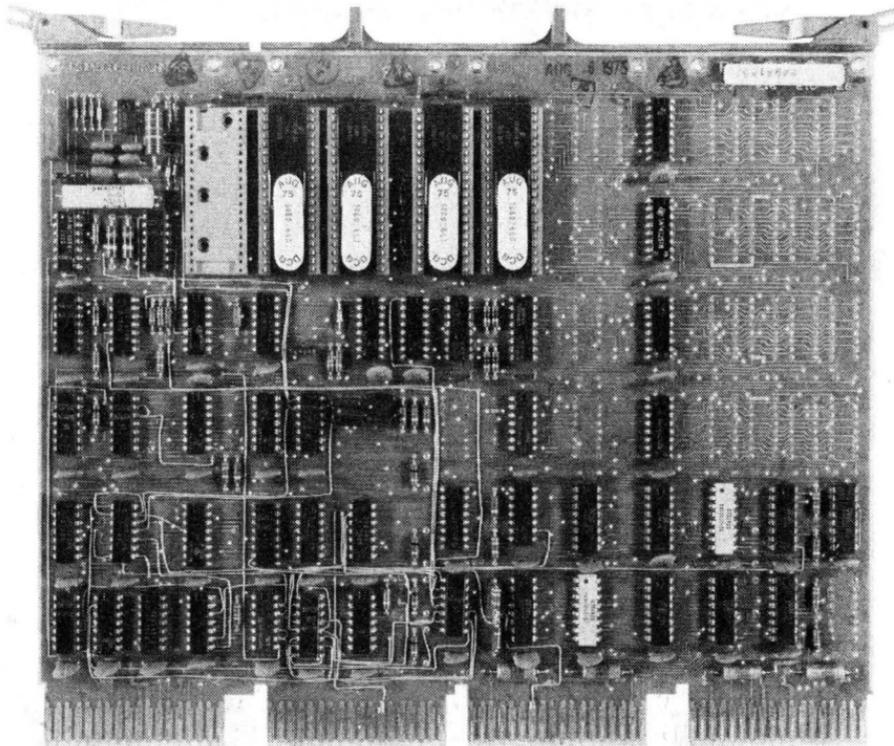


Figure 1-5 KD11-J LSI-11 Microcomputer (Processor Module)

- A bus structure that provides position-dependent priority as peripheral device interfaces are connected to the I/O bus.
- Fast interrupt response without device polling.
- A powerful and convenient set of programming instructions.
- A jumper-selected power-up mode that enables restart through a power-up vector, console Octal Debugging Technique (ODT) micro-code subset, or a bootstrap program.
- On-board 4K RAM.
- An ODT microprogram that controls all manual entry/display functions previously performed by a control panel through a serial ASCII device (optional) which is capable of transmitting and receiving ODT commands and data.
- Compact size (only 8.5 by 10 in.).

### 1.2.2 I/O Bus Concept

The LSI-11 I/O bus is simple, fast, and easy to use as an interface between the LSI-11 microcomputer, memory, and peripheral interface modules. It comprises 17 control lines and a 16-line data/address bus. All modules connected to this bus receive the same interface signals.

Address/data and control lines are open-collector lines that are asserted low. The microcomputer module is capable of driving six device locations along the bus. Peripheral interface or memory modules can be installed in any location along this bus.

Both address and data words (or bytes) are time multiplexed over 16 bus lines. For example, during a programmed data transfer, the LSI-11 microcomputer first asserts an address on the bus for a fixed time. After the address time has been completed, the processor performs either an input or output data transfer; the actual data transfer is asynchronous and requires a response from the addressed device. Bus synchronization and control signals provide this function.

Control signal lines include two daisy-chained grant signals that provide a priority-structured I/O system. The highest priority device is the module electrically closest to the KD11-F (or KD11-J) module. Higher priority devices pass a grant signal to lower priority devices only when not requesting service. (Memory options or devices that do not use these signals must connect the chain.)

The KD11-F contains a memory address register and 4K bank address decoder for its resident memory, which can be assigned to bank 0 or bank 1. Bank 7 is also decoded when addresses ranging from 160000 to 177777 are placed on the bus. These addresses are normally used for addressing nonmemory devices, thus eliminating the need for bank address decoding on peripheral device interface modules.

The bus provides a vectored interrupt capability for any interface device. Hence, device polling is not required in interrupt processing routines. This results in a considerable savings in processing time when many devices requiring interrupt service are interfaced along the bus. When a device receives an interrupt grant (acknowledge), the KD11-F inputs

the device's interrupt vector. The vector points to two addresses that contain a new processor status word and the starting address of the interrupt service routine for the device.

One bus signal line functions as an external event interrupt input to the KD11-F module. This signal line can be connected to a frequency source, such as a line frequency, and used as a line time clock (LTC) interrupt. A jumper on the KD11-F module enables or inhibits this function. When enabled, the device connected to this line has the highest interrupt priority external to the processor. Interrupt vector  $100_8$  is reserved for this function, and an interrupt request via the BEVNT line causes new PC and PS words to be loaded from locations  $100_8$  and  $102_8$ .

Memory refresh of dynamic MOS read/write memory is accomplished by bus signals. Refresh operation is controlled by either the processor module microcode or a direct memory access (DMA) device, such as the REV11-A or REV11-C.

The processor can be placed in the Halt mode by asserting one bus signal. This allows peripheral devices or a separate switch to invoke console ODT microcode operation.

Power-up/power-down sequencing is controlled by two bus signals. One signal, when in its true state, implies that primary power is normal. The second signal is in its true state when sufficient dc power is available (and voltages are normal) for normal system logic operation. These signals are produced by circuits contained in the H780 power supply (PDP-11/03 only) or by the user's system (circuits external to the LSI-11 system components).

DMA operation is controlled by three bus signals. Logic on the processor module, which is normally bus master, arbitrates DMA requests and grants bus mastership to the highest priority device requesting the bus. Priority is position-dependent through the use of a daisy-chained DMA grant signal.

### **1.2.3 Memory Options**

Memory options are available for expanding memory to 28K. The basic LSI-11 microcomputer is supplied with read/write memory. KD11-F's memory consists of a 4K dynamic MOS array that is physically located on the processor module. KD11-J's memory is a 4K magnetic core array contained on a separate module; the processor module supplied with the KD11-J contains no semiconductor memory components.

Optional memory modules include:

**MRV11-AA**—4K by 16-bit programmable read-only memory on an 8.5 by 5 in. module (Figure 1-6). Requires one device location on the I/O bus. Can be configured using either 256 by 4-bit or 512 by 4-bit field programmable or masked read-only memories (ROMs) for a maximum capacity of 2048 or 4096 16-bit words.

**MRV11-AC**—Unprogrammed 512 by 4-bit PROM integrated circuits for use in the MRV11-AA. The integrated circuits are field programmable and should be installed in sets of multiples of four.

**MMV11-A**—4K by 16-bit core memory on an 8.5 by 10 by 0.9 in. unit (Figure 1-7). Requires two device locations on the I/O bus when installed in the backplane (preferred location slots A4-D4). This allows a daughterboard (part of the MMV11-A) to extend slightly beyond the backplane without using additional device locations. If not installed in this location, the MMV11-A requires four device locations because of the additional module thickness (0.9 in. instead of 0.5 in. for all other modules).

**MSV11-B**—4K by 16-bit dynamic MOS read/write memory on an 8.5 by 5 in. module (Figure 1-8). Requires one device location on the I/O bus. Refresh is automatically performed by the KD11-F processor microcode or by an external device, such as the REV11-A and REV11-C.

#### **1.2.4 Peripheral Interface Options**

Four basic interface module options are available for use in LSI-11 systems. All options can be configured with unique device and vector addresses.

**DLV11**—Serial line unit interface on an 8.5 by 5 in. module (Figure 1-9). Requires one device location on the bus. Jumpers select crystal-controlled baud rates (50–9600 baud) and serial word format, including number of stop bits, number of data bits, and even, odd, or no parity bit. Optional interface cables include the BC05M, which connects the DLV11 to 20 mA current loop peripheral devices, and the BC05C, which connects the DLV11 to EIA-compatible devices (modems) via a Cinch DB-25P connector.

**DRV11**—General-purpose parallel line unit interface on an 8.5 by 5 in. module (Figure 1-10). Requires one device location on the bus. Two 40-pin connectors are included on the module for user interface application. One is the 16-bit input and the other is the 16-bit output.

**DRV11-B**—DMA interface on an 8.5 by 10 in. module (Figure 1-11). Requires two device locations on the bus. The interface is programmed by the processor to move variable length blocks of 16-bit data words to or from specified locations in system memory via the LSI-11 bus. Once programmed, no processor intervention is required to complete the data transfer. The DRV11-B is capable of transfer rates up to 250K, 16-bit words per second, and is capable of operating in burst modes and byte addressing. Switches are provided for selection of device addresses and interrupt vector address.

**DRV11-P**—LSI-11 bus foundation on an 8.5 by 10 in. module (Figure 1-12). Requires two device locations on the bus.

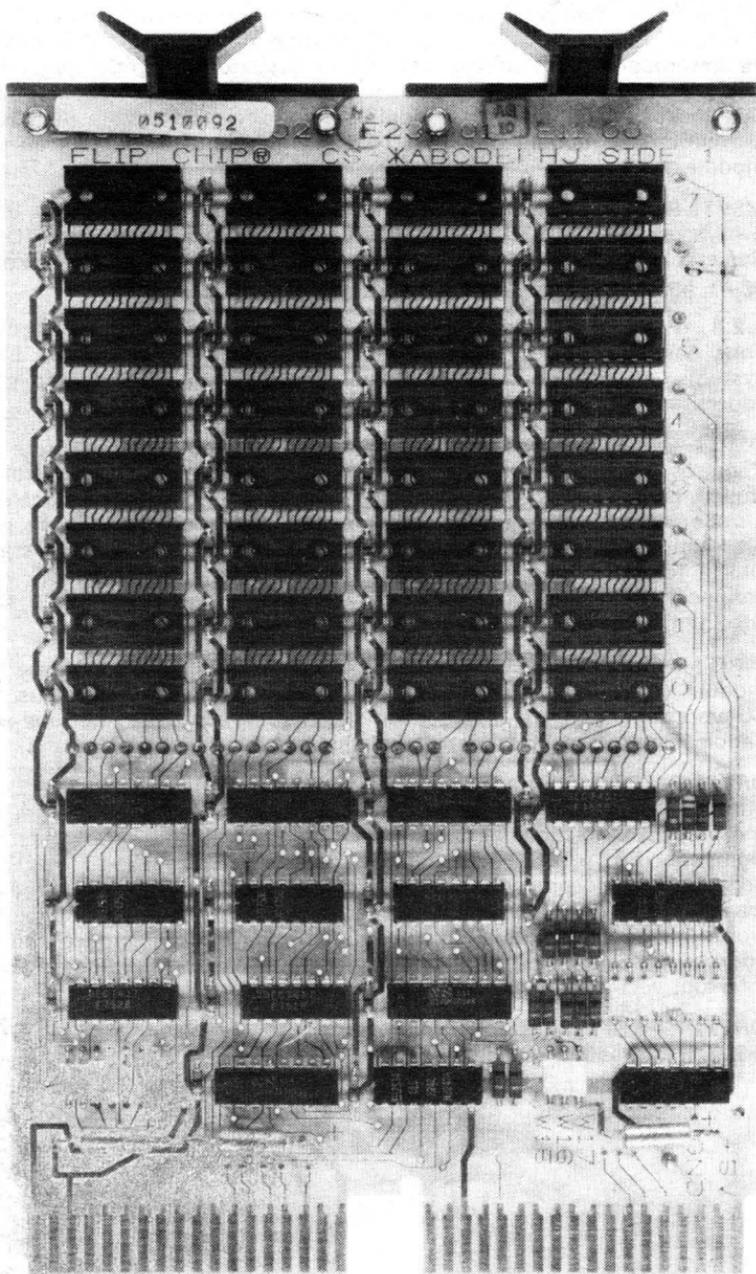


Figure 1-6 MRV11-AA 4K PROM Module

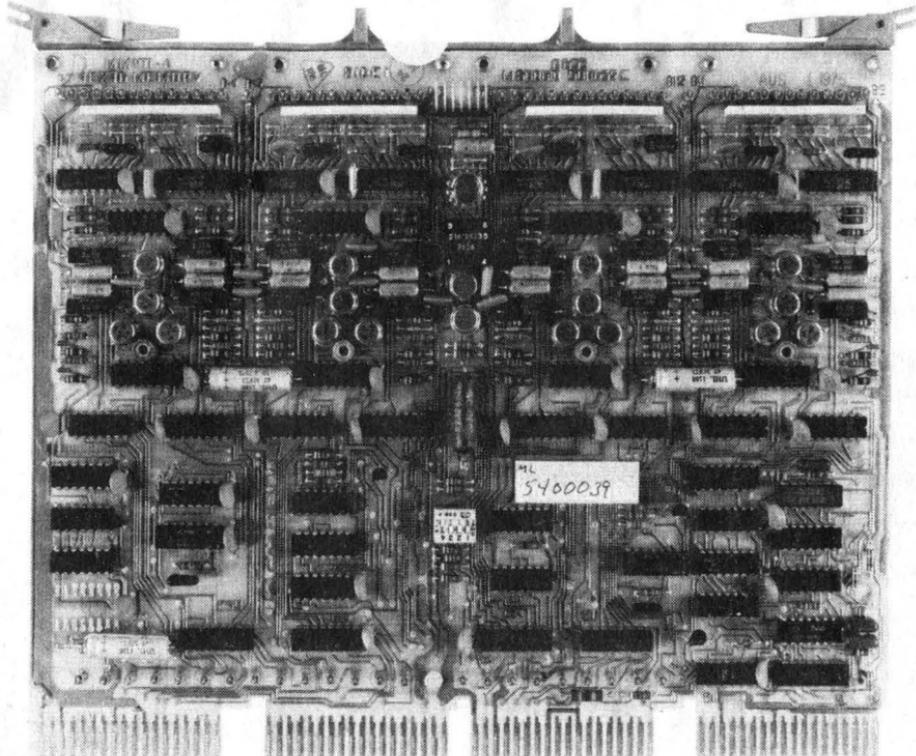


Figure 1-7 MMV11 4K Core Memory

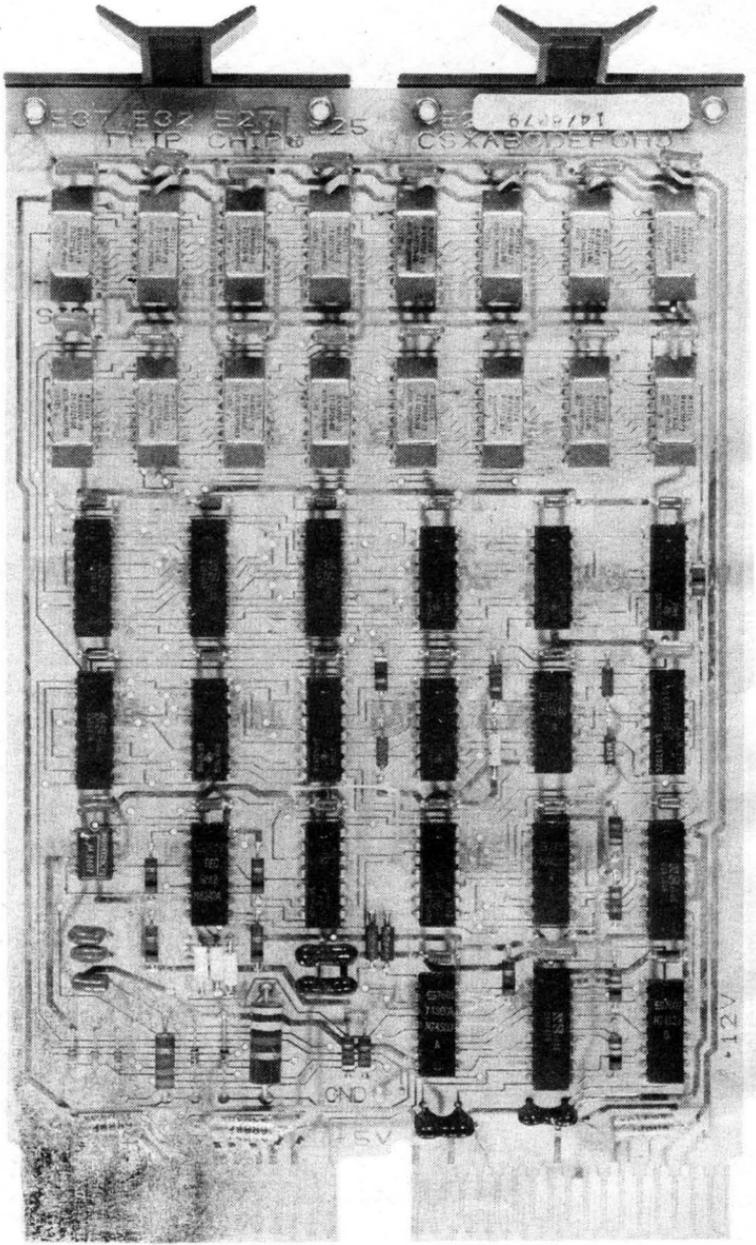


Figure 1-8 MSV11-B 4K Semiconductor Memory



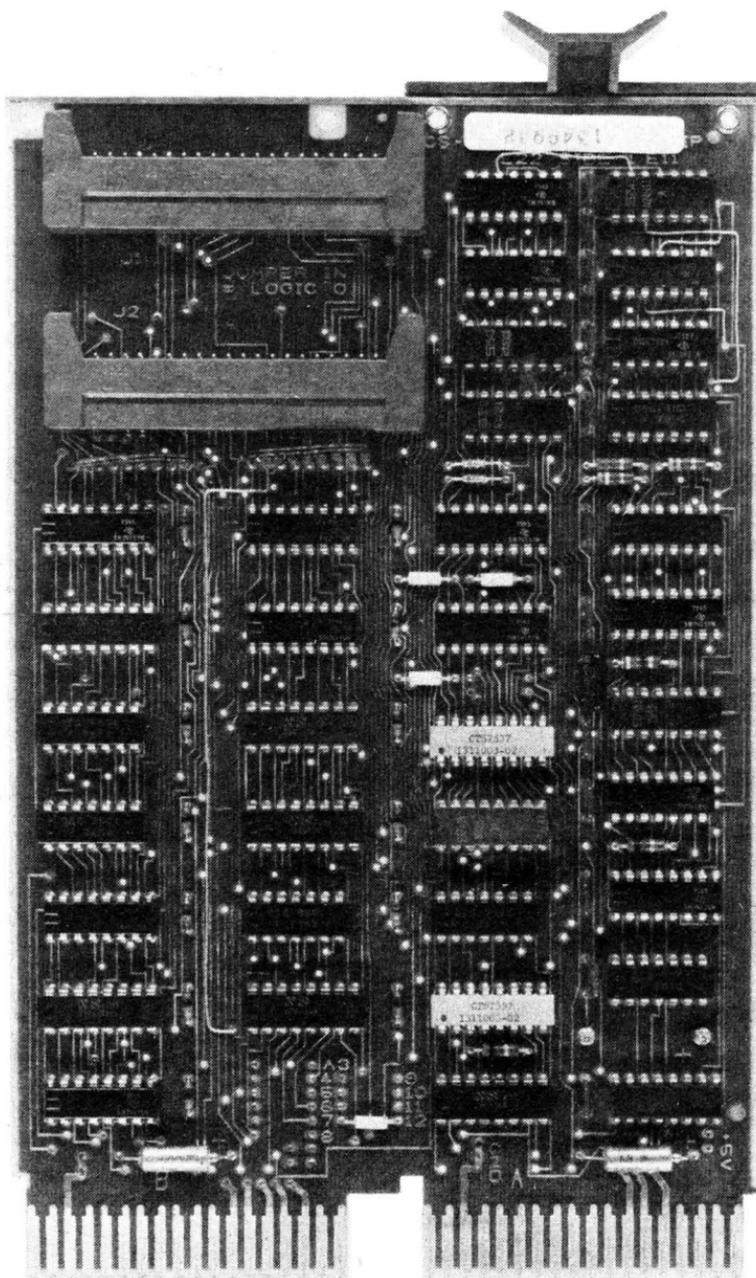


Figure 1-10 DRV11 Parallel Line Unit

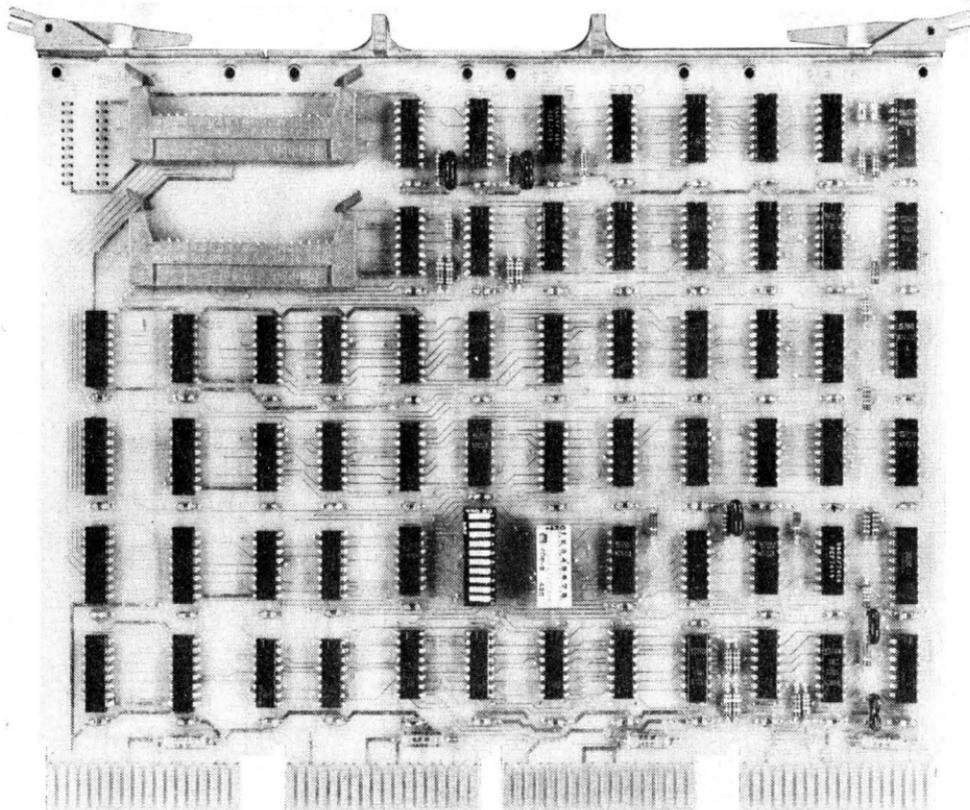


Figure 1-11 DRV11-B DMA Interface

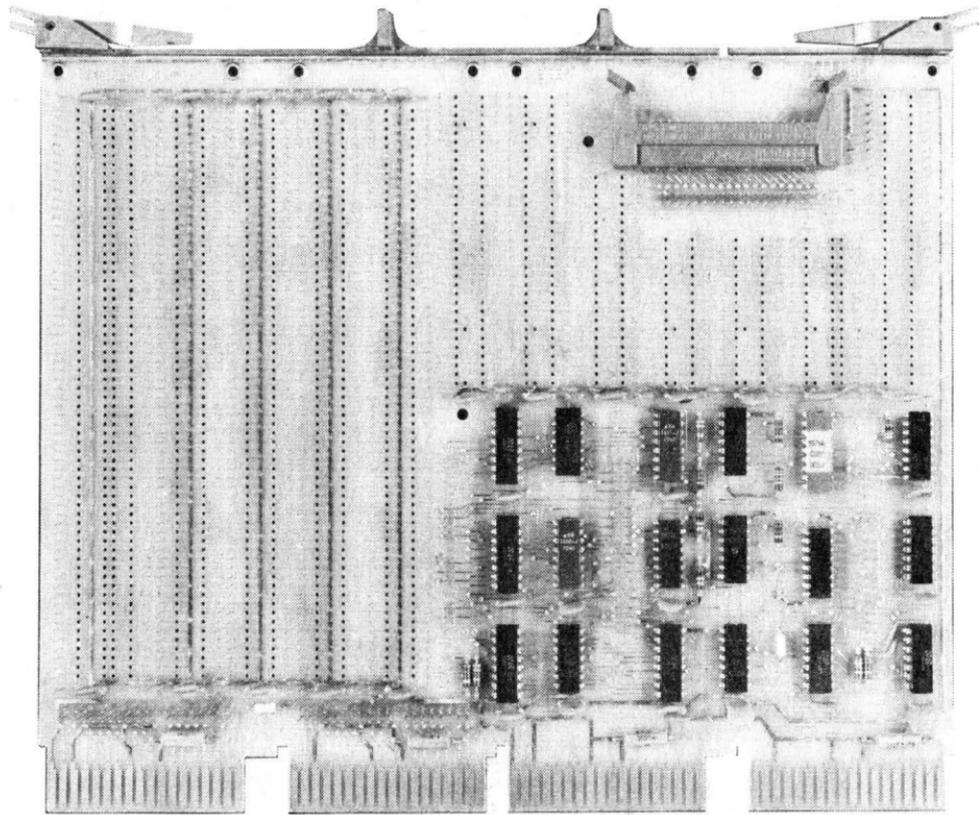


Figure 1-12 LSI-11 Bus Foundation Module

The DRV11-P is a versatile wire wrap module that contains the bus interface logic for operation with the LSI-11 or PDP-11/03 system and provides adequate board area for mounting and connecting integrated circuits (IC's) or discrete components. Because the bus interface logic is included, the module can be efficiently configured by the user to satisfy a variety of device interface logic applications.

A 40-pin connector, conveniently mounted at the board edge, facilitates the connection to a device through several cable assembly types available from DIGITAL.

Except for the bus interface connections, all signals and voltages are terminated to wire wrap pins for user connections. The bus control logic is provided with wire wrap test points to monitor the internal signals. The test points are spaced at 0.1 in. (0.254 cm) between pins to allow a 40-pin connector to be inserted over the wire wrap pins for automated test functions.

Approximately two-thirds of the surface area on the module consists of plated through holes, each connected to a wire wrap pin. The user can mount three different types of dual-in-line IC's or a variety of discrete components into the holes and connect the proper voltages and signals by wire wrapping leads on the board.

### **1.2.5 Backplane Options**

Two backplane options are available: the H9270 and the DDV11-B. The H9270 (Figure 1-13) is a four-by-four slot LSI-11 bus-structured backplane/card guide assembly. It can accept the processor module and up to six options. Power is applied to the backplane via a screw-terminal block located on one end of the assembly. Signal and power bus lines, provided by a printed circuit board, connect each option location as shown in Figure 1-14.

The DDV11-B (Figure 1-15) is an expanded version of the standard LSI-11 backplane (the H9270) for use when additional LSI-11 option module space is required. A nine-by-four slot section of this backplane is LSI-11 bus-structured and will accept the processor module, up to 15 option modules, and one TEV-11 bus terminator module. An additional nine-by-two slot section of the backplane is provided with power connections (+5 Vdc, -12 Vdc, and ground), only; wire wrap pins allow the user to interconnect the slots with appropriate signals.

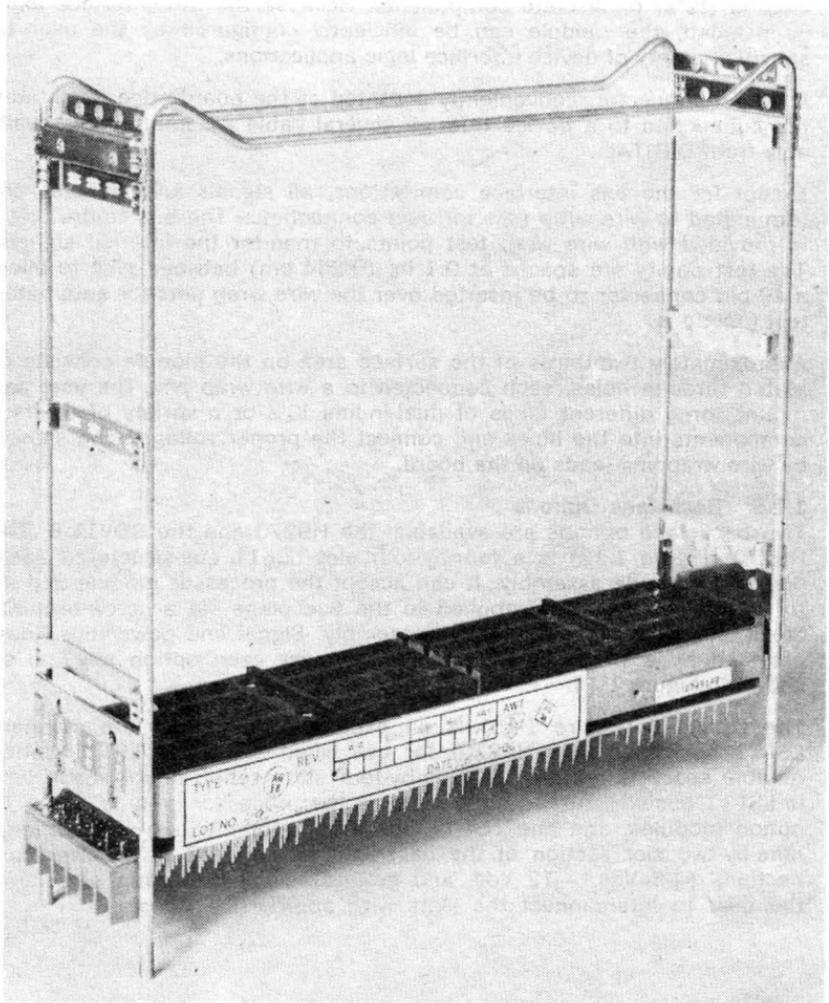


Figure 1-13 H9270 Backplane

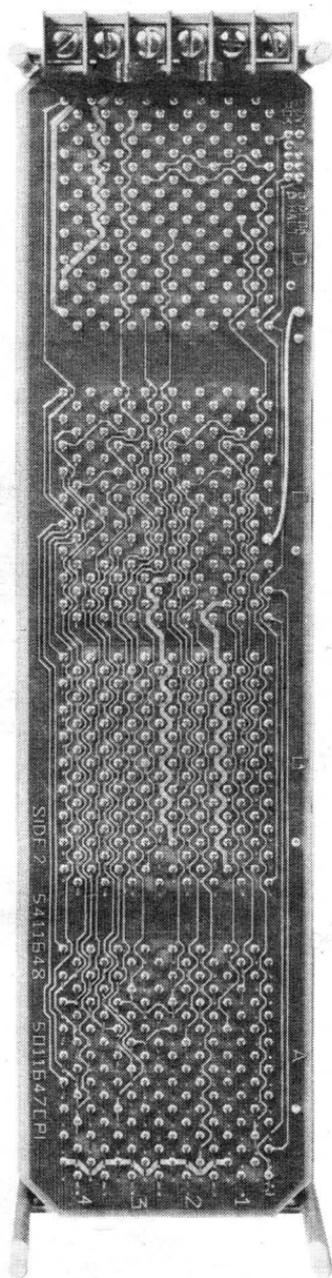


Figure 1-14 Printed Circuit Board

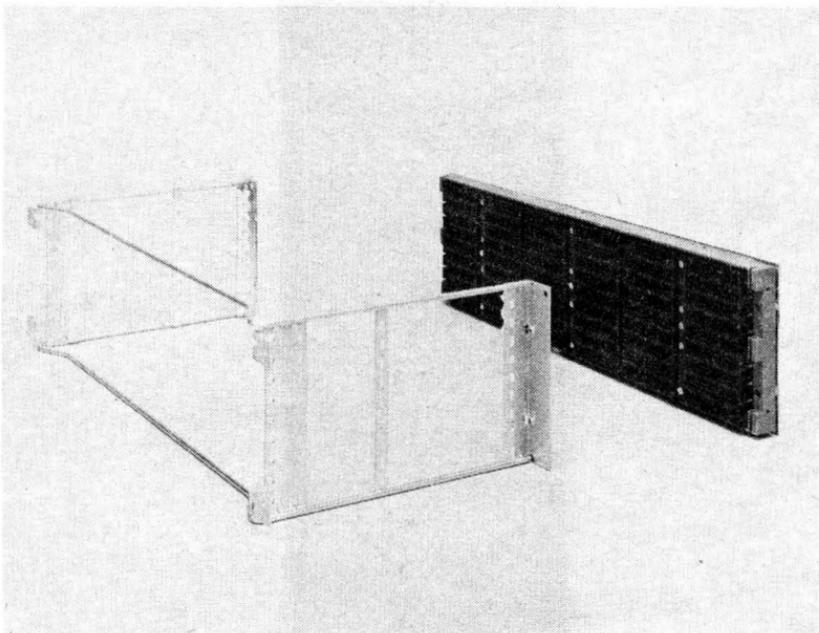


Figure 1-15 DDV11-B Expanded Backplane and  
H0341 Card Cage Assembly

An optional card cage, type H0341, is available for use with the DDV11-B backplane. It provides physical protection to modules and serves as a card guide. The card gage completely surrounds the DDV11-B on the module side of the backplane.

### 1.2.6 Expansion Boxes

BA11-ME and BA11-MF expansion boxes provide a most convenient means for expanding LSI-11, PDP-11/03, and PDP-11V03 systems. Each expansion box includes an H9270 backplane, an H780 power supply, which is capable of supplying sufficient power to all modules contained within the expansion box, and a rack-mountable enclosure. The enclosure is identical in size and design to the PDP-11/03; however, the switch/indicator control panel is not included (part of the H780 power supply in PDP-11/03 systems). The BA11-ME requires 115 Vrms, 50 or 60 Hz input power, and the BA11-MF requires 230 Vrms, 50 or 60 Hz input power.

### 1.2.7 Bus Accessory Options

Several LSI-11 bus accessory options are available for bus expansion, bus termination, DMA refresh, bootstrap ROM, and combinations of the preceding. The options can be used in both LSI-11 and PDP-11/03 applications. A summary of the options is provided below:

| Module No. | Figure | Includes  | System Functions  |
|------------|--------|---|---|
| REV11-A    | 1-16   | M9400-YA Module   | 120 $\Omega$ bus terminator, DMA refresh, bootstrap ROM.  |
| REV11-C    |        | M9400-YC Module   | DMA refresh, bootstrap ROM.   |
| TEV-11     | 1-17   | M9400-YB Module   | 120 $\Omega$ bus terminator.  |
| BCV1A-XX   | 1-18   | Two BC05L-XX cables, one M9400-YD module, and one M9401 module. | Bus expansion: two expansion cables and two backplane connector modules (M9400-YD and M9401). Normally used for expansion from second to third backplane in 3-backplane systems. (A TEV11 or REV11-A 120 $\Omega$ terminator must be installed in the last device slot in backplane 3.) |

#### NOTE

The -XX in BCV1A-XX and BCV1B-XX options denotes cable lengths. Options are available with cable lengths of 2, 4, 6, and 10 ft. For example, a BCV1A-06 includes two 6-ft cables. When the BCV1A and BCV1B options are used in a three backplane system, their lengths should differ by 4 ft.

|          |      |   |   |
|----------|------|---|---|
| BCV1B-XX | 1-19 | Two BC05L-XX cables, one M9400-YE module, and one M9401 module. | Bus expansion: 250 $\Omega$ terminator (M9400-YE), two expansion cables, backplane connector (M9401). Normally used for expansion from first to second backplane in 2 or 3 backplane systems. |
|----------|------|---|---|

The REV11-A and REV11-C options contain programs stored in the ROM. These programs include processor and memory diagnostics, bootstrap programs for the RV11 floppy disk system, and absolute loader programs for paper tape readers. (Note that the diagnostic programs are short tests to provide a quick check of system operation; these tests do not replace system diagnostics used for maintenance purposes.) A command set used for running the programs is described in Section II, Chapter 3.

TEV11 (or REV11-A), BCV1A, and BCV1B options are used for system expansion in multiple backplane systems. In addition, the REV11-A or TEV11 can be used to terminate the DDV11-B backplane (required when more than six option modules are installed on the backplane).

### 1.3 THE PDP-11/03 SYSTEM

The PDP-11/03 is a packaged version of the LSI-11 system. The system includes the LSI-11 processor and 4K memory, an H780 power supply, and an H9270 backplane factory installed in a rack-mountable enclosure. System models are listed below:

| Model        | Primary Power | Memory Type   |
|--------------|---------------|---------------|
| PDP-11/03-EA | 115 V, 60 Hz  | Semiconductor |
| PDP-11/03-EB | 230 V, 50 Hz  | Semiconductor |
| PDP-11/03-FA | 115 V, 60 Hz  | Core          |
| PDP-11/03-FB | 230 V, 50 Hz  | Core          |

### 1.4 THE PDP-11V03 SYSTEM

The PDP-11V03 is a complete, ready to use system. It includes a PDP-11/03 packaged LSI-11 system, an additional 4K memory (8K system memory, total), RXV11 dual floppy disk system, DLV11 serial line unit interface for the console terminal, and either a VT52 DECscope terminal or an LA36 DECwriter II terminal. All system components, except the console terminal, are contained in a system cabinet. The cabinet can remain on the floor and moved, as desired (casters are included), placed under a table, or placed on top of a table. Detailed information on the PDP-11V03 system is contained in the *PDP-11V03 System Manual*. System models and specifications are included in this handbook, Section I, Chapter 2.

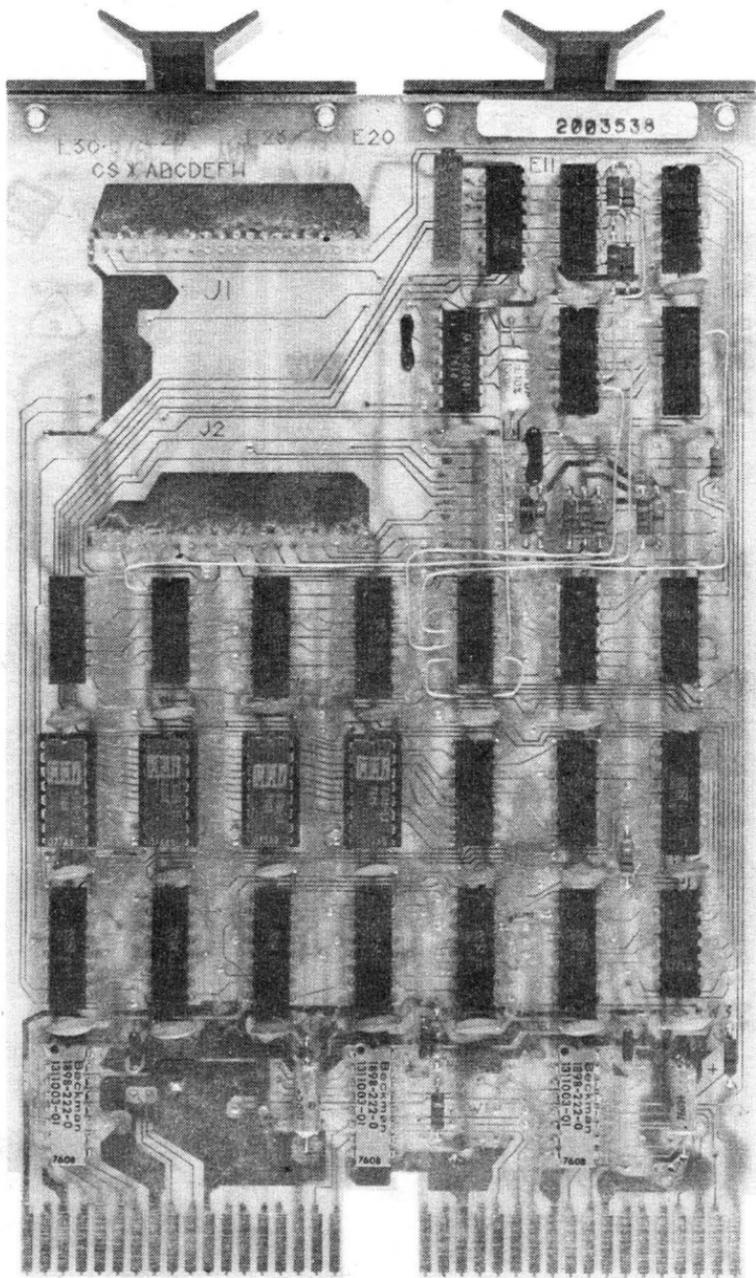


Figure 1-16 REV11-A Bus Terminator, DMA Refresh, Bootstrap ROM Module

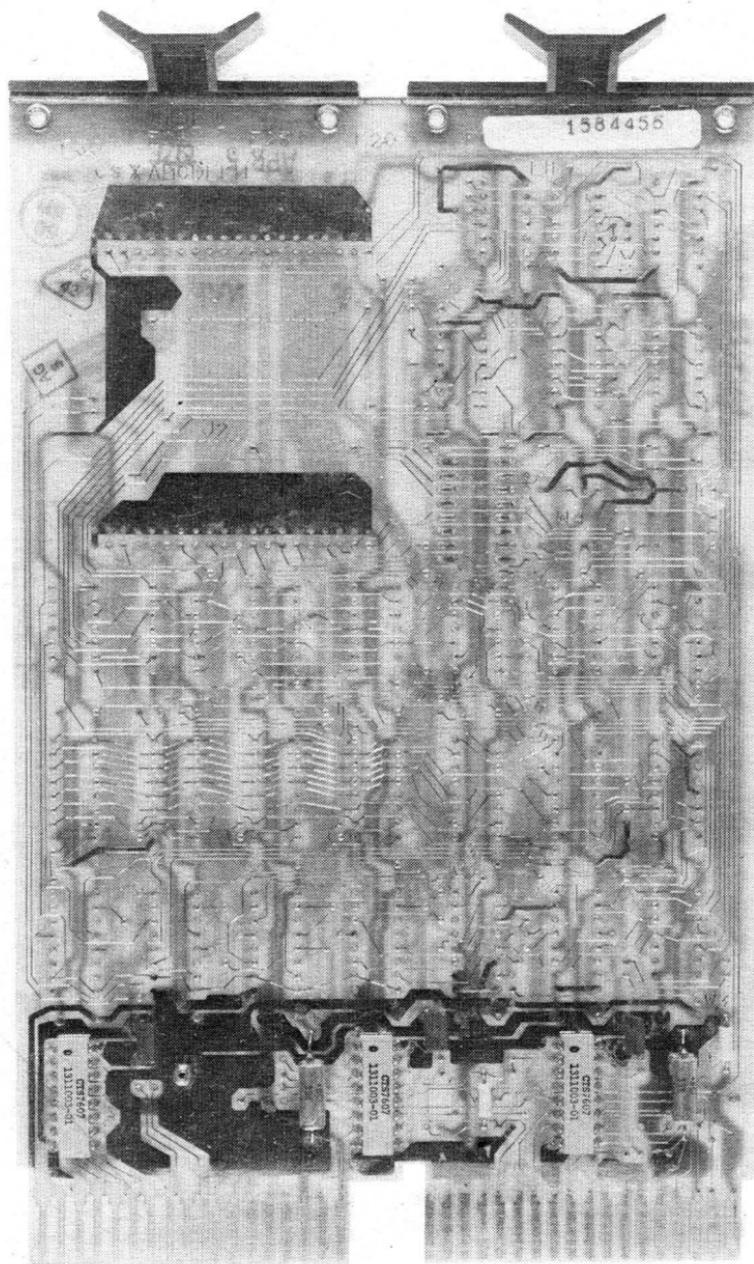


Figure 1-17 TEV11 Bus Terminator Module

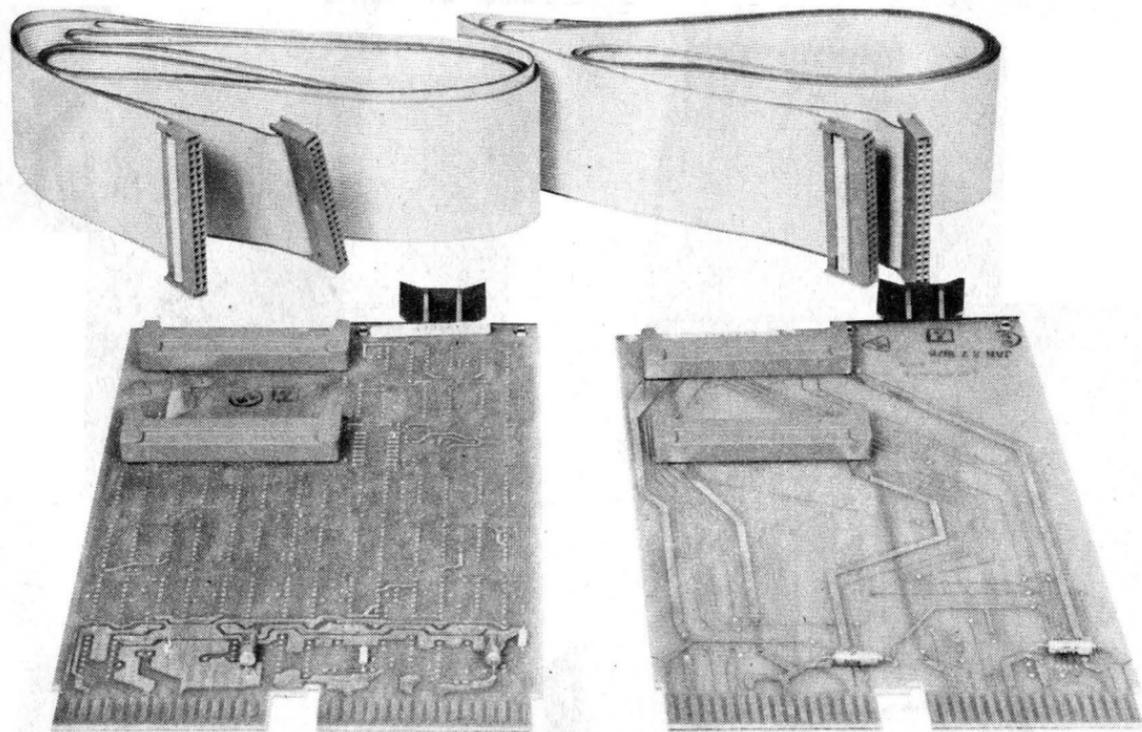


Figure 1-18 BCV1A Bus Expansion Option

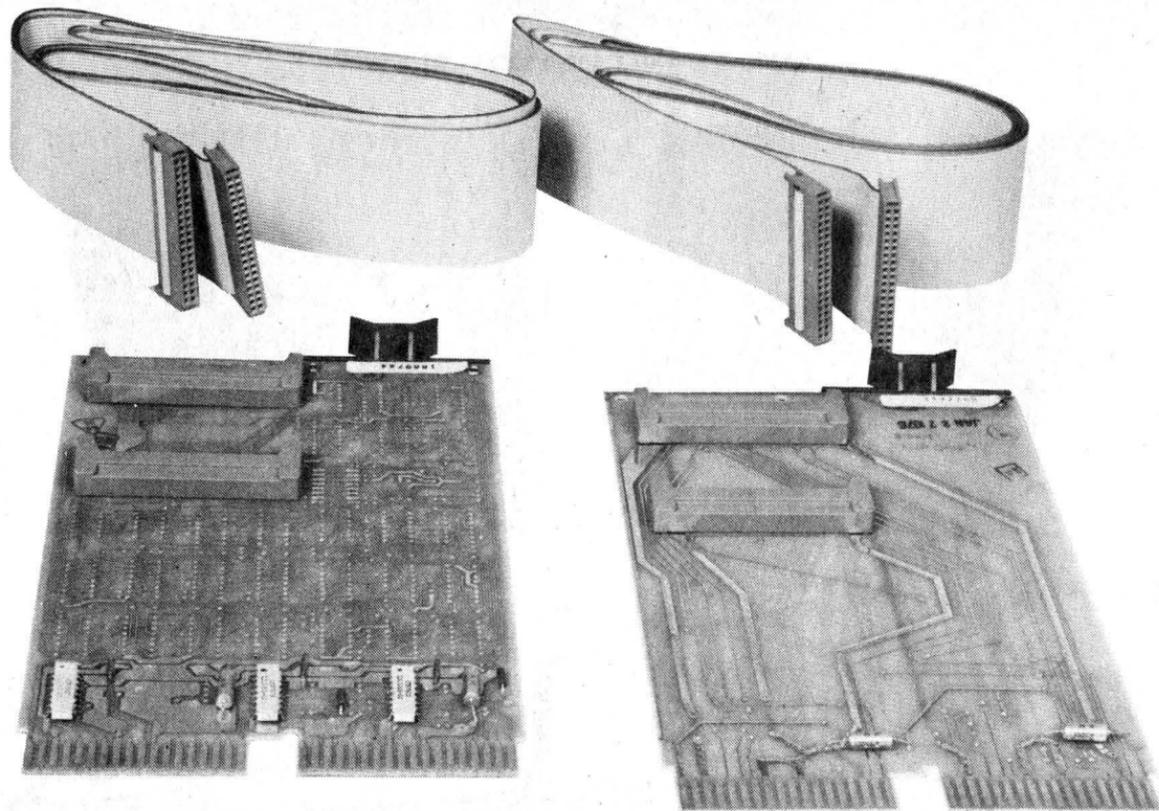


Figure 1-19 BCV1B Bus Expansion Option

## SPECIFICATIONS

**2.1 GENERAL**

This chapter contains applicable electrical, mechanical, and environmental specifications for LSI-11 systems. LSI-11 system components (modules and backplanes), PDP-11/03, and PDP-11V03 systems are covered in separate paragraphs.

**2.2 LSI-11 SYSTEM COMPONENTS****2.2.1 Modules**

Table 2-1 lists the electrical and mechanical specifications of the LSI-11 modules. All LSI-11 modules will operate at temperatures of 41° F to 122° F (5° C to 50° C) with a relative humidity of 10% to 95% (no condensation), with adequate airflow across the modules. When operating at the maximum temperature (122° F or 50° C), air flow must maintain the inlet to outlet air temperature rise across the modules to 12.5° F (7° C) maximum. Air flow should be directed across the modules as shown in Figure 2-1, and as described in Paragraph 6.7.

**2.2.2 Backplane Options**

**2.2.2.1 H9270 Backplane**—The H9270 backplane will accept the KD11-F processor module and up to six option modules, or the KD11-J processor and core memory modules and up to two or four option modules, depending on the location of the core memory in the backplane. When used for bus expansion in multiple backplane systems, the H9270 provides space for up to six option modules, plus the required expansion cable connector module(s) and/or terminator module.

Mounting dimensions for the H9270 are shown in Figure 2-1. Figure 2-3 illustrates possible methods of mounting the backplane.

Backplane pinning and signal functions are included in Chapter 3. Option positions are shown in Figure 2-2. Numbers indicate device interrupt and/or DMA priority; lowest numbered positions receive the highest priority.

Table 2-1 Module Specifications

| Option Desig. | Module No(s).            | Description  | Power Requirements    |                       | Height         | Size—in. (cm) |               |
|---------------|--------------------------|--|-----------------------|-----------------------|----------------|---------------|---------------|
|               |                          |  | +5 V ±5%              | +12 V ±3%             |                | Length*       | Width         |
| KD11-F        | M7264                    | LSI-11 processor and 4K × 16 RAM   | 1.8 A<br>(2.4 A max.) | 0.8 A<br>(1.6 A max.) | 10.5<br>(26.6) | 8.9<br>(22.8) | 0.5<br>(1.27) |
| KD11-J        | M7264-YA<br>H223<br>G653 | LSI-11 processor and 4K × 16 core memory unit  | 6.4 A<br>(9.0 A max.) | 1.2 A<br>(1.5 A max.) | 10.5<br>(26.6) | 8.9<br>(22.8) | 0.5<br>(1.27) |
|               |                          |  |                       |                       | 10.5<br>(26.6) | 8.9<br>(22.8) | 0.9<br>(2.29) |
| DLV11         | M7940                    | Serial line unit interface   | 1.0 A<br>(1.6 A max.) | —                     | 5.2<br>(13.2)  | 8.9<br>(22.8) | 0.5<br>(1.27) |
| DRV11         | M7941                    | Parallel line unit interface   | 0.9 A<br>(1.6 A max.) | —                     | 5.2<br>(13.2)  | 8.9<br>(22.8) | 0.5<br>(1.27) |
| DRV11-B       | M7950                    | DMA interface  | 1.9 A                 | —                     | 10.5<br>(26.6) | 8.9<br>(22.8) | 0.5<br>(1.27) |
| DRV11-P       | M7948                    | LSI-11 Foundation module   | 1.0 A                 | —                     | 10.5<br>(26.6) | 8.9<br>(22.8) | 0.5<br>(1.27) |
| MRV11-AA      | M7942                    | 4K × 16 read-only memory (less PROM integrated circuits) (with 32 512 × 4 PROM integrated circuits (MRV11-AC)) | 0.4 A<br>(0.6 A max.) | —                     | 5.2<br>(13.2)  | 8.9<br>(22.8) | 0.5<br>(1.27) |
| MSV11-B       | M7944                    | 4K × 16 read-write MOS memory  | 2.8 A<br>(4.1 A max.) | —                     |                |               |               |
|               |                          |  | 0.6 A<br>(1.2 A max.) | —                     | 5.2<br>(13.2)  | 8.9<br>(22.8) | 0.5<br>(1.27) |

\* Length as stated is approximate, and includes module handles. Actual module length is 8.5 in. (21.6 cm).

Table 2-1 Module Specifications (Cont.)

| Option Desig. | Module No(s).     | Description  | Power Requirements      |                 | Size—in. (cm)                |               |               |
|---------------|-------------------|--|-------------------------|-----------------|------------------------------|---------------|---------------|
|               |                   |  | +5 V $\pm 5\%$          | +12 V $\pm 3\%$ | Height                       | Length*       | Width         |
| MMV11-A       | H223              | 4K $\times$ 16 core memory (standby power) (operating power) | 3.0 A                   | 0.2 A           | 10.5                         | 8.9           | 0.9           |
|               | G653              |  | 7.0 A                   | 0.6 A           | (26.6)                       | (22.8)        | (2.29)        |
| REV11-A       | M9400-YA          | 120 $\Omega$ terminator, DMA refresh, bootstrap ROM          | 1.64 A<br>(2.24 A max.) | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
| REV11-C       | M9400-YC          | DMA refresh, bootstrap ROM                                   | 1.0 A<br>(1.88 A max.)  | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
| REV11-H       | M9400-YH          | DMA refresh, bootstrap ROM                                   | 1.0 A<br>(1.88 A max.)  | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
| TEV11         | M9400-YB          | 120 $\Omega$ terminator                                      | 0.54 A<br>(0.70 A max.) | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
| BCV1A-XX      | M9400-YD<br>M9401 | Bus expansion option:<br>2 cables, 2 modules                 | —                       | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
|               |                   |  |                         |                 | (dimensions for each module) |               |               |
| BCV1B-XX      | M9400-YE<br>M9401 | Bus expansion option:<br>2 cables, 2 modules                 | 0.29 A<br>(0.37 A max.) | —               | 5.2<br>(13.2)                | 8.9<br>(22.8) | 0.5<br>(1.27) |
|               |                   |  |                         |                 | (dimensions for each module) |               |               |

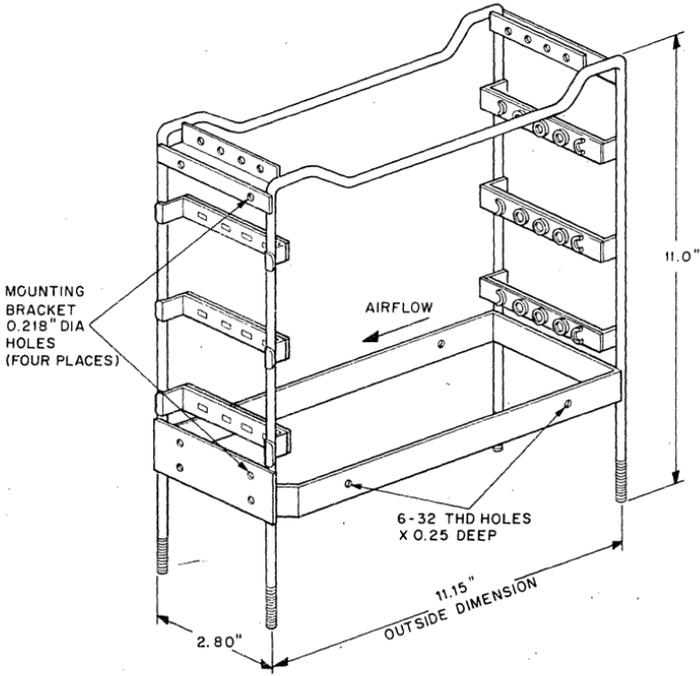


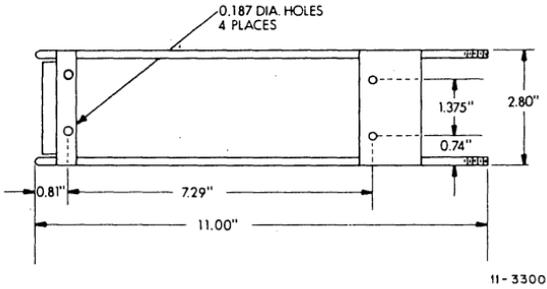
Figure 2-1 H9270 Backplane Mounting

|   | A          | B | C          | D   |
|---|------------|---|------------|---|
| 1 | POSITION 1 |   | POSITION 2 | ← POSITIONS 1 & 2 ARE NORMALLY USED FOR THE PROCESSOR MODULE IN SINGLE BACKPLANE SYSTEMS AND THE FIRST BACKPLANE IN MULTIPLE BACKPLANE SYSTEMS. |
| 2 | POSITION 4 |   | POSITION 3 |   |
| 3 | POSITION 5 |   | POSITION 6 |   |
| 4 | POSITION 8 |   | POSITION 7 |   |

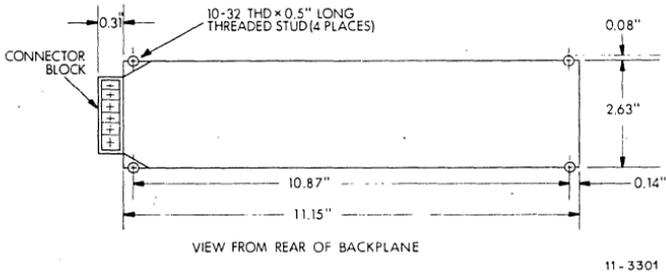
11-3299

Figure 2-2 H9270 Option Positions

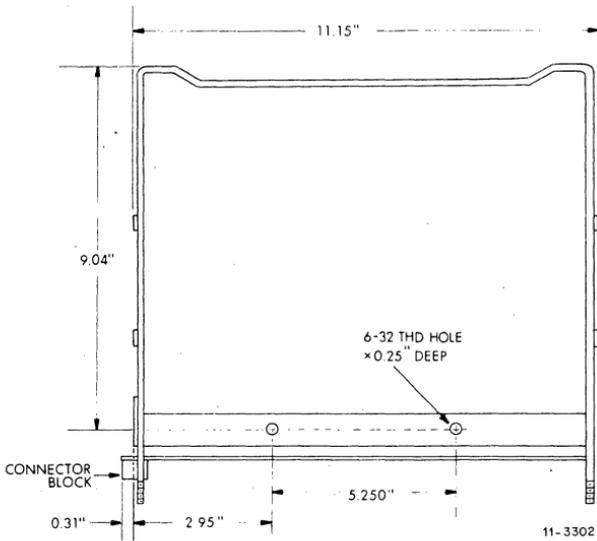
**SIDE MOUNTING**



**REAR MOUNTING**



**TOP AND BOTTOM MOUNTING**



**Figure 2-3 Backplane Mounting**

**2.2.2.2 DDV11-B Backplane**—The DDV11-B backplane will accept the KD11-F processor module and up to 15 option modules, plus a terminator module; or the KD11-J processor module and core memory modules and up to 11 option modules, plus a terminator module. Mounting and overall dimensions for the DDV11-B are shown in Figure 2-4. Clearance for the H0341 card cage assembly is shown by the dotted outline; the H0341 is a separate option and it is not supplied with the DDV11-B option.

In addition to providing space for mounting LSI-11 modules, 18 user-defined slots are provided for custom applications. These slots are located in E and F slots, which are adjacent to LSI-11 bus-structured slots A through D, as shown in Figure 6-3. Option positions shown in the figure are numbered to show device interrupt and/or DMA priority; lowest-numbered positions receive the highest priority. Backplane pinning and signal functions are included in Chapter 3.

**2.2.3 BA11-ME and BA11-MF Expansion Boxes**

BA11-ME and BA11-MF expansion boxes provide a most convenient means for expanding LSI-11, PDP-11/03, and PDP-11V03 systems. Each expansion box includes an H9270 LSI-11 bus-structured backplane, an H780 power system, and an enclosure that is identical to the PDP-11/03 enclosure. However, the front panel, including three indicators and three switches, is not included. Refer to PDP-11/03 Figures 2-5 and 2-6 for mechanical details. Electrical specifications are listed below:

**AC Input Power:**

BA11-ME 100-127 Vrms, 50±1 Hz or 60±1 Hz, 400 W maximum

BA11-MF 200-254 Vrms, 50±1 Hz or 60±1 Hz, 400 W maximum

**DC Output Power:**

+5 Vdc ±3%, 0-18 A load (static and dynamic)

+12 Vdc ±3%, 0-3.5 A load (static and dynamic)

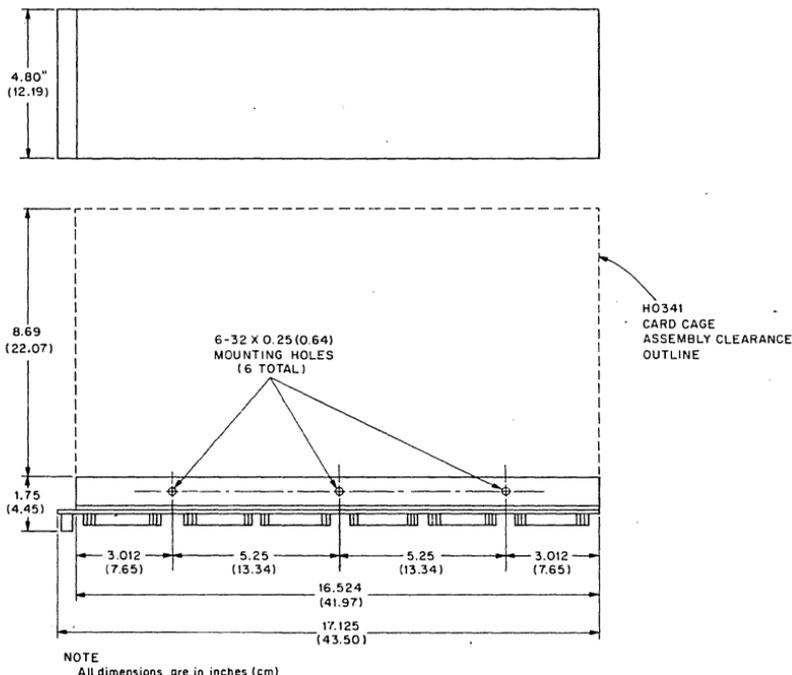
Maximum output power: 120 W (total)

**2.3 PDP-11/03 SYSTEM**

**2.3.1 General**

The PDP-11/03 system includes an LSI-11 processor and 4K memory (KD11-F or KD11-J), an H780 power supply, an H9270 backplane, and a rack-mountable enclosure. The PDP-11/03 is available in four models, providing a selection of 115 or 230 V primary power and 4K MOS or 4K core memory. The models are listed below:

| Model        | Description   |
|--------------|---|
| PDP-11/03-EA | KD11-F processor and resident 4K memory (M7264), H780-A power supply (115 V input), H9270 backplane, BA11 enclosure.            |
| PDP-11/03-EB | Same as above except the H780-A power supply is replaced with the 230 V input H780-B power supply.                              |
| PDP-11/03-FA | KD11-J processor and 4K core memory (M7264-YA, H223, G653), H780-A power supply (115 V input), H9270 backplane, BA11 enclosure. |



11-4291

Figure 2-4 DDV11-B Outline and Mounting Dimensions

PDP-11/03-FB Same as above except the H780-A power supply is replaced with the 230 V input H780-B power supply.

### 2.3.2 H780 Power Supply

The H780 power supply provides sufficient dc power for modules installed in one H9270 backplane. Cooling air is provided for system components by fans included in the H780 power supply.

Dc output voltages and currents are listed below. The user can calculate actual dc power requirements for system components by obtaining power requirements for the processor module and options in Table 2-1.

| Voltage         | Current | dc Power                |
|-----------------|---------|-------------------------|
| +5 V $\pm 3\%$  | 0-18 A  | } 120 W (total) maximum |
| +12 V $\pm 3\%$ | 0-3.5 A |                         |

Input power requirements are as follows:

#### Models

#### Input Power

PDP-11/03-EA, 100-127 Vrms,  $50 \pm 1$  Hz or  $60 \pm 1$  Hz, 400 W maximum  
 PDP-11/03-FA

PDP-11/03-EB, 200-254 Vrms, 50±1 Hz or 60±1 Hz, 400 W maximum  
PDP-11/03-FB

A front panel is included on the H780 power supply that contains three indicators and three switches. They are described below:

| Switches    | Function   |
|-------------|--|
| DC ON/OFF   | Enables or disables H780 dc voltage outputs. (Does not remove ac power from power supply circuits and fans.)   |
| HALT/ENABLE | HALT position: Halts the processor and enables console ODT microcode operation. Single-instruction execution can be performed when this switch remains in the HALT position.<br><br>ENABLE position: Enables the processor Run mode. Programs are started by entering a "G" or "P" command via the console terminal (or automatically, as appropriate, during power-up). |
| LTC ON/OFF  | Enables or disables the Line Time Clock signal generated by the H780. This signal generates vectored interrupts at the line frequency (50 or 60 Hz).   |

| Indicators | Function   |
|------------|--|
| DC ON      | Lit when dc voltages are supplied to the backplane.  |
| RUN        | Lit when the processor is executing programs (Run mode). Not lit when the processor is in the Halt mode. |
| —          | Spare  |

In addition to the front panel controls and indicators, an AC ON/OFF switch and fuse are included on the rear of the unit. The switch normally remains in the ON position, allowing an external circuit (user-supplied) to control application of primary power. Overload protection is provided by a fuse, as listed below:

| Model                                     | Fuse Type      |
|---|----------------|
| PDP-11/03-EA and<br>PDP-11/03-FA (H780-A) | 5 A, fast blow |
| PDP-11/03-EB and<br>PDP-11/03-FB          | 3 A, fast blow |

#### NOTE

Detailed H780 power supply specifications are included in Paragraph 4.12.2.

### 2.3.3 Environmental Specifications

Temperature: 41° F-104° F (5° C-40° C)  
Derate at 11° F (6° C)/1000 ft at altitudes above 8000 ft

Relative Humidity: 10% to 95% (no condensation)

### 2.3.4 Mechanical Specifications

The PDP-11/03 (Figure 2-5) is designed with a removable front panel.

Removing the front panel exposes the LSI modules and cables. This enables replacement or installation of a module from the front of the PDP-11/03. The 11/03 power supply is located on the right-hand side of the PDP-11/03 when viewed from the front. The power supply contains three front panel switches and indicators, which are accessible through a cutout in the front panel. Therefore, when the front panel is removed, the lights and switches are still attached and functional.

The PDP-11/03 is designed to mount in a standard 19 in. cabinet (Figure 2-6). A standard 19 in. cabinet has two rows of mounting holes in the front, spaced  $18\frac{3}{16}$  in. apart. The holes are located  $\frac{1}{2}$  in. or  $\frac{5}{8}$  in. apart from each other. Standard front panel increments are  $1\frac{3}{4}$  in.

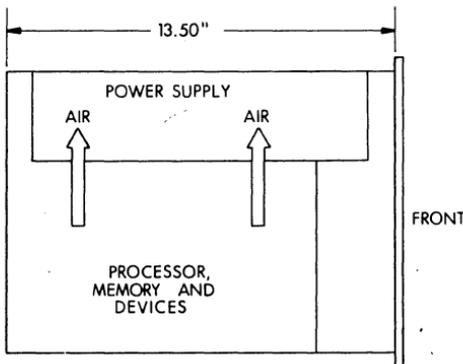
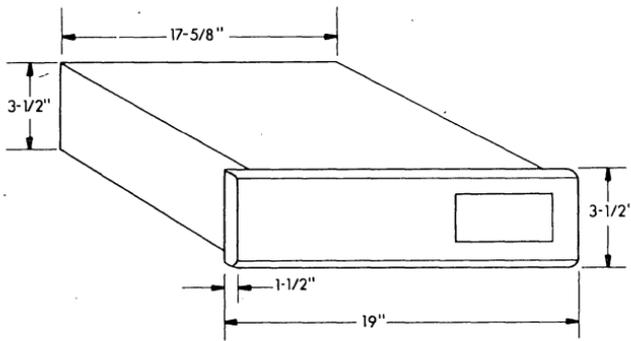


Figure 2-5 PDP-11/03 Assembly Unit

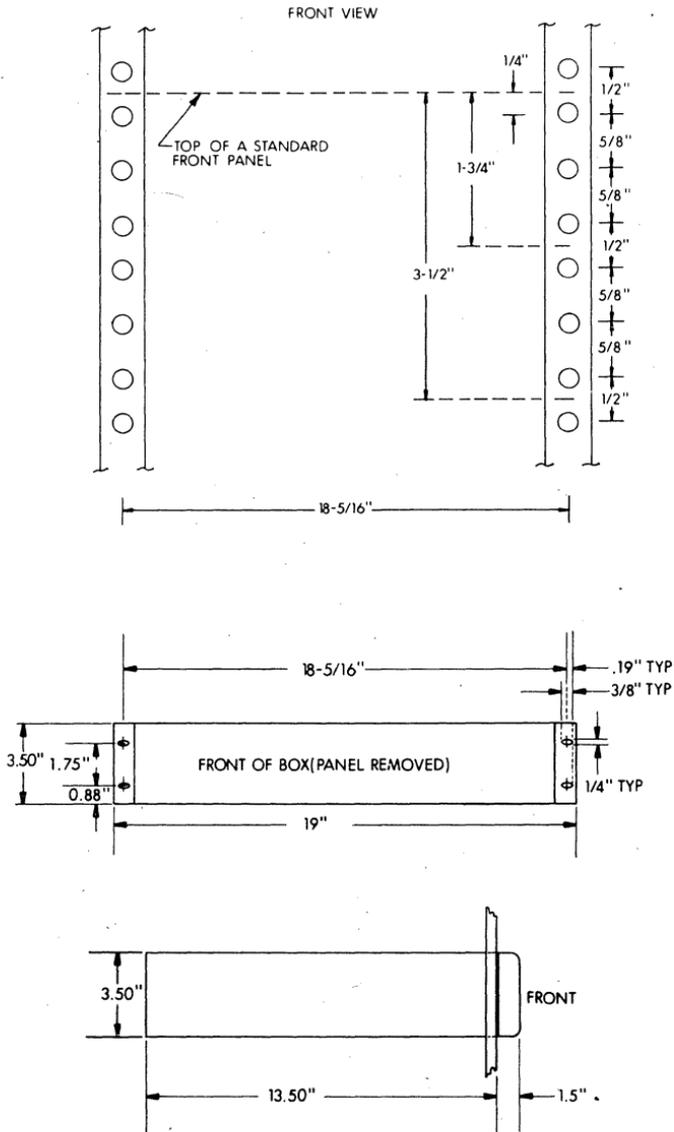


Figure 2-6 PDP-11/03 Cabinet Mounting

## 2.4 PDP-11V03 SYSTEMS

### 2.4.1 General

PDP-11V03 systems are available in four basic models, depending on selection of input power and terminal type. All PDP-11V03 systems include a PDP-11/03 system (Paragraph 2.3), RXV11 dual floppy disk system, an appropriate terminal, REV11-A (DMA refresh/ROM bootstrap/terminator module), DLV11 (terminal serial line unit interface) configured for the terminal supplied, MSV11 (4K memory, comprising 8K memory, total), system cabinet, and RT-11 system software and diagnostics. The four models include the major hardware system components as listed below:

| System Model | Terminal Model | Input Power |
|--------------|----------------|-------------|
| PDP-11V03-AA | VT52-AA        | 115 V/60 Hz |
| PDP-11V03-AD | VT52-AB        | 230 V/50 Hz |
| PDP-11V03-EA | LA36-DE        | 115 V/60 Hz |
| PDP-11V03-ED | LA36-DJ        | 230 V/50 Hz |

| PDP-11/03 Model | RXV11 Model | System Cabinet |
|-----------------|-------------|----------------|
| PDP-11/03-EA    | RXV11-BA    | H984-BA        |
| PDP-11/03-EB    | RXV11-BD    | H984-BB        |
| PDP-11/03-EA    | RXV11-BA    | H984-BA        |
| PDP-11/03-EB    | RXV11-BD    | H984-BB        |

All required signal and power interconnection cables are included in the system hardware.

### 2.4.2 Specifications

#### Size (H984 Cabinet):

Height: 26 in. (66 cm) (including casters)  
Length: 28 in. (71.1 cm)  
Width: 21½ in. (54.6 cm)

#### Weight (Basic System Modules and Components in cabinet):

205 lb with expander box fully loaded (not including terminal)

#### Power Requirements:

##### Input Voltage

100-127 Vrms, 50 ± 1 Hz or 60 ± 1 Hz  
200-254 Vrms, 50 ± 1 Hz or 60 ± 1 Hz

##### Input Current (Maximum configuration):

H984-BA (115 V input): 12 A  
H984-BB (230 V input): 6 A

#### Environmental Characteristics:

##### Temperature

Operating: 59° to 90° F (15° to 32° C) ambient; maximum temperature gradient = 20° F/hr (11.1° C/hr)  
Nonoperating: -30° to +140° F (-35° to +60° C)  
Media, non-operating: -30° to +125° F (-35° to 52° C)

### NOTE

Media temperature must be within operating temperature range before use.

#### Relative humidity

|                      |  |
|----------------------|--|
| Operating:           | 77° F (25° C) maximum wet bulb<br>36° F (2° C) minimum dew point<br>20% to 80% relative humidity |
| Nonoperating:        | 5% to 98% relative humidity (no condensation)  |
| Media, non-operating | 10% to 80% relative humidity   |

|                       |  |
|-----------------------|--|
| <b>Magnetic field</b> | Media exposed to a magnetic field strength of 50 Oe or greater may lose data |
|-----------------------|--|

## THE LSI-11 BUS

**3.1 CHOOSING AN I/O TRANSFER TYPE**

Before interfacing the processor with any peripheral device, the designer must determine the type of I/O transfer that would be best suited for the application: programmed I/O transfers, DMA, or interrupt-driven transfers.

Programmed I/O transfers are executed by single- or double-operand instructions. The instruction can be used to input or output a 16-bit data word or an 8-bit byte. By including the device's address as the effective source or destination address, the user selects the input or output operation. In many instances, the programmer inputs a byte from the device's control/status register (CSR) to determine that the device has input data ready or that it is ready to accept the processor's output data.

DMA transfers are the fastest method of transferring data between memory and a device. They can occur between processor bus cycles and do not alter processor status in any way. Addressing, controlling the size of the data block (number of word or byte transfers in the operation), and type of transfer are under the control of the requesting device. The processor does not modify data being moved in the DMA mode. Thus, blocks of data can be moved at memory speeds via the DMA transfer mode. The processor sets up these conditions before the DMA transfer is executed.

Interrupts allow the processor to continue a programmed operation (sometimes called a background program) without waiting for a device to become ready to transfer data. When the device does become ready, it interrupts the processor's background program execution and causes execution of a device interrupt service routine. After the device's service routine has been executed, the background program is restored and program execution resumes at the point where it was interrupted.

**3.2 DEVICE PRIORITY**

Each device has an I/O priority based on its distance from the processor. When two or more devices request interrupt service, the device electrically closer to the microcomputer will receive the interrupt grant (acknowledge). The microcomputer can be inhibited from issuing more grants by setting the processor's priority to 4 in the PS word. Bit 7 in the new PS word should be a 1. If further interrupts are to be serviced, the processor's priority should be 0, and bit 7 in the new PS word should be a 0. Consequently, interrupts can be nested to any level. Factors to consider when assigning device priorities are:

1. *Device Operating Speed*—Data from a fast device is available for only a short period; highest priorities are usually assigned to fast devices to prevent loss of data and to prevent the bus from being tied up by slower devices.

2. *Ease of Data Recovery*—If data from a device is lost, recovery may be automatic, may require manual intervention, or may be impossible to recover; highest priorities are assigned to devices whose data cannot be recovered.
3. *Service Requirements*—Some devices cannot function without help from the processor, while DMA devices can operate independently and require only minimal processor intervention; devices requiring continual help from the processor for servicing are assigned to lowest priorities to prevent typing up the processor.

Both address and data are multiplexed onto the 16 BDAL lines. In addition, individual control signals sequence programmed I/O operations, direct memory access (DMA), and processor interrupts. Any bus-compatible module can be inserted into any bus location and still receive interface signals; however, the module's priority, which is position-dependent along the bus, will change.

### 3.3 MODULE CONTACT FINGER IDENTIFICATION

DIGITAL plug-in (FLIP CHIP) modules, including LSI-11 modules, all use the same contact finger (pin) identification system. The LSI-11 I/O bus is based on the use of double-height modules. These modules plug into a two-slot bus connector, each containing 36 lines per slot (18 each on component and solder sides of the circuit board). Although the LSI-11 processor module and core memory module are quad-height modules that plug into four connector slots, only two slots (A and B) are used for interface purposes on the processor module. Etched circuit jumpers on the unused portion of the module maintain continuity of grant signals BIAKI L to BIAKO L and BDMGI L to BDMGO L. These daisy-chained signals are described later.

Slots, shown as ROW A and ROW B in Figure 3-1, include a numeric identifier for the side of the module. The component side is designated side "1" and the solder side is designated side "2." Letters ranging from A through V (excluding G, I, O, and Q) identify a particular pin on a side of a slot. Hence, a typical pin is designated as:

|                          |                |   |
|--------------------------|----------------|---|
|                          | BE2            |   |
| Slot (Row)<br>Identifier | Pin Identifier | Module Side Identifier<br>"solder side" |
| "Slot B"                 | "Pin E"        |   |

Note that the positioning notch between the two rows of pins mates with a protrusion on the connector block for correct module positioning.

Quad-height modules are similarly pin numbered. They are identified in Figure 3-2.

Individual connector pins, viewed from the underside (wiring side) of a backplane, are identified as shown in Figure 3-3. Only the pins for one bus location (two slots) are shown in detail. This pattern of pins is repeated eight times on the H9270 backplane, allowing the user to install one LSI-11 microcomputer module (four slots) and up to six additional two-slot modules.

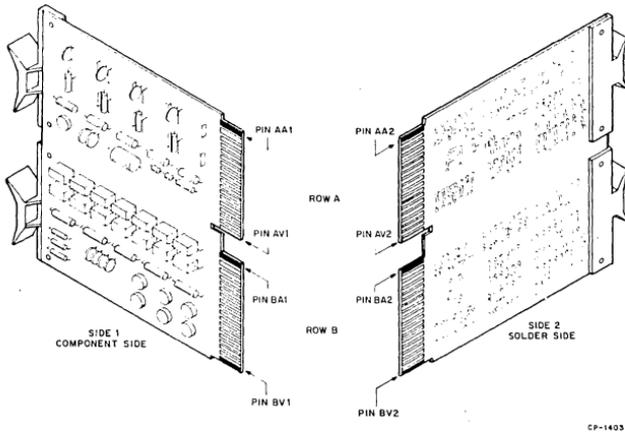


Figure 3-1 Module Contact Finger Identification

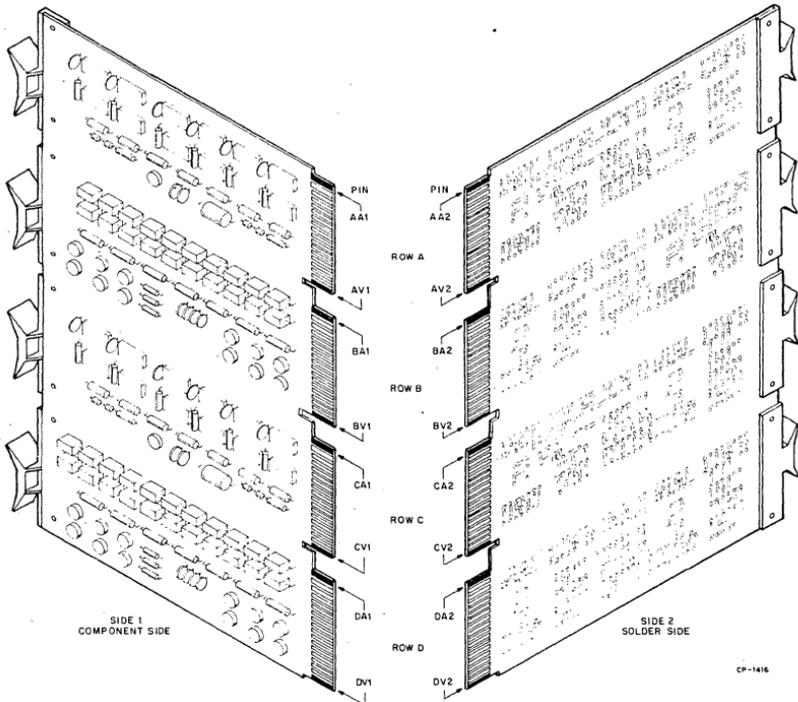
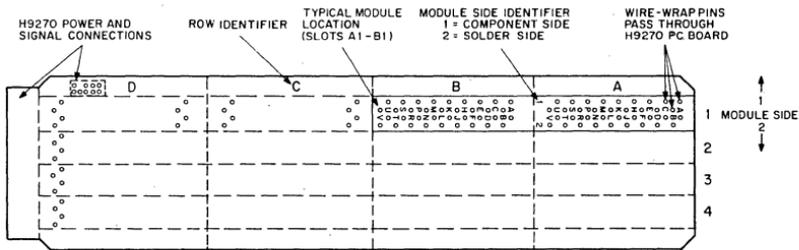


Figure 3-2 Quad Module Contact Finger Identification



CP-1773

Figure 3-3 LSI-11, PDP-11/03 Backplane Module Pin Identification

### 3.4 BUS-SIGNALS

H9270 backplane pin assignments are listed and described in Table 3-1. Only slots A and B are listed. However, they are identical to slots C and D, respectively. Applicable bus cycle timing and specifications are discussed in Paragraphs 3.12, 3.13, and 3.14.

Table 3-1 Backplane Pin Assignments

| Bus Pin | Mnemonic | Description   |
|---------|----------|---|
| AA1     | BSPARE1  | Bus Spare (Not Assigned, Reserved for DIGITAL use.)   |
| AB1     | BSPARE2  |   |
| AC1     | BAD16    | Extended address bits (not implemented)   |
| AD1     | BAD17    |   |
| AE1     | SSPARE1  | Special Spare (Not assigned, not bused. Available for user interconnections.)   |
| AF1     | SSPARE2  |   |
| AH1     | SSPARE3  |   |
| AJ1     | GND      | Ground—System signal ground and dc return.  |
| AK1     | MSPAREA  | Maintenance Spare—Normally connected together on the backplane at each option location (not bused connection).  |
| AL1     | MSPAREA  |   |
| AM1     | GND      | Ground—System signal ground and dc return.  |
| AN1     | BDMRL    | Direct Memory Access (DMA) Request—A device asserts this signal to request bus mastership. The processor arbitrates bus mastership between itself and all DMA devices on the bus. If the processor is not bus master (it has completed a bus cycle and BSYNC L is not being asserted by the processor), it grants bus mastership to the requesting device by asserting BDMGO L. The device responds by negating BDMR L and asserting BSACK L. |
| AP1     | BHALT L  | Processor Halt—When BHALT L is asserted, the processor responds by halting normal program execution. External interrupts are ignored but  |

**Table 3-1 Backplane Pin Assignments (Cont.)**

| <b>Bus Pin</b>   | <b>Mnemonic</b> | <b>Description</b>  |
|--|-----------------|---|
|  |                 | memory refresh interrupts (enabled if W4 on the processor module is removed) and DMA request/grant sequences are enabled. When in the halt state, the processor executes the ODT microcode and the console device operation is invoked. |
| AR1  | BREF L          | Memory Refresh—Asserted by a processor microcode-generated refresh interrupt sequence (when enabled) or by an external device. This signal forces all dynamic MOS memory units to be activated for each BSYNC L/BDIN L bus transaction. |
| <b>CAUTION</b>   |                 |   |
| The user must avoid using multiple DMA data transfers [Burst or "hog" mode] during a processor-generated refresh operation so that a complete refresh cycle can occur once every 1.6 ms. |                 |   |
| AS1  | PSPARE3         | Spare (Not assigned. Customer usage not recommended.)   |
| AT1  | GND             | Ground—System signal ground and dc return.  |
| AU1  | PSPARE1         | Spare (Not assigned. Customer usage not recommended.)   |
| AV1  | +5B             | +5 V Battery Power—Secondary +5 V power connection. Battery power can be used with certain devices.   |
| BA1  | BDCOK H         | DC Power OK—Power supply-generated signal that is asserted when there is sufficient dc voltage available to sustain reliable system operation.  |
| BB1  | BPOK H          | Power OK—Asserted by the power supply when primary power is normal. When negated during processor operation, a power fail trap sequence is initiated.   |
| BC1  | SSPARE4         | Special Spare (Not assigned, not bused. Available for user interconnections.)   |
| BD1  | SSPARE5         |   |
| BE1  | SSPARE6         |   |
| BF1  | SSPARE7         |   |
| BH1  | SSPARE8         |   |
| BJ1  | GND             | Ground—System signal ground and dc return.  |
| BK1  | MSPAREB         | Maintenance Spare—Normally connected together on the backplane at each option location (not a bused connection).  |
| BL1  | MSPAREB         |   |
| BM1  | GND             | Ground—System signal ground and dc return.  |

**Table 3-1 Backplane Pin Assignments (Cont.)**

| <b>Bus Pin</b> | <b>Mnemonic</b> | <b>Description</b>   |
|----------------|-----------------|--|
| BN1            | BSACK L         | This signal is asserted by a DMA device in response to the processor's BDMGO L signal, indicating that the DMA device is bus master.   |
| BP1            | BSPARE6         | Bus Spare (Not assigned. Reserved for DIGITAL use.)  |
| BR1            | BEVNT L         | External Event Interrupt Request—When asserted, the processor responds (if PS bit 7 is 0) by entering a service routine via vector address 100 <sub>8</sub> . A typical use of this signal is a line time clock interrupt. |
| BS1            | PSPARE4         | Spare (Not assigned. Customer usage not recommended.)  |
| BT1            | GND             | Ground—System signal ground and dc return.   |
| BU1            | PSPARE2         | Spare (Not assigned. Customer usage not recommended.)  |
| BV1            | +5              | +5 V Power—Normal +5 V dc system power.  |
| AA2            | +5              | +5 V Power—Normal +5 V dc system power.  |
| AB2            | -12             | -12 V Power— -12 V dc (optional) power for devices requiring this voltage.   |

**NOTE**

LSI-11 modules which require negative voltages contain an inverter circuit (on each module) which generates the required voltage(s) hence, -12 V power is not required with DIGITAL-supplied options.

|     |         |  |
|-----|---------|--|
| AC2 | GND     | Ground—System signal ground and dc return.   |
| AD2 | +12     | +12 V Power—+12 V dc system power.   |
| AE2 | BDOUT L | Data Output—BDOUT, when asserted, implies that valid data is available on BDOL0—15 L and that an output transfer, with respect to the bus master device, is taking place. BDOUT L is deskewed with respect to data on the bus. The slave device responding to the BDOUT L signal must assert BRPLY L to complete the transfer. |
| AF2 | BRPLY L | Reply—BRPLY L is asserted in response to BDIN L or BDOUT L and during IAK transaction. It is generated by a slave device to indicate that it has placed its data on the BDAL bus or that it has accepted output data from the bus.   |
| AH2 | BDIN L  | Data Input—BDIN L is used for two types of bus operation:<br><ol style="list-style-type: none"> <li>1. When asserted during BSYNC L time, BDIN L implies an input transfer with respect to the current bus master, and requires a re-</li> </ol>   |

**Table 3-1 Backplane Pin Assignments (Cont.)**

| <b>Bus Pin</b> | <b>Mnemonic</b>    | <b>Description</b>   |
|----------------|--------------------|--|
|                |                    | <p>sponse (BRPLY L). BDIN L is asserted when the master device is ready to accept data from a slave device.</p> <p>2. When asserted without BSYNC L, it indicates that an interrupt operation is occurring.</p> <p>The master device must deskew input data from BRPLY L.</p>  |
| AJ2            | BSYNC L            | Synchronize—BSYNC L is asserted by the bus master device to indicate that it has placed an address on BDALO—15 L. The transfer is in process until BSYNC L is negated.   |
| AK2            | BWTBT L            | Write/Byte—BWTBT L is used in two ways to control a bus cycle: <ol style="list-style-type: none"> <li>1. It is asserted during the leading edge of BSYNC L to indicate that an output sequence is to follow (DATO or DATOB), rather than an input sequence.</li> <li>2. It is asserted during BDOUT L, in a DATOB bus cycle, for byte addressing.</li> </ol>   |
| AL2            | BIRQ L             | Interrupt Request—A device asserts this signal when its Interrupt Enable and Interrupt Request flip-flops are set. If the processor's PS word bit 7 is 0, the processor responds by acknowledging the request by asserting BDIN L and BIAKO L.   |
| AM2<br>AN2     | BIAKI L<br>BIAKO L | Interrupt Acknowledge Input and Interrupt Acknowledge Output—This is an interrupt acknowledge signal which is generated by the processor in response to an interrupt request (BIRQ L). The processor asserts BIAKO L, which is routed to the BIAKI L pin of the first device on the bus. If it is requesting an interrupt, it will inhibit passing BIAKO L. If it is not asserting BIRQ L, the device will pass BIAKI L to the next (lower priority) device via its BIAKO L pin and the lower priority device's BIAKI L pin. |
| AP2            | BBS7 L             | Bank 7 Select—The bus master asserts BBS7 L when an address in the upper 4K bank (address in the 28-32K range) is placed on the bus. BSYNC L is then asserted and BBS7 L remains active for the duration of the addressing portion of the bus cycle.   |
| AR2<br>AS2     | BDMGI L<br>BDMGO L | DMA Grant-Input and DMA Grant Output—This is the processor-generated daisy-chained signal which grants bus mastership to the highest priority DMA device along the bus. The processor generates BDMGO L, which is routed to the  |

**Table 3-1 Backplane Pin Assignments (Cont.)**

| <b>Bus Pin</b> | <b>Mnemonic</b>    | <b>Description</b>  |
|----------------|--------------------|---|
|                |                    | BDMGI L pin of the first device on the bus. If it is requesting the bus, it will inhibit passing BDMGO L. If it is not requesting the bus, it will pass the BDMGI L signal to the next (lower priority) device via its BDMGO L pin. The device asserting BDMR L is the device requesting the bus, and it responds to the BDMGI L signal by negating BDMR, asserting BSACK L, assuming bus mastership, and executing the required bus cycle. |
|                |                    | <b>CAUTION</b>  |
|                |                    | DMA device transfers must be single transfers and must not interfere with the memory refresh cycle.   |
| AT2            | BINIT L            | Initialize—BINIT is asserted by the processor to initialize or clear all devices connected to the I/O bus. The signal is generated in response to a power-up condition (the negated condition of BDCOK H).  |
| AU2<br>AV2     | BDALO L<br>BDALI L | Data/Address Lines—These two lines are part of the 16-line data/address bus over which address and data information are communicated. Address information is first placed on the bus by the bus master device. The same device then either receives input data from, or outputs data to the addressed slave device or memory over the same bus lines.   |
| BA2            | +5                 | +5 V Power—Normal +5 V dc system power.   |
| BB2            | -12                | -12 V Power— -12 V dc (optional) power for devices requiring this voltage.  |
| BC2            | GND                | Ground—System signal ground and dc return.  |
| BD2            | +12                | +12 V Power—+12 V system power.   |
| BE2            | BDAL2 L            |   |
| BF2            | BDAL3 L            |   |
| BH2            | BDAL4 L            |   |
| BJ2            | BDAL5 L            |   |
| BK2            | BDAL6 L            |   |
| BL2            | BDAL7 L            |   |
| BM2            | BDAL8 L            |   |
| BN2            | BDAL9 L            |   |
| BP2            | BDAL10 L           |   |
| BR2            | BDAL11 L           |   |
| BS2            | BDAL12 L           |   |
| BT2            | BDAL13 L           |   |
| BU2            | BDAL14 L           |   |
| BV2            | BDAL15 L           |   |
|                |                    | Data/Address Lines—These 14 lines are part of the 16-line data/address bus previously described.  |

## 3.5 BUS CYCLES

### 3.5.1 General

Every processor instruction requires one or more I/O operations. The first operation required is a data input transfer (DATI), which fetches an instruction from the location addressed by the program counter (PC or R7). This operation is called a DATI bus cycle. If no additional operands are referenced in memory or in an I/O device, no additional bus cycles are required for instruction execution. However, if memory or a device is referenced, additional DATI, data input/output (DATIO or DATIOB), or data output transfer (DATO or DATOB) bus cycles are required. Between these bus cycles, the processor can service DMA requests. In addition, the processor can service interrupt requests only prior to an instruction fetch (DATI bus cycle) if the processor's priority is zero. (PS word bit 7 is 0.)

The following paragraphs describe the types of bus cycles. Note that the sequences for I/O operations between processor and memory or between processor and I/O device are identical. DATO (or DATOB) cycles are equivalent to write operations, and DATI cycles are equivalent to read operations. In addition, DATIO cycles include an input transfer followed by an output transfer. The DATIO cycle provides an efficient means of executing an equivalent read-modify-write operation by making it unnecessary to assert an address a second time.

### 3.5.2 Input Operations

The sequence for a DATI operation is shown in Figure 3-4. DATI cycles are asynchronous and require a response from the addressed device or memory. The addressed memory or device responds to its input request (BDIN L) by asserting BRPLY L. If BRPLY is not asserted within 10  $\mu$ s (max) after BDIN L is asserted, the processor terminates the cycle and traps through location 4.

Note that BWTBT L is not asserted during the address time, indicating that an input data transfer is to be executed.

A DATIO cycle is equivalent to a read-modify-write operation. An addressing operation and an input word transfer are first executed in a manner similar to the DATI cycle; however, BSYNC L remains in the active state after completing the input data transfer. This causes the addressed device or memory to remain selected, and an output data transfer follows without any further addressing. After completing the output transfer, the device terminates BSYNC L, completing the DATIO cycle. The actual sequence required for a DATIO cycle is shown in Figure 3-5. Note that the output data transfer portion of the bus cycle can be a byte transfer; hence, this cycle is shown as DATIOB.

### 3.5.3 Output Operations

The sequence required for a DATO or the equivalent output byte (DATOB) bus cycle is shown in Figure 3-6. Like the input operations, failure to receive BRPLY L within 10  $\mu$ s after asserting BDOUT L is an error, and results in a processor time-out trap through location 4.

Note that BWTBT L is asserted during addressing portion of the cycle to indicate that an output data transfer is to follow. If a DATOB is to be executed, BWTBT L remains active for the duration of the bus cycle;

**BUS MASTER  
(PROCESSOR OR DEVICE)**

**SLAVE  
(MEMORY OR DEVICE)**

**ADDRESS DEVICE/MEMORY**

- Assert BDALO-15 L with address and
- Assert BBS7 if the address is in the 28 - 32K range
- Assert BSYNC L

**DECODE ADDRESS**

- Store "device selected" operation

**REQUEST DATA**

- Remove the address from BDALO-15 L and negate BBS7 L
- Assert BDIN L

**INPUT DATA**

- Place data on BDALO-15 L
- Assert BRPLY L

**TERMINATE INPUT TRANSFER**

- Accept data and respond by negating BDIN L

**TERMINATE BUS CYCLE**

- Negate BSYNC L

**OPERATION COMPLETED**

- Terminate BRPLY L

11-3138

Figure 3-4 DATI Bus Cycle

however, if a DATO (word transfer) is to be executed, BWTBT L is negated during the remainder of the cycle.

#### NOTE

Normally, all devices, when addressed, will respond to both BDIN L and BDOUT L by asserting BRPLY L to acknowledge the bus cycle. However, there are two special cases with which to be concerned. First, by PDP-11 convention, ROM locations do not respond to BDOUT L signals; in this case, processor instructions that generate BDOUT L will result in a non-existent memory trap. Second, processors may generate unnecessary BDOUT L signals, as when a processor fetches an operand via a DATIO cycle. Because of the first case, no destination operands can be located in ROM, except for the following (non-modifying) instructions: TST(B), CMP(B), BIT(B), JMP, and JSR. Because of the second case, other (processor dependent) instructions may not access source operands from ROM, as in the following cases: MTPS (all LSI-11 processors), and EIS instructions MUL, DIV, ASH, ASHC (processors in which the KEV11 EIS/FIS option is installed).

### 3.6 DMA OPERATIONS

DMA I/O operations involve a peripheral device and system memory. A device can transfer data to or from the 4K memory on the processor module or any read/write memory module along the bus. The actual sequence of operations for executing the data transfer once a device has been granted DMA bus control is as previously described for input and output I/O bus cycles, except the DMA device, not the processor, is bus master (controls the operation). Memory addressing, timing, and control signal generation/response are provided by logic contained on the device's DMA interface module; the processor is not involved with address and data transfers during a DMA operation.

The required DMA sequence is shown in Figure 3-7. A device requests the I/O bus by asserting BDMR L. After completing the present bus cycle, the processor responds by asserting BDMGO L, allowing the device to become bus master. It also inhibits further processor generation of BSYNC L, preventing processor-initiation of a new bus cycle. The device responds by asserting BSACK L and negating BDMR L, causing the processor to terminate BDMGO L; the device is now bus master and it can execute the required data transfer in the same manner described for a DATI, DATIO, DATIOB, DATO, or DATOB bus cycle. When the data transfer is completed, the device returns bus master control to the processor by terminating the BSACK L and BSYNC L signals.

**BUS MASTER  
(PROCESSOR OR DEVICE)**

**SLAVE  
(MEMORY OR DEVICE)**

**ADDRESS DEVICE/MEMORY**

- Assert BDAL0-15 L with address
- Assert BBS7 L and if the address is in the 28 - 32K range
- Assert BSYNC L

**DECODE ADDRESS**

- Store "device selected" operation

**REQUEST DATA**

- Remove the address from BDAL0 - 15 L and negate BBS7 L
- Assert BDIN L

**INPUT DATA**

- Place data on BDAL0-15 L
- Assert BRPLY L

**TERMINATE INPUT TRANSFER**

- Accept data and respond by terminating BDIN L

**COMPLETE INPUT TRANSFER**

- Remove data
- Terminate BRPLY L

**OUTPUT DATA**

- Place output data on BDAL0-15 L
- (Assert BWTBT L if an output byte transfer)
- Assert BDOUT L

**TAKE DATA**

- Receive data from BDAL lines
- Assert BRPLY L

**TERMINATE OUTPUT TRANSFER**

- Terminate BDOUT L, and remove data from BDAL lines

**OPERATION COMPLETED**

- Terminate BRPLY L

**TERMINATE BUS CYCLE**

- Negate BSYNC L (and BWTBT L if in a DATIOB bus cycle)

11-3139

Figure 3-5 DATIO or DATIOB Bus Cycle

**BUS MASTER  
(PROCESSOR OR DEVICE)**

**SLAVE  
(MEMORY OR DEVICE)**

**ADDRESS DEVICE/MEMORY**

- Assert BDAL0-15 L with address and
- Assert BBS7 L (if address is in the 28 - 32K range)
- Assert BWTBT L (write cycle)
- Assert BSYNC L

**DECODE ADDRESS**

- Store "device selected" operation

**OUTPUT DATA**

- Remove the address from BDAL0 - 15 L and negate BBS7 L and BWTBT L (BWTBT L remains active if DATOB cycle)
- Place data on BDAL0-15 L
- Assert BDOUT L

**TAKE DATA**

- Receive data from BDAL lines
- Assert BRPLY L

**TERMINATE OUTPUT TRANSFER**

- Remove data from BDAL0-15L and negate BDOUT L

**OPERATION COMPLETED**

- Terminate BRPLY L

**TERMINATE BUS CYCLE**

- Negate BSYNC L (and BWTBT L if a DATOB bus cycle)

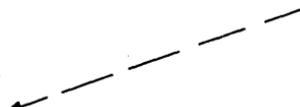
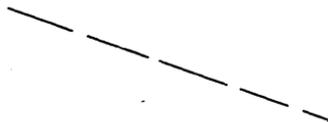
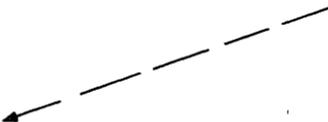
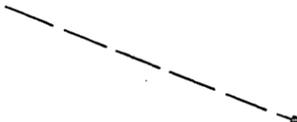
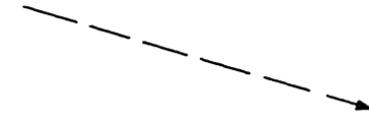


Figure 3-6 DATO or DATOB Bus Cycle

**LSI-11 PROCESSOR  
(MEMORY IS SLAVE)**

**DEVICE**

**REQUEST BUS**

- Assert **BDMR L**

**GRANT BUS CONTROL**

- Near the end of the current bus cycle (**BRPLY L** is negated), assert **BDMGO L** and inhibit new processor generated **BSYNC L** for the duration of the DMA operation.

**ACKNOWLEDGE BUS MASTERSHIP**

- Wait for negation of **BSYNC L** and **BRPLY L**
- Assert **BSACK L**
- Negate **BDMR L**

**TERMINATE GRANT SEQUENCE**

- Negate **BDMGO L** and wait for DMA operation to be completed

**EXECUTE A DMA DATA TRANSFER (DEVICE IS BUS MASTER)**

- Address memory and transfer data as described for **DATI**, **DATIO**, **DATIOB**, **DATO**, **DATOB** bus cycles
- Release the bus by terminating **BSACK L** (no sooner than negation of last **BRPLY L**) and **BSYNC L**.

**RESUME PROCESSOR OPERATION**

- Enable processor-generated **BSYNC L** (Processor is Bus master) Or issue another grant if **BDMR L** is asserted.

11-3141

Figure 3-7 DMA Request/Grant Sequence

### 3.7 INTERRUPTS

Interrupts are requests, made by peripheral devices, which cause the processor to temporarily suspend its present (background) program execution to service the requesting device. Each device which is capable of requesting an interrupt must have a user-supplied service routine that is automatically entered when the processor acknowledges the interrupt request. After completing the service routine execution, program control is returned to the interrupted program. This type of operation is especially useful for the slower peripheral devices.

A device can interrupt the processor only when interrupts are enabled and services interrupts only when its PS bit 7 is cleared. Device priority is highest for devices electrically closest to the processor along the bus. Any device that can interrupt the processor can also interrupt the service routine execution of a lower priority device if the processor's priority is 0 (PS bit 7 is cleared) during that execution; hence, interrupt nesting to any level is possible with this interrupt structure. Each device normally contains a control status register (CSR), which includes an interrupt enable bit. A program must set this bit before an interrupt can be generated by the device.

An interrupt vector associated with each device is hard-wired into the device's interface/control logic. This vector is an address pointer that is transmitted to the processor during the interrupt acknowledge sequence, allowing automatic entry into the service routine without device polling.

When the BEVNT L signal line is asserted, the processor automatically services the request via location 100<sub>8</sub>; it does not input a vector address as done for other external interrupt devices. This function is normally used for a line time clock input based on the frequency of the local ac power (50 or 60 Hz).

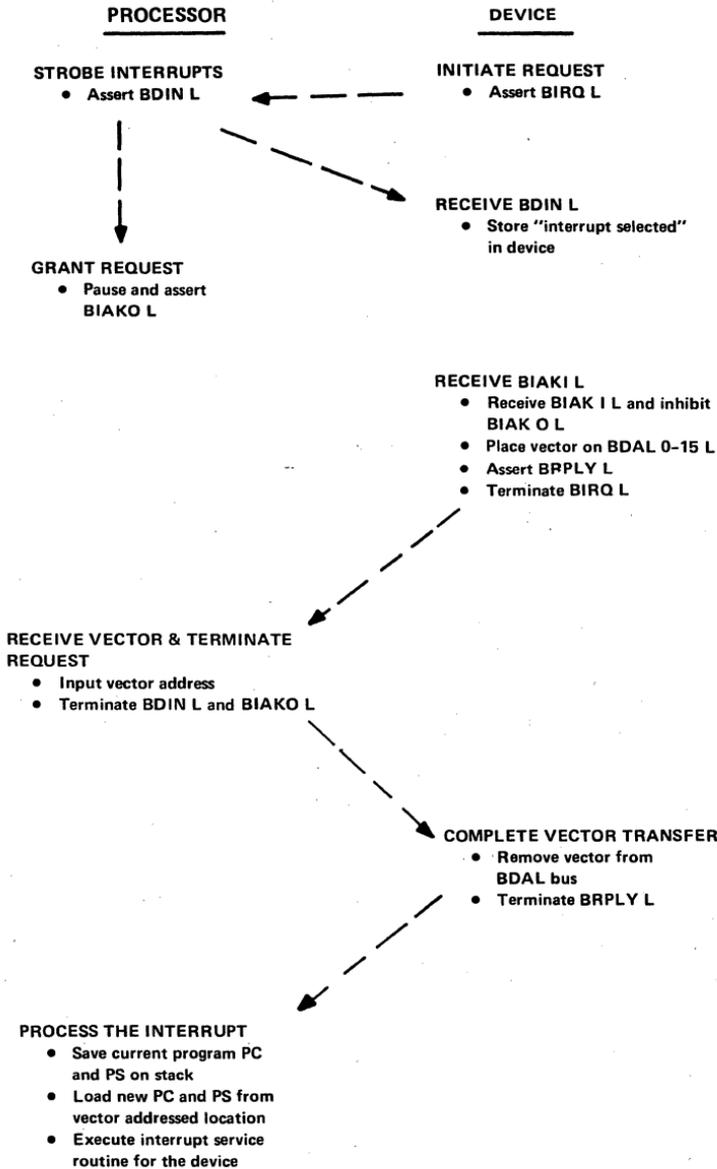
The interface control and data signal sequence required for interrupts is shown in Figure 3-8. A device requests interrupt service by asserting BIRQ L. The processor can acknowledge interrupt requests only between instruction executions by generating an active (low) BDIN L signal, enabling the device's first response. The processor then asserts the BIAKO L signal. The first device on the bus receives this daisy-chained signal at its BIAKI L input. If it is not requesting service, it passes the signal via its BIAKO L output to the next device, and so on, until the requesting device receives the signal. This device will not pass the BIAKO L signal and it responds by asserting BRPLY L (low) and placing its interrupt vector on data/address bus lines BDALO—15 L. Automatic entry to the service routine is then executed by the processor as previously described.

#### NOTE

If a device fails to assert BRPLY L in response to BDIN L within 10  $\mu$ s, the processor enters the Halt state.

### 3.8 BUS INITIALIZATION

Devices along the I/O bus are initialized whenever the system dc voltages are cycled on or off, or when a RESET instruction is executed.



11-3142

Figure 3-8 Interrupt Request/Acknowledge Sequence

Initialization during the power-on/power-off sequence is described in Paragraph 3.9. When the RESET instruction is executed, the processor responds by asserting BINIT L for approximately 10  $\mu$ s. Devices along the bus respond to the BINIT L signal, as appropriate, by clearing registers and presetting or clearing flip-flops.

### **3.9 POWER-UP POWER-DOWN SEQUENCE**

Power status signals BPOK H and BDCOK H must be asserted or negated in a particular sequence as dc operating power is applied or removed. Initially, BDCOK H and BPOK H are passive (low). As dc voltages rise to operating levels, BINIT L is asserted by the processor module. Approximately 3 ms (min) after +5 V and +12 V power are normal, an external signal source, or the H780 power supply in PDP-11/03 systems, produces an active BDCOK H signal; the processor responds by negating BINIT, and waits for BPOK H. The BPOK H signal, produced by an external signal source or the H780 power supply, goes true (high) 70 ms (min) after BDCOK H goes high. The processor responds by executing the user-selected power-up routine (Paragraph 5.2); if BHALT L is asserted, the console microcode is executed.

During a power-down sequence, the external signal source first negates BPOK H, causing the processor to execute the power-fail trap (PC at 024, PS at 026). Approximately 3 ms (max) later, the processor initializes the bus by asserting BINIT L in response to the external signal negation of BDCOK H.

### **3.10 HALT MODE**

The BHALT L bus signal can be asserted low to place the processor in the Halt mode. When in the Halt mode, the RUN indicator (PDP-11/03 only) is extinguished, interrupts external to the processor module are ignored, and the processor executes the console ODT microcode. Although the user could assert this line by a separate switch or a custom module, it is normally asserted by the HALT/ENABLE switch (PDP-11/03 only) or the user-designated device's SLU interface module when the Framing Error Halt is enabled. Note that when in the Halt mode, the processor arbitrates DMA requests, and refresh operations. Thus, in addition to bus transactions between the processor and the console device, bus transactions can occur for DMA and refresh.

### **3.11 MEMORY REFRESH**

Memory refresh operations are required when any dynamic MOS memory devices are used in a system. These memory devices are included on KD11-F and MSV11-B modules. Memory refresh is normally controlled by the processor microcode, which is automatically executed once every 1.6 ms. However, refresh could be controlled by the REV11-A or REV11-C options, as described in Chapter 4 and 5, or a user-supplied DMA device on the bus. (For example, when used in an intelligent terminal application, the refresh logic could be included on the user's DMA interface module.)

A complete refresh operation requires 64 BSYNC/BDIN transactions which must be completed within 2 ms. The processor (or other device controlling the refresh operation) first asserts BREF L for each BSYNC/BDIN transaction during the addressing portion of each refresh operation. BREF L causes all dynamic MOS memory devices to be simul-

taneously enabled and addressed, overriding local bank selection circuits. Refresh is then accomplished by executing 64 BSYNC/BDIN transactions, in a manner similar to the DAT1 bus cycle, incrementing the "row" address (bits 1—6) once for each transaction. Address bit 0 is not significant in the refresh operation. When refresh is controlled by processor microcode, the operation takes approximately 130  $\mu$ s.

Note that only one dynamic MOS memory device is required to assert BRPLY L during the refresh BSYNC/BDIN transactions. This should be performed by the slowest device on the bus. MSV11-B modules each contain a jumper which the user can insert to prevent the module from asserting BRPLY L during refresh operations. The slowest memory device will normally be the MSV11-B module located the greatest electrical distance from the processor module along the bus.

### 3.12 BUS SPECIFICATIONS

#### Electrical

##### Input Logic Levels

|                   |             |
|-------------------|-------------|
| TTL Logical Low:  | 0.8 Vdc max |
| TTL Logical High: | 2.0 Vdc min |

##### Output Logic Levels

|                   |             |
|-------------------|-------------|
| TTL Logical Low:  | 0.4 Vdc max |
| TTL Logical High: | 2.4 Vdc min |

##### Bus Receivers

|               |                                      |
|---------------|--------------------------------------|
| Logical Low:  | 1.3 Vdc max, $-10\mu$ A max at 0 V   |
| Logical High: | 1.7 Vdc min, 80 $\mu$ A max at 2.5 V |

##### Bus Drivers

|               |                         |
|---------------|-------------------------|
| Logical Low:  | 0.8 Vdc max at 70 mA    |
| Logical High: | 25 $\mu$ A max at 3.5 V |

#### NOTE

All bus lines are open-collector, resistor-terminated to 3.4 V nominal.

#### Bus Drivers and Receivers

##### Recommended Bus Drivers

Type 957, P/N DEC 888-1, quad 2-input NAND gates. (Refer to specifications in Table 8-1.)

##### Recommended Bus Receivers

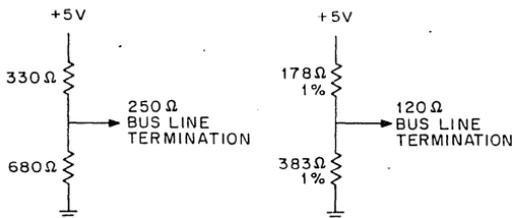
Type 956, P/N DEC 8640, quad 2-input NOR gates. (Refer to specifications in Table 8-1.)

##### Recommended Bus Transceivers

Type DEC 8641, quad unified bus transceiver.

### 3.13 BUS CONFIGURATIONS

In the following descriptions, a unit load is equal to one bus receiver and two bus drivers and less than 10 pF of circuit board etch. Bus terminations are shown in Figure 3-9.



CP-1828

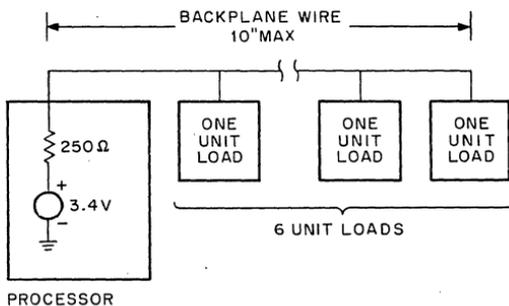
Figure 3-9 Bus Line Terminations

**Minimum Configuration (Figure 3-10)**

1. The processor terminates the bus lines to  $Z_t = 250 \Omega$ .
2. Ten-inch maximum backplane wire (each bus line for a 4 by 4 backplane), 6 unit loads or less.

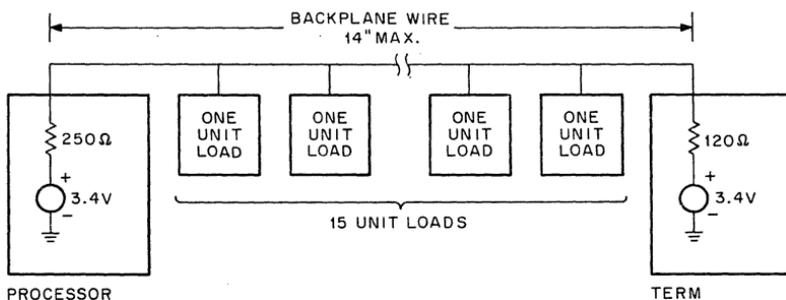
**Intermediate Configuration (Figure 3-11)**

1. The processor terminates the bus lines to  $Z_t = 250 \Omega$ .



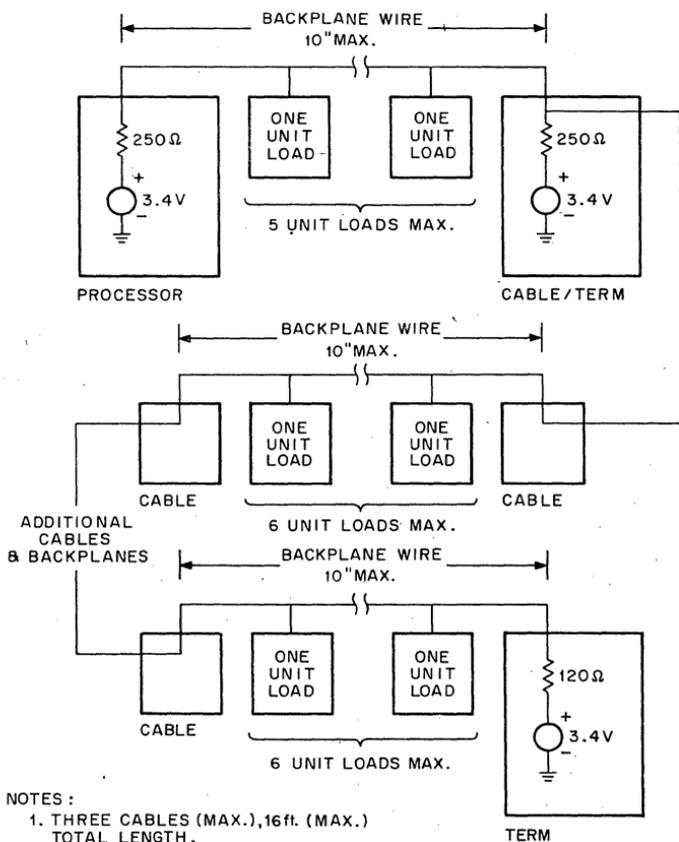
CP-1829

Figure 3-10 Minimum Configurations



CP-1830

Figure 3-11 Intermediate Configuration



CP-1831

Figure 3-12 Maximum Configuration

- Fourteen-inch maximum backplane wire (each bus line for a 9 by 4 backplane), 15 unit loads or less.
- An additional 120 Ω termination is required.

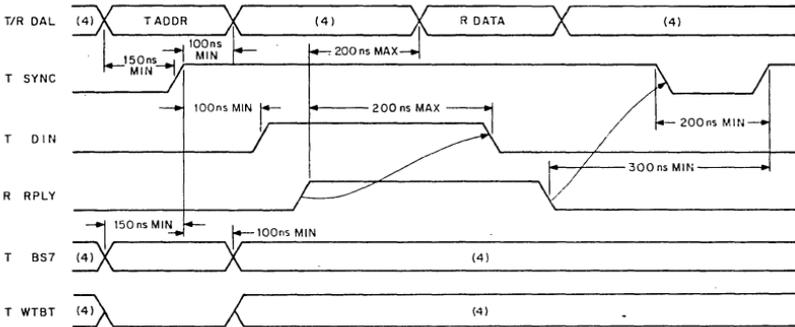
**Maximum Configuration (Figure 3-12)**

- The processor terminates the bus lines to  $Z_t = 250 \Omega$ .
- Ten-inch maximum backplane wire on each backplane (each bus line for 4 by 4 backplanes); 6 unit loads maximum each backplane, 15 unit loads total (maximum); daisy-chained on 2 feet (minimum) 120 Ω cable, three cables maximum, total cable length not exceeding 16 feet.

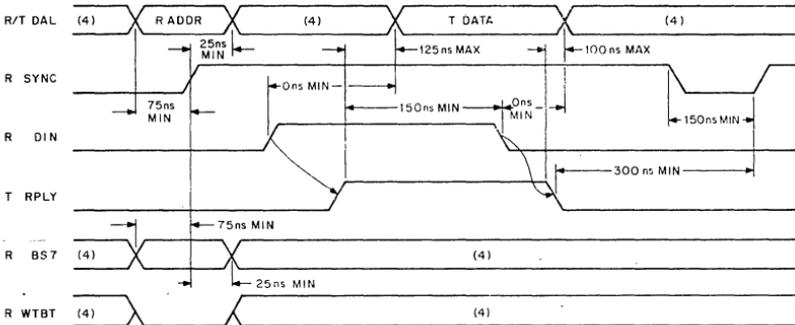
3. Two additional terminations (one 250  $\Omega$  and one 120  $\Omega$ ) are required.

### 3.14 BUS SIGNAL TIMING

Bus signal timing requirements at master and slave devices are shown in Figures 3-13 through 3-18.



TIMING AT MASTER DEVICE



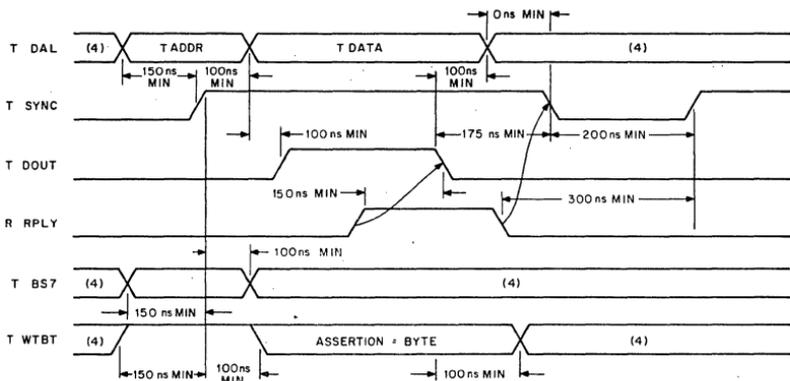
TIMING AT SLAVE DEVICE

**NOTES:**

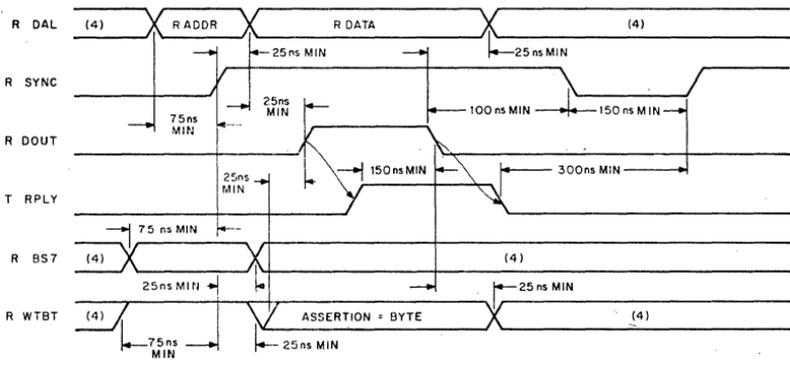
1. Timing shown at Master and Slave Device.  
Bus Driver inputs and Bus Receiver Outputs.
2. Signal name prefixes are defined below:  
T = Bus Driver Input  
R = Bus Receiver Output
3. Bus Driver Output and Bus Receiver Input  
signal names include a "B" prefix.
4. Don't care condition.

CP-1774

Figure 3-13 DAT1 Bus Cycle Timing



TIMING AT MASTER DEVICE



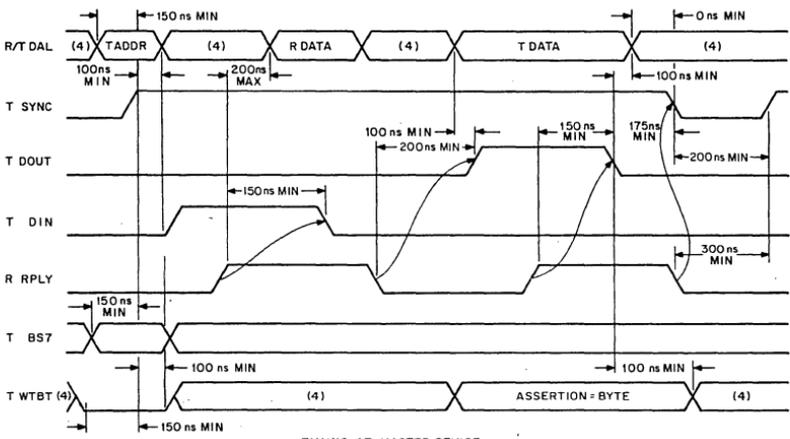
TIMING AT SLAVE DEVICE

NOTES:

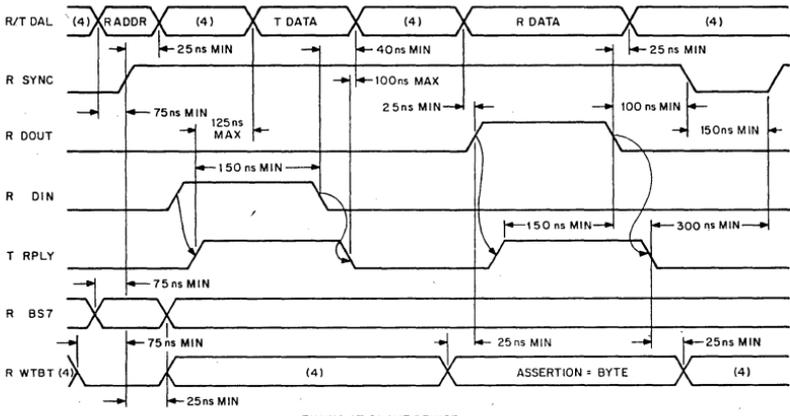
1. Timing shown at Master and Slave Device Bus Driver Inputs and Bus Receiver Outputs.
2. Signal name prefixes are defined below:  
T = Bus Driver Input  
R = Bus Receiver Output
3. Bus Driver Output and Bus Receiver Input signal names include a "B" prefix.
4. Don't care condition.

CP-1775

Figure 3-14 DATO or DATOB Bus Cycle Timing



TIMING AT MASTER DEVICE



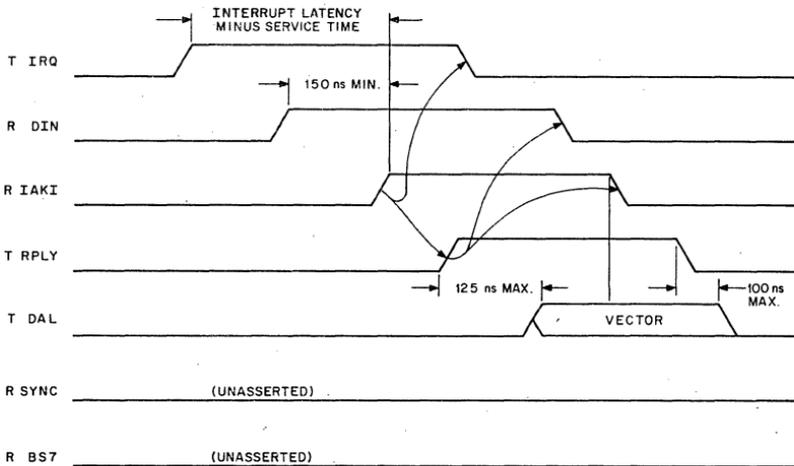
TIMING AT SLAVE DEVICE

NOTES:

1. Timing shown at Requesting Device  
Bus Driver Inputs and Bus Receiver Outputs.
2. Signal name prefixes are defined below:  
T = Bus Driver Input  
R = Bus Receiver Output
3. Bus Driver Output and Bus Receiver Input  
signal names include a "B" prefix.
4. Don't care condition.

CP-1776

Figure 3-15 DATIO Bus Cycle Timing

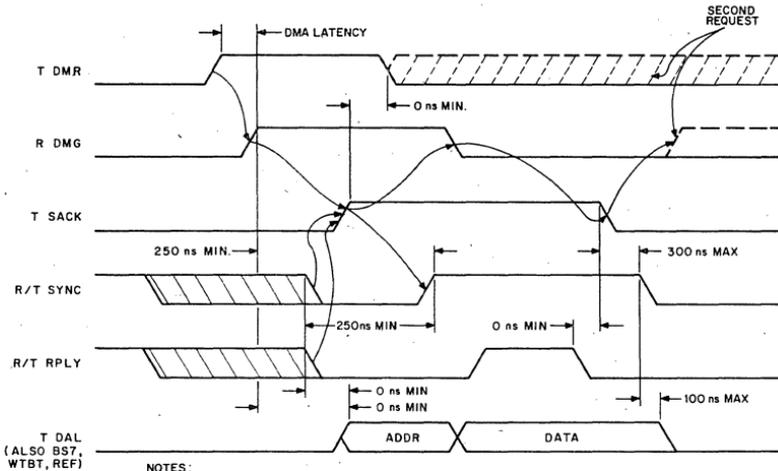


NOTES:

1. Timing shown at Requesting Device Bus Driver Inputs and Bus Receiver Outputs.
2. Signal Name Prefixes are defined below:  
 T = Bus Driver Input  
 R = Bus Receiver Output
3. Bus Driver Output and Bus Receiver Input signal names include a "B" prefix.

CP-1777

Figure 3-16 Interrupt Transaction Timing



NOTES:

1. Timing shown at requesting device bus driver inputs and bus receiver outputs.
2. Signal name prefixes are defined below:  
 T = Bus Driver Input  
 R = Bus Receiver Output
3. Bus Driver Output and Bus Receiver Input signal names include a "B" prefix.

CP-1778

Figure 3-17 DMA Request/Grant Timing

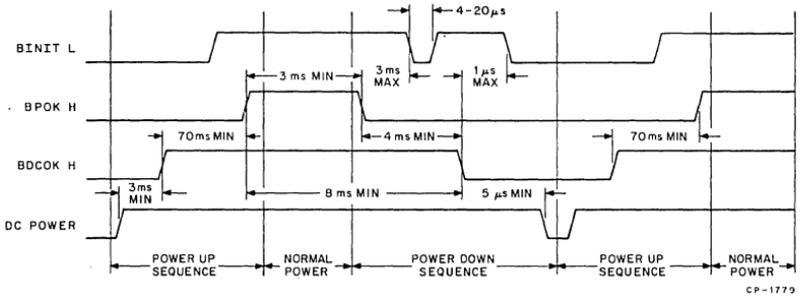


Figure 3-18 Power-Up/Power-Down Timing



## LSI-11 MODULE DESCRIPTIONS

## 4.1 GENERAL

This chapter contains detailed descriptions of each LSI-11 module. The level of coverage is sufficient to enable users to interface their systems with the PDP-11/03 or LSI-11 using standard LSI-11 modules or user-designed interfaces. Refer to Chapter 3 for detailed bus timing information.

LSI-11 modules covered in this chapter are listed in Table 4-1. Note that a separate description for the KD11-J microcomputer is not provided; it comprises the same M7264 module as the KD11-F microcomputer, except that a resident semiconductor memory is not supplied. Instead, the MMV11 core memory module is supplied with the KD11-J option.

Table 4-1 LSI-11 Modules

| Option            | Module Number        | Module Option                                       | Description Para. No. |
|-------------------|----------------------|---|-----------------------|
| KD11-F            | M7264                | LSI-11 microcomputer and 4K semiconductor memory    | 4.2                   |
| KD11-J            | M7264-YA, H223, G653 | LSI-11 microcomputer and 4K core memory             | 4.2                   |
| KEV-11            | —                    | EIS/FIS processor chip                              | 4.2                   |
| MMV11-A           | H223, G653           | K4 by 16-bit core memory                            | 4.3                   |
| MRV11-AA          | M7942                | 4K by 16-bit PROM                                   | 4.4                   |
| MSV11-B           | M7944                | 4K by 16-bit dynamic read-write memory              | 4.5                   |
| DLV11             | M7940                | Serial line unit                                    | 4.6                   |
| DRV11             | M7941                | Parallel line unit                                  | 4.7                   |
| DRV11-B           | M7950                | DMA interface                                       | 4.8                   |
| DRV11-P           | M7948                | LSI-11 bus foundation                               | 4.9                   |
| REV11-A           | M9400-YA             | 250 $\Omega$ terminator, DMA refresh, bootstrap ROM | 4.10                  |
| REV11-C           | M9400-YC             | DMA refresh, bootstrap ROM                          | 4.10                  |
| REV11-H           | M9400-YH             | DMA refresh, bootstrap ROM                          | 4.10                  |
| TEV11             | M9400-YB             | 120 $\Omega$ terminator                             | 4.10                  |
| BCV1A             | M9400-YD<br>M9401    | Bus expansion modules and cables                    | 4.10                  |
| BCV1B             | M9400-YE<br>M9401    | Bus expansion modules and cables                    | 4.10                  |
| H780-A and H780-B |                      | Power Supply  | 4.11                  |

## 4.2 KD11-F MICROCOMPUTER

### 4.2.1 General

The KD11-F microcomputer is contained on a single 8.5 by 10 inch printed circuit board (M7264). The module includes all basic microcomputer functions common to both the KD11-F and KD11-J microcomputers and a resident 4K by 16-bit semiconductor read/write memory. KD11-F functions are shown in Figure 4-1.

#### NOTE

The following description reflects the circuits shown in drawing CS M7264 Rev. J.

### 4.2.2 Basic Microcomputer Functions

Basic functional blocks of the LSI-11 microcomputer are shown in Figure 4-1 and described in the following paragraphs. The KD11-F's resident memory is described separately (Paragraph 4.2.3).

**4.2.2.1 Microprocessor Chip Set**—The main function contained on the processor module is the microprocessor chip set. This chip set includes a control chip, a data chip, and two microinstruction ROM chips (microms). In addition, an optional KEV11 microm that contains EIS/FIS microcode can be installed on the module. Microprocessor chips communicate with each other over a special 22-bit microinstruction bus, WMIBO-21 L. All address and data communication between the microprocessor chips and other processor module functional blocks is via the data chip and the 16-bit data/address lines, WDALO-15 H (from the data chip).

Processor module control signals interface with the microprocessor chips via the control chip. Eight input and five output microprocessor control signals provide this function.

Timing and synchronization of all microprocessor chips (and all processor module functions) are controlled by four nonoverlapping clock pulses (Paragraph 4.2.2.2). Typical operating speed is 400 ns (100 ns each phase), based on a 10 MHz oscillator signal.

The control chip generates a sequence of microinstruction addresses which access the microinstruction microm chips. The addressed microinstruction is then transferred to the data and control chips. Most of the microinstructions are executed by the data chip; however, various jumps, branches, and I/O operations are executed in the control chip.

The data chip contains the data paths, logic, arithmetic logic unit (ALU), processor status bits, and registers that are most familiar to PDP-11 and LSI-11 users. Registers include the eight general registers (R0-R7) and an instruction register. The user's program has access to all general registers and processor status (PS) bits. All PDP-11 instructions enter this chip via the WDAL bus. Data and addresses to and from the microprocessor are also transferred to and from the processor over this 16-bit bus.

#### CAUTION

Do not remove processor chips from their sockets. Improper handling could permanently damage the chips.

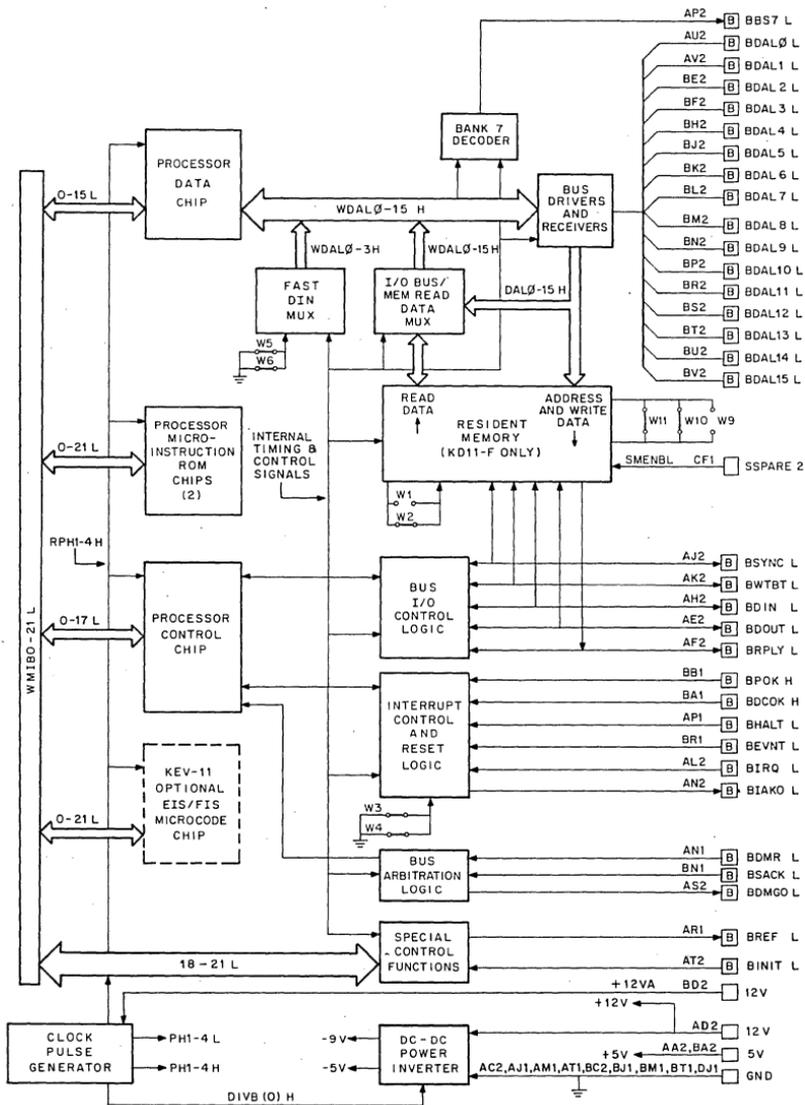


Figure 4-1 KD11-F Microrcomputer Logic Block Diagram

**4.2.2.2 Clock Pulse Generator**—The clock pulse generator produces four nonoverlapping clock signals for processor timing and synchronization. A voltage-controlled oscillator generates a basic 10 MHz CK H signal.

Maintenance clock gates receive and distribute the basic CK H signal to a two-stage counter and an RC filter circuit. The two-stage counter outputs are decoded by the four-state decoder, producing the basic four nonoverlapping clock phases. The pulse produced on the leading edge of each basic clock pulse inhibits the decoder for 10 ns, preventing the overlap of each phase. Each of the four phase signals (RPH1 through RPH4) are positive-going, MOS-compatible 100 ns (nominal) pulses which are used to each of the microprocessor chips through resistors. PH1 L through PH4 L and PH1 H through PH4 are similarly timed; however, they are TTL-compatible for distribution elsewhere on the module.

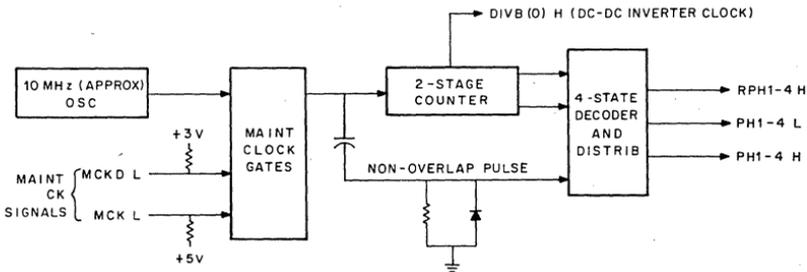


Figure 4-2 Clock Pulse Generator

11-3144

**4.2.2.3 Bus Interface and Data/Address Distribution**—All LSI-11 processor module communication to and from external I/O devices and memories is accomplished using the LSI-11 bus 16-bit data/address lines (BDALO-15 L) and bus control signals. The processor module interfaces to the bus using bus driver/receiver chips, as shown in Figure 4-3. Each DEC 8641 chip contains four open-collector drivers and four high-impedance receivers. Each driver output is common to a receiver input. Hence, either processor output data (from the driver outputs) or input data (from the bus) can stimulate bus receiver inputs.

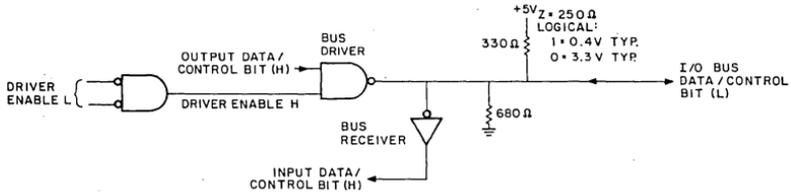
Note that all four drivers in a chip are enabled or disabled by a pair of DRIVER ENABLE L inputs. A high input will inhibit all four drivers; when both enable inputs are low, the drivers are enabled and output data is gated onto the bus. Signals which control bus drivers include EDAL L, INIT (1) H, and DMGCY H. False states enable certain control signals which are described later.

EDAL L is a control signal which enables the 16-bit data/address bus drivers. When in the active state, EDAL L gates WDALO-15 H onto the BDALO-15 L bus.

EDAL L is generated by the logic shown in Figure 4-4. During a processor-controlled address/data output bus cycle, or during the addressing portion of a processor-controlled input bus cycle, SACK L and DMG(1) L are passive (high). The passive signals are gated, producing

a low (passive) DMGCY H signal. This signal is inverted and gated with the passive DIN L signal, producing the active EDAL L signal. During a DMA cycle in which data in the processor module's resident 4K memory is to be read by a DMA device, BANK OR REF H goes high; this signal is gated with DINR H and DMG CYCLE H to produce the active EDAL L signal.

DMGCY H and INIT (1) H are processor module logic control signals which inhibit certain bus drivers during an Initialize or DMA operation. Bus drivers are enabled when these signals are in the false (low) state.



11-3145

Figure 4-3 LSI-11 Bus Loading and Driver/Receiver Interface

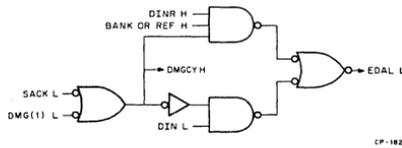


Figure 4-4 EDAL L Logic

A list of bus driver output signals and their respective enable signals is provided below.

| Bus Driver (Signal) | Enable Signal(s) (Low = Enable) |
|---------------------|---------------------------------|
| BDAL0-15 L          | EDAL L                          |
| BSYNC L             | INIT (1) H, DMGCY H             |
| BBS7 L              |                                 |
| BREF L              |                                 |
| BIAKO L             |                                 |
| BDMG L              |                                 |
| BRPLY L             | INIT (1) H                      |
| BDIN L              |                                 |
| BDOUL L             |                                 |
| BWTBT L             |                                 |
| BINIT L             | Always enabled                  |

The near-end bus termination resistors are contained on the processor module. Each bus driver output is terminated by a pair of resistors, as shown in the figure, establishing the nominal 250  $\Omega$  bus impedance and the 3.4 V nominal voltage level. No additional terminations are required for bus-compatible devices connected to the same backplane.

Address and data information are distributed on the processor module via the DWALO-15 H and DALO-15 H 16-bit buses. WDALO-15 H interface directly with the microprocessor's data chip, the DEC 8641 bus drivers, and the I/O bus/memory read data multiplexer. All processor input data from the I/O bus is via the bus receivers, the DALO-15 H bus, the data multiplexer, the WDALO-15 H bus, and the microprocessor's data chip. Resident memory data input is discussed later.

**4.2.2.4 Bus I/O Control Signal Logic**—Bus I/O control signals include BSYNC L, BWTBT L, BDIN L, BDOUT L, and BRPLY L. In addition, BIAKO L can be considered a bus I/O control signal; however, since it is only used during the interrupt sequence, it is discussed in Paragraph 4.2.2.6. Logic circuits which produce and/or distribute these signals are shown in Figure 4-5. Each signal is generated or received as described in the following paragraphs.

**BSYNC L**—The control chip initiates the BSYNC L signal sequence by raising WSYNC H during PH2. Inverters apply the high SYNC H signal to the Sync flip-flop D input. On the trailing edge of PH3 L, the Sync flip-flop sets, producing an active (high) SYNC (1) H input to the BSYNC L bus driver. SYNC (1) H is gated with REPLY (1) H (when active) to produce a direct preset input to the Sync flip-flop. This ensures that BSYNC L will remain active until after the bus slave device terminates its BRPLY L signal and the Reply flip-flop is reset. [REPLY (1) H is low.] The Sync flip-flop then clocks to the reset (BSYNC L passive) state on the trailing edge of PH3 L.

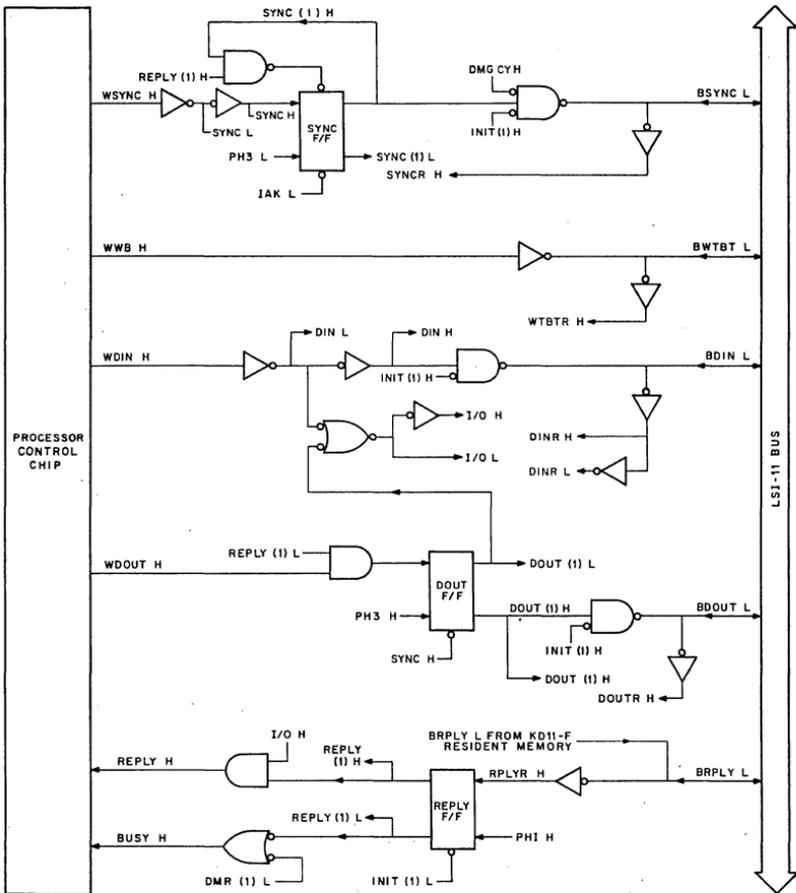
**BWTBT L**—BWTBT L is the buffered/inverted control chip WWB H output signal. This signal asserts during PH1 of the addressing portion of a bus cycle to indicate that a write (output) operation follows. It remains active during the output data transfer if a DATOB bus cycle is to be executed.

**BDIN L**—BDIN L is the inverted, buffered control chip's WDIN H signal. This signal goes active during PH2 following an active RPLY H signal.

**BDOUT L**—The control chip initiates the BDOUT L signal sequence by raising WDOUT H during PH2. This signal is gated with the passive REPLY (1) L (high) signal to produce an active low D input to the DOUT flip-flop. The flip-flop sets on the leading edge of PH3 H, producing an active BDOUT L signal. It clocks to the reset state on PH3 following the REPLY (1) L active (low) signal.

**BRPLY L**—BRPLY L is a required response from a bus slave device during input or output operations. DIN L and DOUT (1) L are ORed to produce an active I/O L signal whenever a programmed transfer occurs. I/O L enables the time-out counter in the bus error detection portion of the interrupt logic. I/O L is inverted to produce I/O H, which enables the reply gate REPLY H signal input to the control chip.

BRPLY L is received either from the LSI-11 bus or resident memory and inverted to produce a high input to the Reply flip-flop. PH1 H clocks the flip-flop to set state, producing active REPLY (1) H and REPLY (1) L signals. REPLY (1) L is ORed with DMR (1) L to produce an active BUSY H signal. The processor's control chip responds by entering a wait state, inhibiting completion of the processor-generated bus transfer for the duration of REPLY (1) L. REPLY (1) H is gated with I/O H to produce an active REPLY H signal, informing the processor that the output data has been taken or that input data is available on the bus. REPLY H goes passive when I/O H goes passive. The bus slave device will then terminate the BRPLY L signal, indicating that it has completed its portion of the data transfer. On the next PH1 H clock pulse, the Reply flip-flop resets and REPLY (1) H and L and BUSY H go passive.



11-3146

Figure 4-5 Bus I/O Control Signal Logic

**4.2.2.5 Bank 7 Decoder**—The bank 7 decode circuit is shown in Figure 4-6. Buffers receive WDALO-15 H bits and distribute them to the bank 7 decoder and BDAL bus drivers. Bank 7 is decoded during the addressing portion of the bus cycle. If a peripheral device address is referenced, an address in bank 7 (28-32K address space) is used, and WDAL13, 14, and 15 H are all active (high). This address is decoded and BBS7 L is asserted. When active, BBS7 L enables addressing of nonmemory devices along the bus. During interrupt vector bus transactions, IAK L becomes asserted. IAK L inhibits WDAL15 H, preventing BBS7 L signal generation, which could result in an invalid input data transfer.

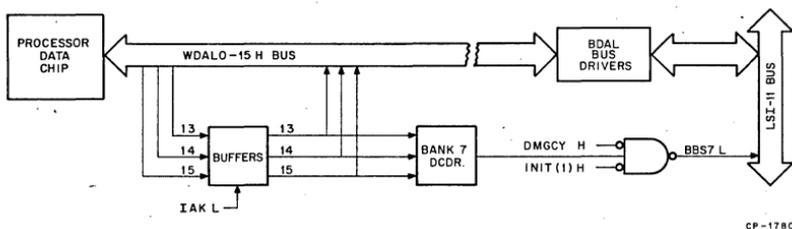


Figure 4-6 Bank 7 Decoder

CP-1780

**4.2.2.6 Interrupt Control and Reset Logic**—Interrupt control and reset logic functions are shown in Figure 4-7. Reset functions include bus error and power-fail (BDCOK H negated). Interrupt functions include power-fail (impending), Halt mode (console microcode control), refresh interrupt, event (or line time clock) interrupt, and external BIRQ interrupts. The various functions are described in the following paragraphs.

**Power-Fail/Restart Sequence**—A power-fail sequence is initiated when BPOK H goes low, clocking the Power-Fail flip-flop to the set state. PFAIL (1) L is ORed with HALT L to produce a high signal. This signal is latched during PH2 H, producing an active IPIRQ H (interrupt 1) input to the processor control chip. The processor then interrupts program execution. Note that the low (passive) BPOK H signal is inverted to produce an active PFAIL H input to the fast DIN multiplexer; this signal status is checked by the microcode to ensure that BPOK H is asserted.

Upon entry to this microcode routine, the processor requests a fast DIN cycle. This request is decoded as ROM CODE 15 L, presetting the fast DIN flip-flop. FDIN (0) H goes low, enabling the fast DIN multiplexer to place start-up microcode option jumper data, the passive time-out error [TERR (1) H] signal, and the active PFAIL H signal on WDALO-3 H. The processor receives the fast DIN information via the data chip. An active PFAIL H signal informs the processor that a power-fail condition is in progress, rather than the halt condition.

If the power failure continues, BDCOK H goes passive (low) and produces an active DC LO L signal, clearing the Power-Fail flip-flop and the power-fail/halt and reset latches and initializing the processor and all devices (Paragraph 4.2.2.1). The active RESET L signal then initializes

the processor, causing it to abort console (halt) or power-fail microcode execution and enter a "no operation" state. The processor remains in this condition until BDCOK H returns to the active state.

The power-up restart condition occurs when DC LO L goes false; RESET L goes passive (high) on the next PH2 H clock pulse. The processor responds by executing a fast DIN cycle to determine the start-up microcode option jumper configuration. Once the fast DIN cycle has been completed, the processor executes the power-up option selected, and normal operation resumes when BPOK is asserted.

**Halt Mode**—The Halt mode is entered by executing the HALT instruction, by a device asserting the BHALT L signal, by a double bus error condition, or by a bus error (time-out) during an interrupt. The processor halts program execution and enters microcode execution as described for a power-fail operation. However, when the processor executes the fast DIN cycle, the PFAIL H bit (WDAL3 H) is not active and console microcode (not a power-fail sequence) is executed. Negation of BHALT L will allow the processor to resume PDP-11 program execution. On the next PH2 H clock pulse, IPIRQ H goes false (low) and the processor Run mode is enabled.

**Bus Errors**—A bus error results in aborting program execution and entry into a trap service routine via vector location 004. A bus error occurs when a device fails to respond to the processor's DBIN L or BDOU L signal by not returning a BRPLY L signal within 10  $\mu$ s (approximately). An active I/O signal inhibits the reset input of the 5-stage time-out counter, enabling counter operation. [When not in a processor-controlled bus I/O cycle, I/O L is passive (high), clearing the counter.] The counter proceeds with counting PH3 H clock pulse signals. Normally BRPLY L would be asserted, producing an active REPLY (1) H signal which inhibits the counter; the count would remain stable until cleared by a passive I/O L signal. However, if BRPLY L is not received within 10  $\mu$ s, the full count (32<sub>0</sub>) is attained. This is the error condition; TERR L goes low and TERR (1) H goes high. The next PH2 H clock pulse clocks the reset latch to the reset (active) state, producing an active RESET L signal. The processor responds by executing the reset microcode. After entering the microcode, the processor executes a fast DIN cycle and determines that a time-out (bus) error TERR (1) H, rather than a power-fail condition, has occurred. It then responds by executing the bus error trap service routine. TFCLR L (ROM code 2) is generated by the processor to clear the TERR latch.

**Normal I/O Interrupts**—"Normal" I/O interrupts are those interrupt requests which are generated by external devices using bus interrupt request BIRQ L. The request is initiated by asserting BIRQ L. This signal is inverted to produce a high signal, which is stored in the interrupt request latch on the next PH2 H pulse. The stored request produces IOIRQ H, which informs the processor of the request. If processor status word priority is 0, the processor responds by producing an active WIAK H (interrupt acknowledge) and WDIN H signals. WDIN H is buffered onto the BDIN L signal line to signal devices to stabilize their priority arbitration. WIAK H is inverted, producing IAK L, setting the Interrupt Acknowledge flip-flop on the trailing edge of PH1 L one cycle after

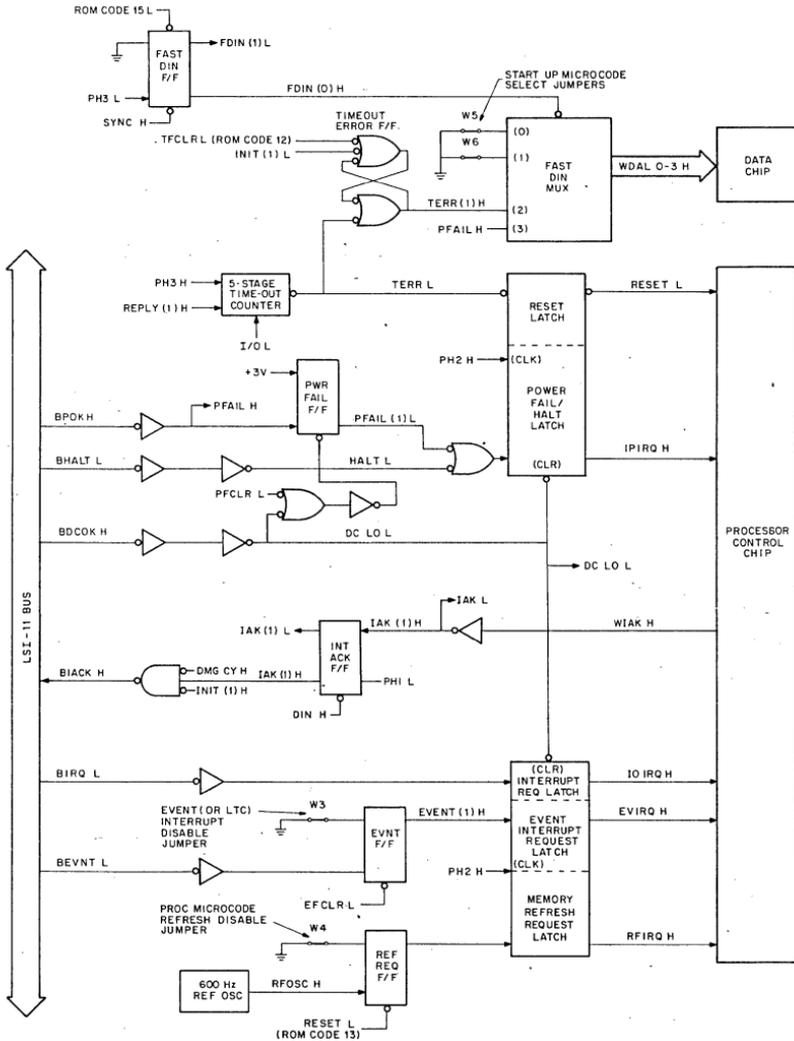


Figure 4-7 Interrupt Control and Reset Logic

BDIN L is asserted. The high (active) interrupt acknowledge signal is enabled onto the BIAKO L signal line by passive (low) DMGCY H and INIT (1) H signals. The highest priority device requesting interrupt service responds to the processor's BDIN L and BIAK L signals by placing its vector on the BDAL bus and asserting BRPLY L, inputting its vector to the processor. Note that BSYNC L is not asserted during this operation and that no device addressing occurs. The device also clears its

BIRQ L signal. The processor responds to BRPLY L by terminating BDIN L and BIAK L.

**Refresh**—Memory refresh is initiated by a 600 Hz refresh oscillator. This function is enabled when jumper W4 is not installed. The leading edge of RFOSC H clocks the Refresh Request flip-flop to the set state. On the next PH2 H clock pulse, the memory refresh request latch stores the request and applies an active RFIRQ H signal to the processor's control chip. The processor responds by producing an active RF SET L signal and executing the refresh microcode. RF SET L sets the Refresh flip-flop, producing the BREF L signal (Paragraph 4.2.2.7) and clearing the Refresh Request flip-flop, which terminates the request. TFCLR L resets the Refresh flip-flop when the refresh operation is completed. Note that BREF is not asserted if DMGCY H or INIT (1) H is asserted.

**Event Line Interrupt**—The event line interrupt function can be used as a line time clock interrupt, or as desired by the user. This interrupt differs from the normal I/O interrupt request by being the highest priority external interrupt, and it does not input a vector in order to enter its service routine. The interrupt is initiated by the external device by asserting BEVNT L. This signal is inverted to produce a high (active) signal, which clocks the Event flip-flop to the set state. (Note that when W3 is installed, the flip-flop remains reset and the event function is disabled.) On the next PH2 H clock pulse, the event interrupt request latch stores the active EVNT (1) H signal. An active EVIRQ H signal is then applied to the control chip. If processor status word priority is 0, the interrupt will be serviced. Service is gained via vector 100<sub>8</sub>, which is dedicated to the event interrupt. Hence, a bus DIN operation does not occur when obtaining the vector. The request is cleared by the microcode generated EFCLR L signal.

**4.2.2.7 Special Control Function**—Special control functions include microcode-generated bus initialize and memory refresh operations and five special control signals which are internally on the processor module. Special control function logic circuits are shown in Figure 4-8. Microinstruction bus lines WMIB18-21 L are buffered to produce the four SROM0-3 H signals. The actual codes for the special functions are contained on SROM0-2 H; SROM3 H is always active when a special function is to be decoded, enabling the 1 of 8 decoder during PH3 H. The resulting decoded functions are described below.

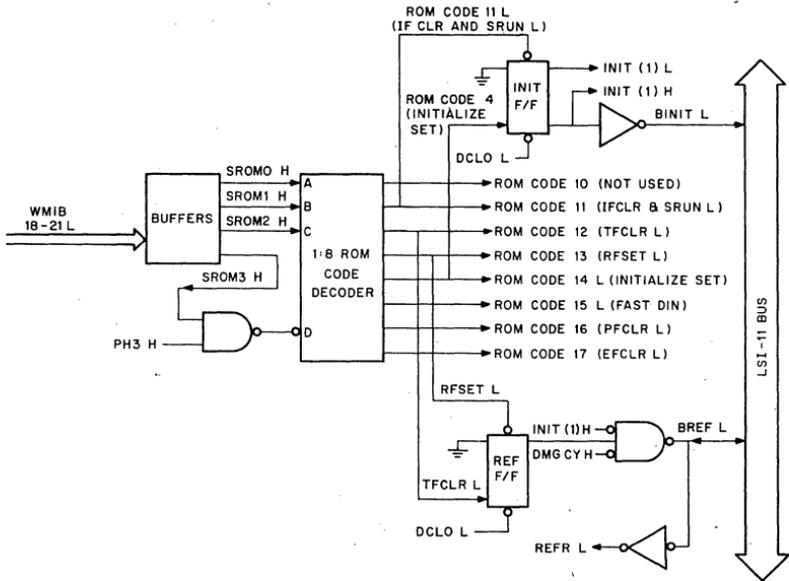
**ROM Code 10**—Not used.

**ROM Code 11 [IFCLR and SRUN L]**—This code is produced by the processor to clear the Initialize flip-flop and to assert the SRUN L signal for a RUN indicator in PDP-11/03 systems.

**ROM Code 12 [TFCLR L]**—This code is a trap function clear signal which clears the Refresh Request and Time-Out Error flip-flops (Paragraph 4.2.2.6).

**ROM Code 13 [RFSET L]**—This code is used to set the Refresh flip-flop. The active (high) flip-flop output is gated with passive (low) INIT (1) H and DMG (1) H signals to produce the active BREF L signal. The flip-flop normally resets by the microcode-generated TFCLR L signal after

completing the refresh operation, or whenever a power failure occurs. (DC LO L goes active and clears the flip-flop.)



11-3148

Figure 4-8 Special Control Functions

**ROM Code 14 [Programmed Initialize]**—A programmed LSI-11 bus initialize operation can be performed by executing the RESET instruction. The processor responds by generating ROM Code 14 L (decoded). On the positive-going trailing edge of this signal, the Initialize flip-flop clocks to the reset (active) state, producing the active initialize signal. Approximately  $10\mu\text{s}$  later, the processor produces a TFCLR L signal, clearing the initialize signal.

During a power failure, the active DC LO L signal is distributed to the Initialize flip-flop clear input; when cleared, the flip-flop is in the active state and INIT (1) H, INIT (1) L, and BINIT L initialize signals are used to clear (or initialize) all LSI-11 system logic functions. When normal power resumes, the processor microcode terminates the initialize cycle by generating TFCLR L, presetting the Initialize flip-flop; this is the passive (noninitialize) or normal flip-flop state and all initialize signals return to their passive states.

**ROM Code 15 [Fast DIN Cycle]**—The processor generates this code when a fast DIN cycle is required. The fast DIN cycle allows the processor to read (input) the selected start-up mode, time-out error, and power fail signal status (Paragraph 4.2.2.6).

**ROM Code 16 [PFCLR L]**—This code clears the Power Fail flip-flop (Paragraph 4.2.2.7).

ROM Code 17 [EFCLR L]—This code clears the Event flip-flop (or line time clock interrupt request) (Paragraph 4.2.2.7).

**4.2.2.8 Bus Arbitration Logic**—Bus arbitration logic (Figure 4-9) enables the LSI-11 bus to be used by DMA devices or the processor. The device (or processor) controlling the bus is called the bus master. When no DMA requests are pending, the processor is bus master and all data transfers are programmed. When a DMA device is bus master, processor operation is suspended until the DMA operation is finished.

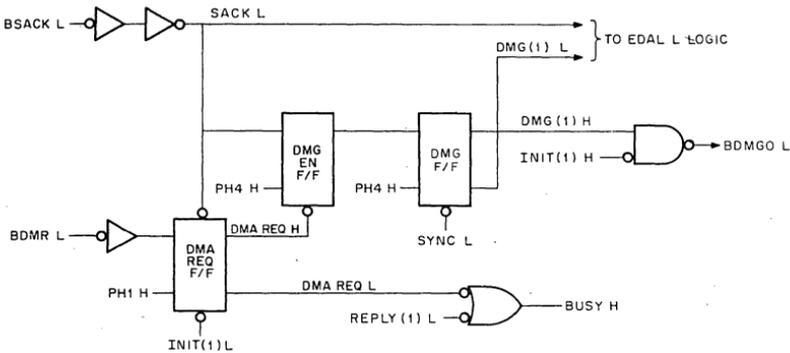


Figure 4-9 Bus Arbitration Logic

Prior to a DMA request, the DMA Request flip-flop is reset (Figure 4-10); the DMA REQ H signal is passive (low), clearing the DMG Enable flip-flop. A device initiates a DMA request by asserting BDMR L. The request is inverted to produce a high signal, which is clocked into the DMA Request flip-flop on the next PH1 H clock pulse, producing active DMA REQ H and L signals. DMA REQ L is ORed with REPLY (1) L, producing BUSY H and causing the processor to “wait” after completing its present bus cycle. On the leading edge of PH4 H, the stored DMA request sets the DMG Enable flip-flop. The processor is finished with its present bus cycle and releases the bus when SYNC L goes passive (high).

On the first PH4 H clock pulse following the passive state of SYNC L, the DMG flip-flop clocks to the set state and DMG (1) H and DMG (1) L go to their active states. DMG (1) H produces the active BDMG grant (BDMGO L) signal. DMG (1) L enables EDAL L signal generation when the DMA operation involves KD11-F resident memory. The DMA device responds to the BDMG signal by negating BDMR L and asserting BSACK L, enabling EDAL L signal generation and keeping the DMA Request flip-flop in the set state. On the first PH4 H clock phase following the active state of BSACK L, the DMG Enable flip-flop clocks to the reset state and DMG EN H goes low. The following PH4 H clock pulse clocks the DMG flip-flop to the reset state and BDMGO L goes passive (high), terminating the DMA request/grant sequence. BSACK L remains asserted for the duration of the DMA operation, preventing new DMA requests from being arbitrated.

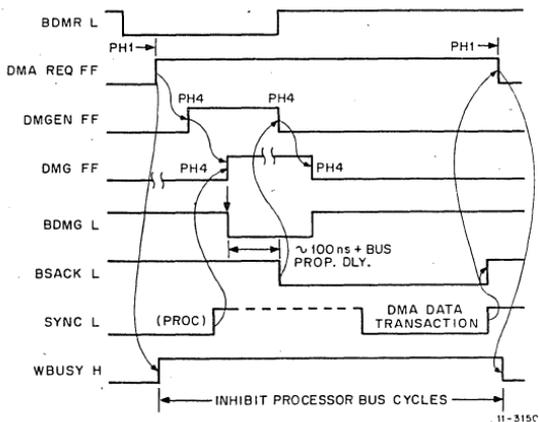


Figure 4-10 DMA Grant Sequence

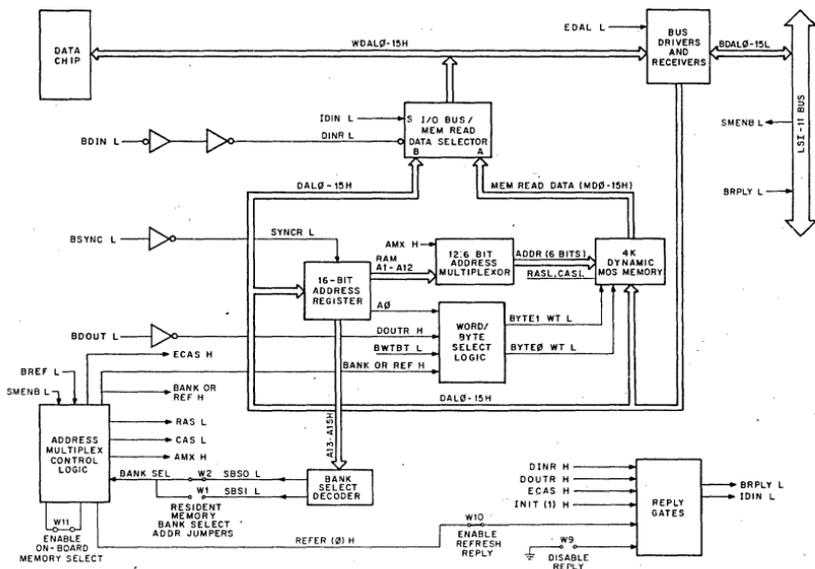


Figure 4-11 KD11-F Resident Memory

The DMA device releases the bus by terminating BSACK L. The following PH1 H clock pulse clocks the DMA request flip-flop to the passive state. BUSY H then goes passive, enabling a processor-initiated bus cycle. Once the processor-initiated cycle is entered, SYNC L inhibits (clears) the DMG flip-flop for the duration of the processor's present bus cycle.

### 4.2.3 KD11-F Resident Memory

The 4K by 16-bit dynamic MOS read/write memory is included on the KD11-F processor module only. (KD11-J basic memory is magnetic core, which is contained on a separate MMV11-A core memory unit.) Resident memory can reside in either the first or second 4K address bank. One of two jumpers can be installed on the module to select the desired bank (bank 0 or 1).

The basic functions involving the resident memory are shown in Figure 4-11. Resident memory comprises sixteen 4K by 1-bit memory chips, addressing, and control logic. The memory chips, which are 16-pin devices, require an address multiplexer to address the chips with two 6-bit bytes. The complete addressing, write, and read operations are described below.

Addressing is initiated by a master device—either the LSI-11 processor or a DMA device—by placing the 16-bit address on BDAL0-15 L and asserting BSYNC L, latching the address in the 16-bit address register. Note that the resident memory address will appear on the BDAL bus even when the processor is bus master; the resident memory functions exactly as a memory located elsewhere along the LSI-11 bus. Address bits are routed via BDAL bus receivers onto the processor module's DAL0-15 H bus to the address register input. Stored address bits A13—A15 H are then decoded by the bank select decoder. SBS0 L (bank 0) and SBS1 L (bank 1) will go active (low) only when their respective bank addresses are decoded. W1 or W2 and W11 then applies the selected address to the address multiplex control logic, enabling the resident memory response. Address multiplex logic immediately generates an active row address strobe (RAS), which remains active for the duration of the BSYNC L signal. Address multiplex control (AMX) is initially high, multiplexing the stored row address (bits A7—A12 H) through the 12:6-bit address multiplexer and into all memory chips. After 150 ns, address multiplex control logic generates an active column address strobe (CAS) and a low AMX signal. The multiplexer output bits (1-A6) are then strobed into all memory chips, completing the addressing portion of the memory operation.

Resident memory bank selection can be accomplished, by an external signal. W11 is removed to disable the on-board memory bank selection. SMENB L is then asserted low by the external circuit to select the resident memory.

When in a memory read operation, each of the 16 memory chips places an addressed bit on the memory read data bus. This data is multiplexed via port A of the I/O bus/memory read data selector only when in a resident memory read (or refresh) operation; the select input of the data selector is asserted low for this data selection. The read data is then placed on WDAL0-15 H, where it can be read by the microprocessor data chip or gated onto the BDAL bus via bus drivers for input to a DMA device.

When in a memory write operation, the addressing portion of the operation is similar to the read cycle addressing, except BWTBT L may be asserted by the master device to indicate that a write operation is to

follow. After the addressing portion of the cycle has been completed, BWTBT L either goes passive (high) if a DATO (word) write cycle is to be performed, or remains asserted (BWTBT L remains low) if a DATOB (byte) write cycle is to be performed. Word 1 byte select logic responds to the DATO cycle by asserting both BYTE 1 WT L and BYTE 0 WT L for the duration of the cycle, enabling DAL0-15 H data bits into the addressed location in all memory chips. However, when in a DATOB cycle, only one active signal is produced, depending upon the state of the stored byte pointer (address bit A0). If A0 is low (even byte), only BYTE 0 WT L goes active, enabling only DAL0-7 H bits to be written into the addressed location in the appropriate eight memory chips. Similarly, if A1 is high (odd byte), only BYTE 1 WT L goes active, enabling only DAL8-15 H bits to be written into the addressed location in the appropriate eight memory chips.

Resident memory, as well as any LSI-11 bus device, must respond to any data transaction by generating an active BRPLY L signal. Reply gates provide this function, Approximately 150 ns after CAS L goes true (as previously described), the reply gates are enabled; the gates will respond to either an active BDIN L or BDOUT L signal by asserting BRPLY L. Reply gates are inhibited during an initialize operation.

Reply gates can be disabled by installing W9. When installed the resident memory will not assert BRPLY L. W10 can be removed to inhibit the reply during memory refresh operations only. REFR (0) 4 normally enables the refresh reply.

Resident memory requires a refresh operation once every 1.6 ms. This operation is entirely under the control of either processor microcode or an external DMA device, as selected by the user. Resident memory responds to BREF L, generated by the refresh-controlling device, by simulating a "bank selected" operation. (All memory banks are simultaneously refreshed.) Refresh is then accomplished by executing 64 successive BSYNC L/BDIN L operations while incrementing BDAL1-6 L by one location on each bus transaction. Refresh is simply a series of forced memory read operations where only the row addresses are significant. Each of the 64 rows in all dynamic MOS memory chips in an LSI-11 system are simultaneously refreshed in this manner.

#### **4.2.4 DC-DC Power Inverter**

The dc-to-dc power inverter circuit provides on-board generation of required negative dc voltages. Input dc power for the inverter circuit is obtained directly from the +12 V input. The inverter switching rate is clocked by the clock pulse generator's DIVB (0) H 2.8 MHz output. Output negative dc voltages are distributed to all resident memory chips. The -5 V output is distributed to microprocessor chips (data chip, microm chips, and control chip).

### **4.3 MMV11-A 4K BY 16-BIT CORE MEMORY**

#### **4.3.1 General**

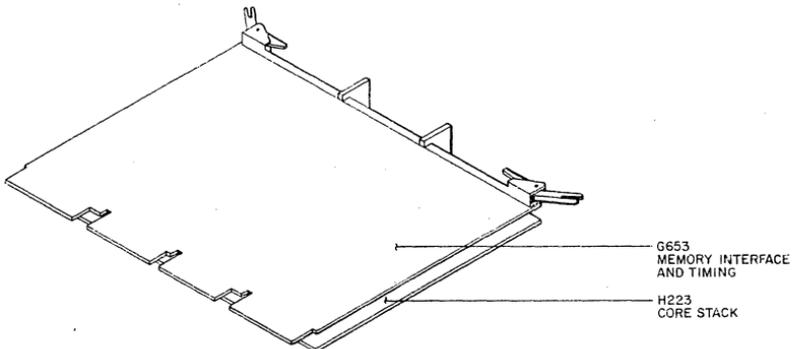
The MMV11-A 4K by 16-bit core memory option provides nonvolatile read/write storage of user programs and data. Memory 4K addressing is user-selected by switches contained on the option. The MMV11-A is completely LSI-11 bus-compatible and capable of either programmed

I/O data transfers with the processor or transfers with another LSI-11 DMA bus device.

The MMV11-A features:

- 4096 by 16-bit capacity
- Typical access time = 425 ns (475 ns maximum); full read/restore cycle time = 1.15 $\mu$ s.
- Nonvolatile read/write storage—stored data remains valid when power is removed.
- User-selected bank address—three switches allow the user to select the bank address for the option.
- +5V and +12 V power—only the normal backplane power is required to power the option.
- No adjustments, no periodic maintenance.

The MMV11-A is contained on two modules which are mated to comprise a single assembly as shown in Figure 4-12. The modules include memory interface and timing board (module type G653) and core stack (module type H223). The actual size of the assembly is 8.5 by 10 by 0.9 in. The G653 module includes handles and retractors on the top edge and fingers on the bottom edge which plug into the LSI-11 bus. Circuits contained on this module include interface, control and timing logic, bus receivers and drivers, the 16-bit data paths, sense amplifiers, and a +5 V dc to -5 Vdc inverter. The H223 module is slightly smaller, includes no handles or bus fingers, and plugs onto the No. 2 (solder) side of the G653 module via special connector pins. Spacers are located between the modules to stiffen the assembly and to maintain the 0.9 in. dimension. Circuits contained on the H223 module include the 4096 by 16 core stack, 12-bit address register, X and Y drives, stack charge, temperature compensation, and a series +11 V, Vcc switch which removes drive power when BDCOK H goes low (power fail) or BINIT L is asserted.



CP-1781

Figure 4-12 MMV11-A Core Memory Option

## 4.3.2 Functional Description

**4.3.2.1 Introduction**—The MMV11-A memory is a read/write, random access, coincident current magnetic core type with a cycle time of 1.15  $\mu$ s and an access time of 425 ns. It is organized in a 3D, 3-wire planar configuration. Word length is 16 bits and the memory consists of 4096 (4K) words.

Major functions contained in the MMV11-A are shown in Figure 4-13. Memory data can be stored (written) or read by executing appropriate bus cycles: DATO (16-bit word) write; DATOB (8-bit byte) write; DATI (16-bit word) read; DATIO (16-bit word) read-modify-write; and DATIOB (16-bit word) read-modify-(8-bit byte) write.

Each of the functions shown in Figure 4-13 is briefly described below:

**Bus Receivers and Drivers**—These devices interface directly with the LSI-11 bus and the G653 logic circuits. BDAL bus drivers are gated on by DATA OUT L during a read operation [DATI or the input portion of a DATIOB(B) bus cycle].

**Bank Decoder**—The bank decoder receives address bits A 13-15 L and responds when the bank address is as user-selected on the three bank address switches on the G653 module. It responds by producing an active DSEL H signal which initiates memory cycle timing. This signal is enabled only when power is normal and bus initialize or refresh operations are not in progress.

**Timing and Control**—Timing and control circuits receive bus and internal control signals and generate appropriate read/write timing and control signals. It also generates the BRPLY L signal in response to BDIN L and BDOUL.

**Address Register**—The address register stores the 12-bit word address within the 4K bank during the addressing portion of the bus cycle. Latched bits LA1—6H are applied to Y drive circuits and LA7—12H are applied to X drive circuits.

**X and Y Drives**—X and Y drive circuits control X and Y read/write currents through all core mats. Address decoding activates 1 out of 64 X wires and 1 out of 64 Y wires. Because the active X and Y wires each have one-half the current required for core saturation, only one core out of 4096 cores in each core mat is saturated. Direction of current is determined by a read or write operation.

**Core Stack**—The core stack comprises sixteen 4096-core mats. Each mat is associated with one memory bit position at all 4096 locations. Each core has three wires passing through it: one X, one Y, and one sense/inhibit wire. The sense/inhibit wire passes through all 4096 cores in one mat. Hence, the stack contains 16 sense/inhibit lines.

The sense/inhibit line ends terminate at sense amplifier inputs. During a write operation, an inhibit current, equal to saturation current, is applied to the center of the sense/inhibit line when a logical 0 is to be written in the addressed core. This current splits and one-half saturation current flows through all cores in the mat and into termination diodes at the sense amplifier inputs. The wire is threaded through the cores in

a manner that causes the current to flow in a direction opposite to that of the Y write current; this prevents core saturation, which would write a logical 1 in the addressed core.

**Sense Amplifiers**—Sense amplifiers respond to induced voltage impulses during the read cycle. They are strobed during a critical time of the cycle, producing an active (high) output when a logical 1 is read, regardless of the induced polarity on the two ends of the sense/inhibit wires for each bit.

**Inverters**—The inverters receive sense amplifier outputs, invert them, and direct-set previously cleared memory data register bits when a logical 1 is sensed.

**Memory Data Register**—The 16-bit memory data register is cleared upon entry to a read cycle; sensed logical 1s set appropriate bits. During a restore cycle (DATI bus cycle) (no memory contents are to be modified), the same bits (low-active) are written into the same addressed location. During a write cycle [DATO, DATOB, or the write portion of a DATIO(B) bus cycle], bus data bits are clocked into the high and/or low byte(s), depending upon the type of bus cycle (word or high byte or low byte).

**Inhibit Drivers**—Inhibit drivers, one for each bit position, produce an inhibit current during the write cycle at INH TIME H if a logical 0 is to be written. The current inhibits core saturation, which would produce a stored logical 1.

**Charge Circuit**—The charge circuit applies the correct operating voltage to X and Y drive circuits during the read and write memory cycles to prevent "sneak" currents through unselected stack diodes.

**X—Y Temperature Compensation**—X—Y temperature compensation circuits alter drive currents over the required operating temperature range to provide reliable operation.

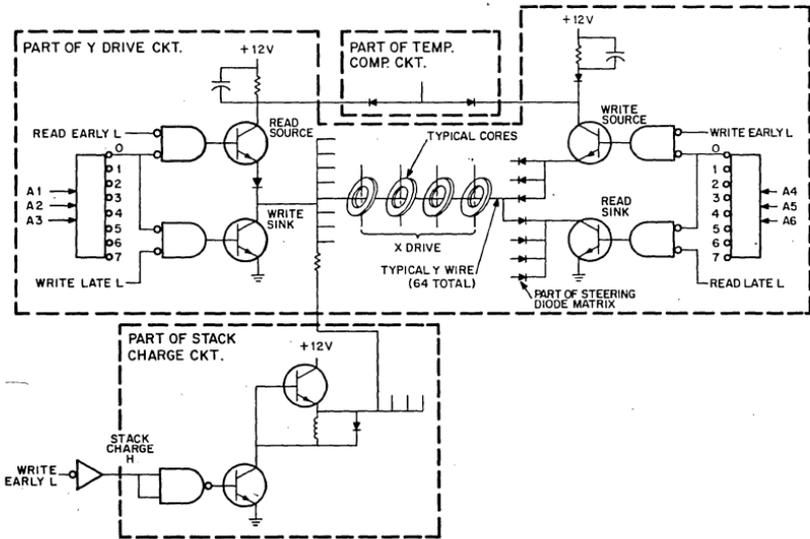
**DC—DC Inverter**—The dc—dc inverter circuit generates -5V power for sense amplifiers from the +5 V power.

**DC Protection**—DC protection circuits respond to an active BINIT L or passive BDCOK H signal by producing active LOCKOUT L, RESET H, RESET L, and passive VCC2OK H signals. These signal conditions prevent memory circuit operation and the possible loss of stored data.

**Vcc Switch**—The Vcc switch applies +11.5 V to X and Y driver circuits when not in an initialize or power fail condition.

**4.3.2.2. Core Addressing**—When a memory location is addressed, one core in each of the 16 mats are accessed for a read or write operation. Figure 4-14 illustrates a portion of the X—Y drive and associated circuits for one Y wire. Six address bits (A1—A6) select 1 of 64 Y wires. A similar circuit (not shown) involving the remaining six address bits (A7—A12) selects 1 of 64 X wires. Hence, by placing 64 cores (in each mat) on each Y wire and passing a different X wire through each core, one of 64 cores on the active Y wire will be selected. Since the remaining Y wires have a similar 64 cores each and receive the same X wires,  $64 \times 64$ , or 1 out of 4096 addressing is accomplished in each of the core mats. A single Y wire is driven as described below.





CP-1783

Figure 4-14 MMV11-A Core Addressing

Two 1:8 (octal) decoders are used in Y wire selection, each receiving three address bits from the address register. Only one output from each decoder will be active during addressing. Assuming address XX00 (the zeros are the Y portion of the 12-bit address), the portion of the Y drive circuit shown will be enabled. During a read operation, READ EARLY L goes active and turns on one of the eight read current source transistors. A diode in its emitter circuit couples the drive to eight Y wires, each terminating at the diode steering matrix. The diodes provide a read current path to all eight read sink transistors. READ LATE L goes active 25 ns after the Y source is turned on, and turns on one of the eight read sink transistors, completing a read current path to ground. Hence, 1 of 64 Y wires is selected, producing a read half-current through 64 cores in all memory mats. Similar X drive circuits will produce an X read half-current in 64 cores in each mat in exactly the same manner. Only one core in each mat will receive an X and a Y read half-current, causing the core to saturate in the 0 state. If the core was previously in the 1 state, a voltage pulse will be induced in the sense/inhibit wire as it switches to the 0 state.

A write cycle is always preceded by a read cycle. The write operation is similar to the read operation, except write current flows through the addressed wire in a direction opposite to the read current direction. The core in each mat receiving X and Y write half-currents will respond by saturating in the 1 state. However, since a 0 may be desired, a third wire (sense/inhibit) will conduct a half-current which opposes the magnetizing effect of the Y write currents. Thus, core saturation is not

attained and the cores where 0s are written remain saturated in the 0 state from the previous read cycle.

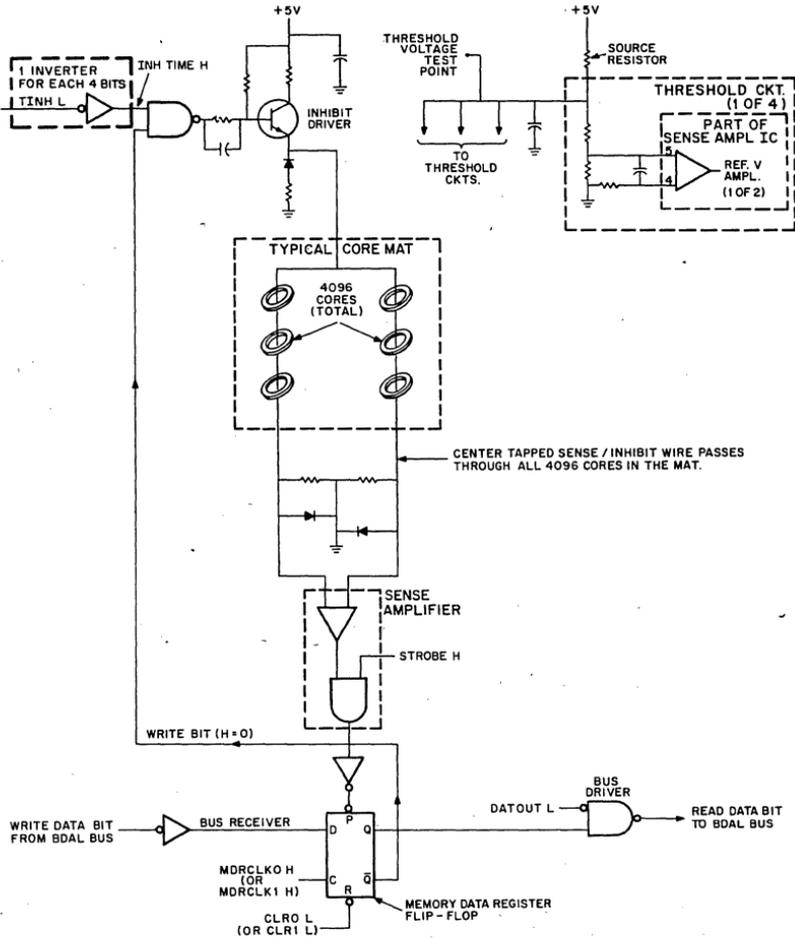
Temperature compensation is applied to driver circuits via a source current, which is inversely proportional to temperature; an increase in temperature decreases available drive current.

The stack charge circuit applies a +11 V (approximately) signal to the sink ends of all X (not shown) and Y wires during the write cycle. The level is applied during WRITE EARLY time. Since WRITE LATE L occurs 25 ns after WRITE EARLY L, the write sink transistor is cut off, and the full 11 V signal charges the stray capacitance of the X-Y lines, reducing the capacitive delay effect as the X and Y write source transistors turn on; the 11 V signal also reverse biases diodes not selected by addressing circuits, preventing sneak current. The addressed sink transistor, turned on by the active WRITE LATE L signal, provides the return path for the selected X and Y wires; only those two wires will go to approximately 0 V, causing one X and one Y diode to become forward biased, enabling the write half-currents to flow. Resistors coupling the charge voltage to write sink transistors limit the charge current through the addressed write sink transistors during the remainder of the write cycle. This circuit performs the same function for read cycles by grounding the buses and preventing sneak currents through unselected stack diodes.

**4.3.2.3 Read/Write Data Path**—The basic read/write data path is shown in Figure 4-13. Upon entering a read cycle, the memory data register is cleared by CLR0 and CLR1 L. X and Y read currents produce active sense amplifier outputs for those cores containing stored logical 1s as they are switched to the 0 states. These signals are inverted and applied to the direct-set inputs of the flip-flops comprising the memory data registers, setting the appropriate bits. During a write cycle, CLR0 L (DATOB low byte address), or CLR1 L (DATOB high byte address), or both CLR0 and CLR1 L (DATO word address) clear the previously read data. The bus data is then received and clocked into the register flip-flops by CLK MDROH and/or CLK MDR1 H, as appropriate. Write data bits are then routed to inhibit drivers which inhibit writing 1s when write bits are 0s (high). Inhibit half-current through addressed cores prevents X-Y write half-currents from switching cores to the 1 state.

The sense/inhibit wire passes through all cores in a core mat, as shown in Figure 4-15. The circuit shown in the figure is repeated for each of the 16 core mats. During the read portion of a memory cycle, a logical 1 stored in the addressed core will cause an induced voltage to appear on the sense/inhibit wire as the core switches from the 1 saturation state to the 0 saturation state. If a 0 was previously written, no appreciable voltage is produced since the core is already saturated in the 0 state. During the read operation, the sense/inhibit wire functions as a loop whose ends terminate at the sense amplifier inputs. Any difference in potential (either polarity) will enable a sense amplifier output. STROBE H occurs during X and Y drive read currents at a critical time (the time of peak core switching output when 1s are read). Thus, only the correct voltage pulse produced when a core goes from the 1 state to the 0 state is gated into the Memory Data Register flip-flop.

The threshold circuit establishes the signal voltage level at which a logical 1 is read during strobe time. A signal voltage magnitude greater than approximately 17 mV during strobe time results in a valid 1 level.



CP-1784

Figure 4-15 MMV11-A Read/Write Bit Data Path

Signal levels less than the 17 mV threshold value are considered invalid and result in 0 levels being read. Four threshold circuits share a common source resistor. Each threshold circuit provides a reference amplifier input voltage to two sense amplifier ICs, each containing two sense amplifiers; hence, one threshold circuit provides a threshold voltage for four data bits.

When in the write portion of the memory cycle, the inhibit driver remains off if a 1 write data bit is stored in the memory data register flip-flop. However, if a 0 is to be written, the write bit is high, enabling a gate input for the inhibit driver. At INH TIME H during the write cycle, the inhibit driver produces an inhibit current equal to core saturation in a direction that would produce a logical 0. However, note that the inhibit current is applied to the center of the sense/inhibit wire. Thus, half-currents flow into each half of the sense/inhibit wire, preventing the addressed core from saturating in the 1 state. Diodes at the sense amplifier ends of the wire provide a ground return for the two inhibit half-currents. The two resistors terminate the ends of the wires. The inhibit driver transistor collector is clamped to ground through a diode and resistor to prevent breakdown during turnoff. The emitter resistor limits peak current.

**4.3.2.4 Timing and Control**—All memory bus cycles comprise a read and a write operation. During a DATI bus transaction, a memory read-restore cycle is executed. The data is first read and placed on the I/O bus. The same data is then written in the same addressed location. During a DATO bus transaction, a memory read-modify-write cycle is executed. After reading the contents of the addressed location, bus data is clocked into the memory data register. Previously read data is lost. The modified word is then written into the addressed location during the remainder of the cycle. If a DATOB bus transaction is being executed, only an 8-bit portion of the memory data register is modified, and one byte of the previously read word is retained for the write operation. A DATIO bus transaction actually initiates two separate memory cycles. The first cycle (read-restore) is initiated by the master device by placing the memory address on BDAL0—15 L and asserting BSYNC L. After receiving and modifying the memory read data, the master device outputs the new data to the memory and asserts BDOUT L, which initiates the next memory cycle (read-modify-write). Timing and control logic functions generate all of the timing and control signals for the memory cycles described above. Logic operation for each type of bus transaction is described in detail in the following paragraphs.

A memory cycle is initiated when the correct bank address asserted by the bus master device is decoded on the leading edge of BSYNC L. DSEL H is the decoded bank address signal; note that it is inhibited during refresh bus cycles (when BREF L is asserted), or when an initialize or power fail condition exists. The logical state of DSEL H is clocked into the Busy flip-flop on the leading edge of SYNC H (Figure 4-16). When DSEL H is active (high), the Busy flip-flop sets and FBUSY H and FBUSY L go to their true states. FBUSY L enables one input of the read initiate gate. The remaining gate input is enabled by the negative-going pulse produced by the RC circuit connected to FBUSY L. Thus, on the leading edge of FBUSY L, the state of FBUSY H is clocked into the Read flip-flop, causing it to go to the set state. This sequence is shown in Figures 4-17 and 4-18.

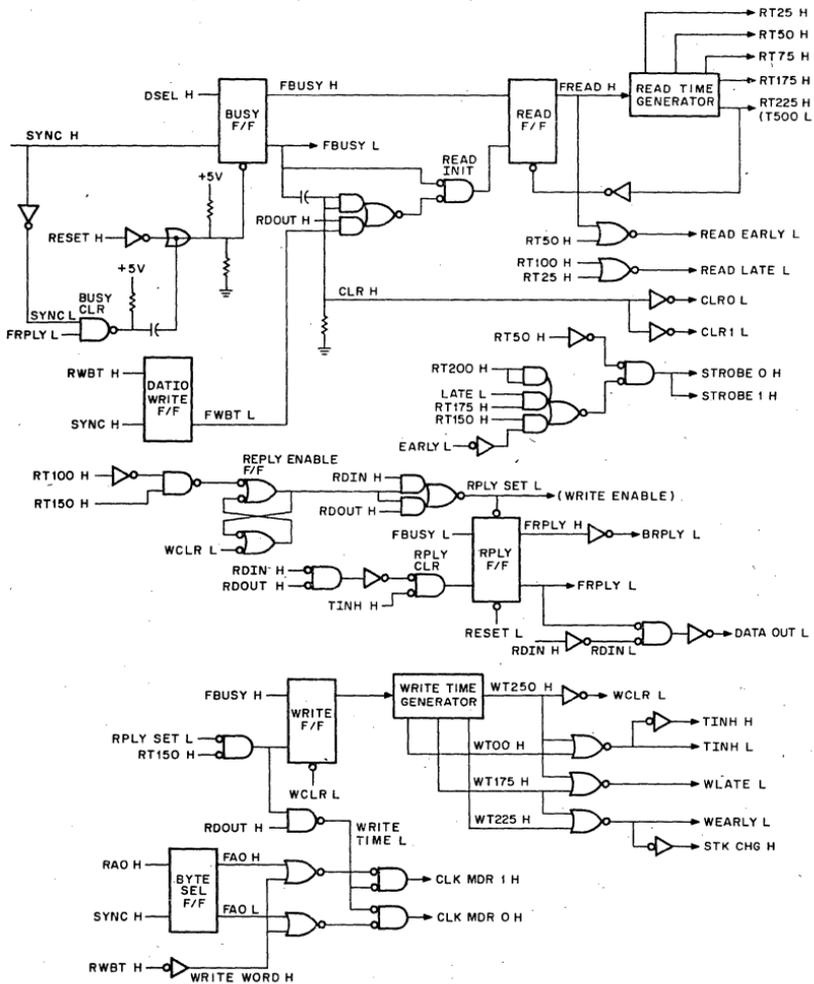
The read-restore (DIN) cycle continues as shown in Figure 4-17. FREAD H activates the read time generator, producing time signals prefixed with "RT." Each signal is a 225 positive-going pulse whose leading edge is delayed with respect to FREAD H. Hence, the leading edge of

RT225 H, shown in Figure 4-16, occurs 225 ns after the leading edge of FREAD H, and approximately 275 ns after BSYNC L is asserted. Note that RT225 H is inverted and applied to the clear input of the Read flip-flop, establishing the 225 ns pulse width for RT pulses. RT225 H goes low, 225 ns later. This time occurs 400 ns (total after BSYNC L is asserted and it is referenced on Figure 4-16 as (T500L).

The pulse produced by the RC network on the leading edge of FBUSY L is inverted to produce the CLR0 L and CLR1 L signals, which clear the memory data register for the new read data. READ EARLY occurs on the leading edge of FREAD H and remains active for the duration of RT50 H, producing a 300 ns pulse. RT25 H goes high 25 ns later, producing the READ LATE L signal; this signal remains true for the duration of RT100, resulting in a 325 ns pulse. Read data is valid at the sense amplifier inputs from 200 to 275 ns after READ EARLY L goes active. RT175 H is gated with RT50 H to produce the sense amplifier strobes STROBE 0 and 1 H. The trailing edge of RT50 H occurs 100 ns after the leading edge of RT175 H, negating the strobes. During strobe time, the sense amplifier data bits set the appropriate flip-flops that comprise the memory data register, and store the memory read data.

The bus master device initiates the data transfer portion of the DATI transaction by asserting BDIN L. The Reply Enable flip-flop is set on the trailing edge of RT100 H 375 ns after BSYNC L. If RDIN H (BDIN L inverted) is received earlier than 375 ns after BSYNC L, the Reply flip-flop input gates wait 375 ns to produce an active RPLY SET L signal (Figure 4-17), which direct-sets the Reply flip-flop and produces the active FRPLY H and BRPLY L signals. FRPLY L is gated with RDIN L and inverted, producing the DATA OUT L signal which gates memory data register bits onto the BDAL bus. If RDIN H is received later than 375 ns after BSYNC L, the Reply flip-flop sets on the leading edge of RDIN H. The trailing edge of RT150 H (T425 L) is gated with RPLY SET L, producing a write initiate pulse which clocks the high FBUSY H signal into the Write flip-flop, initiating the restore portion of the memory cycle.

Restore timing is produced by the write time generator in a manner similar to that described for read time generation. At WT00 H time, TINH H and TINH L (475 ns pulses) are produced for the inhibit drivers. TINH H also inhibits the Reply Clear gate, and the Reply flip-flop remains set for the remainder of the memory cycle. WEARLY L and STK CHG H go active on the leading edge of WT175 H and remain active for 350 ns. Similarly, WLATE L goes active on the leading edge of WT175 H and remains active for 325 ns. At WT250 H time, WCLR L is produced, clearing the Reply Enable and Write flip-flops; thus, write time generator outputs are 250 ns pulses. Memory data is restored (written) during the time that TINH H, WEARLY L, and WLATE L are active. The memory cycle terminates when both SYNC L and FRPLY L go to their passive states. The Busy Clear gate detects this condition, producing a low pulse which clears the Busy flip-flop, and the memory cycle ends.



CP-1785

Figure 4-16 MMV11-A Timing and Control Circuits

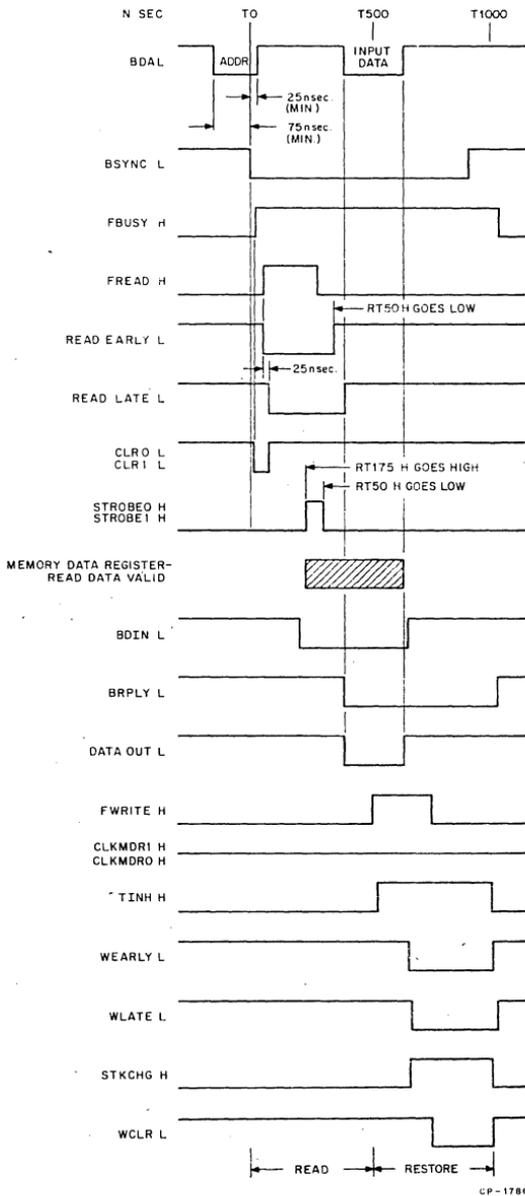
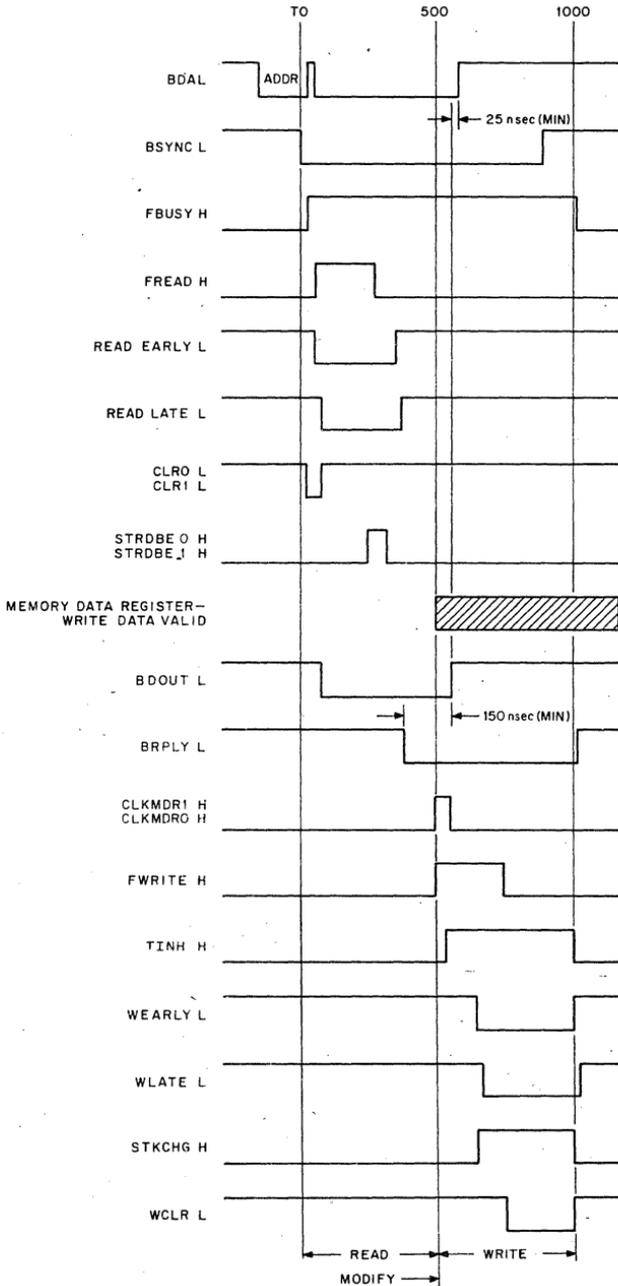


Figure 4-17 Read-Restore Memory Cycle Timing



CP-1787

Figure 4-18 Read-Modify-Write Memory Cycle Timing  
4-28

The DATO cycle is similar to the DATI cycle except that during the addressing portion of the bus cycle, the bus master device asserts BWTBT L. RWBT H goes high, and the leading edge of SYNC H clocks the byte flip-flop to the set state. The active FWBT L signal is only used when in the write portion of the DATIO cycle, as described later. However, during a DATO bus transaction, RDIN H is not received; instead, RDOUT H is received, enabling the REPLY SET L gates, as shown in Figure 4-18. RDOUT enables one input to the WRITE TIME L gate. At the same time that the Write flip-flop clocks to the set state, WRITE TIME L goes low, enabling CLK MDR0 and 1 H gates. Since a DATO bus cycle is in progress, BWTBT L remains passive during the data transfer portion of the bus cycle. Hence, RWBT H is low, WRITE WORD H is high, and the two byte select OR gates apply low signals to the remaining CLK MDR gates. CLK MDR 0 and 1 H then clock the BDAL bus data into the memory data register; the previously read data is lost. The write portion of the cycle continues as described for the restore portion of the DATI operation.

When executing a DATOB bus transaction, BWTBT L and RWBT H remain active for the duration of the bus cycle. Hence, the WRITE WORD H signal remains passive. The Byte Select flip-flop that stores byte addressing bit RAO H during addressing time enables generation of only one CLK MDR H signal. When RAO H is low, FAO L goes high and CLK MDR 0 H clocks low byte data bits from only BDAL0-7 L into the memory data register. Register bits 8-15 remain unchanged. Similarly, when RAO H is high, FAO H goes high and CLK MDR 1 H clocks high byte data bits from only BDAL8-15 L into the memory data register. Register data bits 0-7 remain unchanged. The write portion of the memory cycle then continues as previously described.

When executing a DATIO bus cycle, two complete memory cycles are executed. They include a DATI and a DATO or DATOB cycle as previously described. However, when executing a DATIO bus transaction, BSYNC L remains active for the duration of the transaction. Hence, SYNC H, which generates FBUSY L during the read-restore portion of the cycle, cannot initiate the second read-modify-write memory cycle. Instead, FWBT L, stored during the addressing portion of the cycle, enables a read initiate pulse on the leading edge of RDOUT H. The Read flip-flop goes to the set state and operation continues as described for DATO or DATOB bus transactions.

**4.3.2.5 DC Protection and Vcc Switch—**DC protection and Vcc switch circuits are shown in Figure 4-19. The dc protection circuit is activated

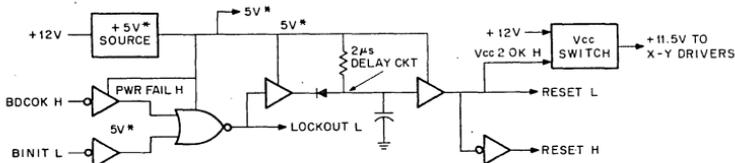


Figure 4-19 DC Protection and Vcc Switch Circuits

during power-fail or bus initialize conditions. BDCOK H and BINIT L are inverted and ORed to produce LOCKOUT L. Normally, this signal is passive (high), enabling bank addressing and resulting in an active DSEL H signal when the memory is addressed. However, if BDCOK H goes low (power fail) or BINIT L is asserted low, LOCKOUT L immediately inhibits the bank addressing function.

The reset signals are also generated by this circuit. RESET L goes active (low) whenever LOCKOUT L is active. A 2  $\mu$ s delay circuit enables the memory to complete its present cycle before RESET. RESET L is also inverted to produce RESET H; both signals are used to clear (initialize) memory timing control circuits.

To produce a 5 V\* source for reset circuits and bus receivers BSYNC L, BDIN L, BDOU L, BWTBT L, and BREF L, +12 V power is regulated. Thus, if +12 V is removed, all MMV11 memory operations are disabled. However, if +5 V is removed and the +12 V remains, the 5 V\* allows memory protect logic to remain functional.

RESET L is also applied to the VCC2OK H input to the Vcc switch circuit. This signal is high only when both +5 V and +12 V power sources are normal. The Vcc switch comprises a transistor (Vcc switch), which is turned on when power is normal to produce +11.5 V power for X-Y driver circuits.

**4.3.2.6 DC-DC Inverter**—The dc-dc inverter circuit is shown in Figure 4-20. It is comprised of an inverter oscillator using a saturable transformer, a negative rectifier, and a filter. A 3-terminal regulator chip produces the regulated -5 V for sense amplifier operation.

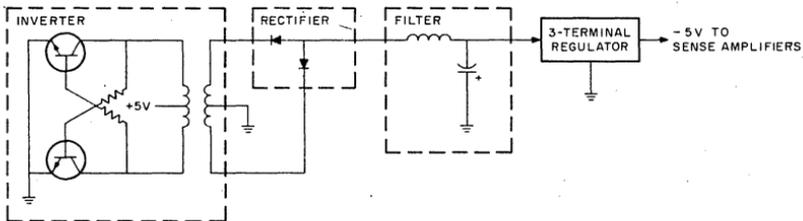


Figure 4-20 DC-DC Inverter Circuit

CP-1789

## 4.4 MRV11-AA 4K BY 16-BIT READ-ONLY MEMORY

### 4.4.1 General

The MRV11-AA is a basic read-only memory module on which the user can install programmable read-only memory (PROM) or masked read-only memory (ROM) chips. All PROM/ROM chip sockets and addressing and control circuits are contained on a single 8.5 by 5 inch module.

The MRV11-AA features:

- 4096 by 16-bit capacity using 512 by 4-bit chips or 2048 by 16-bit capacity using 256 by 4-bit chips.
- Compatibility with chips available from multiple sources.

- Jumpers that allow the user to select the 4K memory address space which the MRV11-AA will respond, chip type, and upper or lower 2K segment (when 256 by 4-bit chips are used.)

#### 4.4.2 Functional Description

**4.4.2.1 General**—Major functions contained on the MRV11-AA module are shown in Figure 4-21. ROM data stored on the module can be addressed and read by the LSI-11 processor or other DMA devices by executing a DAT1 bus cycle. Data/address lines BDAL0-15 L and three bus interface control signals (BSYNC L, BDIN L, and BRPLY L) comprise all interface signals required for accessing the read-only memory. BREF L inhibits BRPLY L and BDAL bus drivers during memory refresh operations.

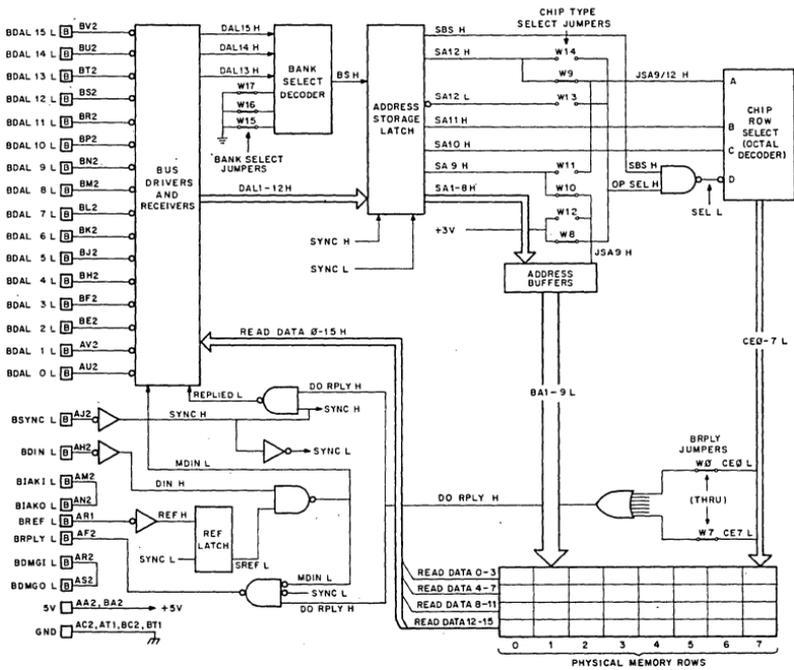


Figure 4-21 MRV11-AA Logic Block Diagram

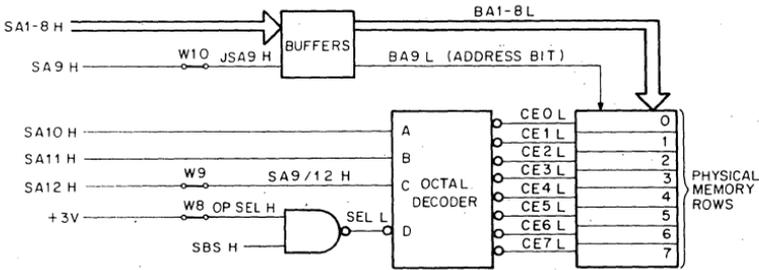
**4.4.2.2 Addressing**—A master device can address any 16-bit word in the 4K module by placing appropriate address bits on BDAL1-15 L during the addressing portion of the DAT1 cycle. BDAL0 is not used on the MRV11-AA since this address bit functions only as a byte pointer during DATOB and the write portion of DATIOB bus cycles. Bus receivers route DAL13-15 H to the bank select decoder and DAL1-12 H to the address storage latch. Bank selection occurs when the 4K address encoded on

DAL13-15 H is equal to the user-configured value selected by jumpers W17-W15. The resulting bank select (BS H) and address bits DAL13-15 H are then stored in the address storage latch on the leading edge of BSYNC L. Stored address bits SA1-8 H are buffered to produce BA1-9 L which are applied to all ROM/PROM chips on the module.

When 512 by 4-bit chips are used, SA9 H is routed via jumper W10 to a buffer, producing the inverted BA9 L address bit for all chips (pin 14). However, when 256 by 4-bit chips are used, W10 is removed and W12 is connected, forcing a low (chip enable) signal to become applied to all chips (pin 14); note that 256 by 4-bit chips do not receive address bit 9.

Memory chips sockets are arranged in eight physical rows of four sockets each. The memory is expanded by installing all four chips in each desired row. Four chips provide the full 16-bit word storage for LSI-11 instructions and data. Only one row is enabled by a chip enable (CE) signal, produced by chip row select logic and chip type jumpers.

When 512 by 4-bit chips are used, jumpers W8, W9, and W10 are installed. The chip row select octal decoder receives stored address bits SA10 H, SA11 H, and SA12 H on its A, B, and C inputs, respectively, as shown in Figure 4-22. Bank Select Stored (SBS H) is gated to produce a low SEL L enable signal, which is applied to the D input of the decoder. (The decoder is actually a decimal decoder; whenever a high signal is applied to its D input, outputs 0-7 are inhibited.) One decoder output goes low, enabling the appropriate physical row addressed by bits SA10-12 L.



11-3159

Figure 4-22 512 by 4-Bit Chip-Jumper Configuration

When 256 by 4-bit chips are used, jumpers W8, W9, and W10 are removed and jumpers W11, W12, and either W13 or W14 are installed, as shown in Figure 4-23. SA10 and SA11 are applied to octal decoder A and B inputs, respectively. Bit SA9, which is not used to directly address the 256 by 4-bit chips is then applied to input C of the octal decoder. SA12 H and SA12 L are available for jumper selection of the desired 2K segment within the 4K bank. W13, when installed, selects the lower 2K; W14 selects the upper 2K. When the selected segment is addressed, OP SEL goes high. This signal is gated with SBS H to produce the low (active) octal decoder enable signal.

Caution must be used when assigning memory to bank 7 to avoid conflicts with preassigned device addresses. This 28-32K address space is normally used for peripheral device addresses. Certain DIGITAL-supplied system programs and operating systems determine the presence or absence of some of these devices by accessing the assigned locations; if a response is obtained (i.e., no bus time-out occurs), the program assumes that the device is present. Thus, having a memory respond to any of these preassigned locations will give the erroneous indication that the corresponding device is installed in the system.

**4.4.2.3 Data Read Operation**—Once the ROM/PROM chip sockets are addressed, the data can be read by the bus master device. Data is available within 120 ns after BSYNC L is received. One active CE0-7 L signal produces the active DO RPLY H signal, which enables reply and BDAL bus driver gating. Active DO RPLY H and SYNC H signals are gated, producing the REPLIED L signal, which enables one of the two bus driver enable inputs. The remaining enable input is MDIN L. The bus master device asserts BDIN L to request the data. DIN H is ANDed with the passive (high) SREF L signal, producing MDIN L, and read data is enabled onto BDALO-15 L. Active MDIN L, SYNC L, and DO RPLY H signals also enable the BRPLY L bus driver, producing the required response to BDIN L.

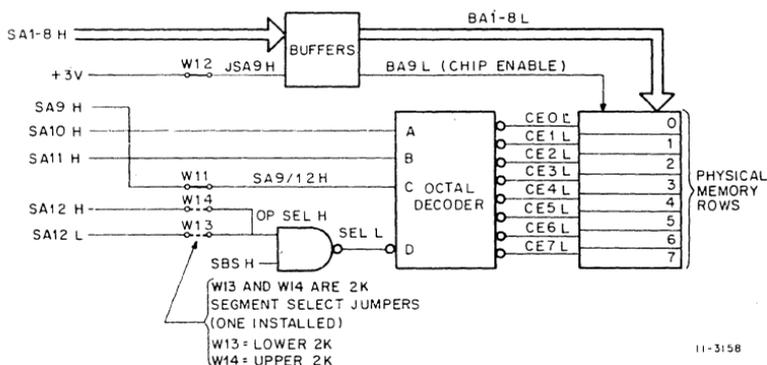


Figure 4-23 256 by 4-Bit Chip-Jumper Configuration

When the system is in a memory refresh operation, the MRV11-A must not respond to the BSYNC/BDIN refresh bus transactions. BREF L is asserted during the addressing portion of the bus cycle and the refresh latch stores REF H on the leading edge of SYNC L. SREF L goes low and inhibits the MDIN L signal. Hence, BDAL and BRPLY L bus drivers are not enabled.

## 4.5 MSV11-B 4K BY 16-BIT SEMICONDUCTOR READ/WRITE MEMORY

### 4.5.1 General

The MSV11-B is a 4K by 16-bit dynamic MOS read/write memory module which can be used for storage of user programs and data. The storage capacity is 4096, 16-bit words. Memory address selection is user-configured by installing or removing jumpers contained on the module.

Memory refresh is directly controlled by LSI-11 bus signals. Refresh operations can be automatically controlled by the LSI-11 microcomputer module once every 1.6 ms (approximately) or performed by another device. The MSV11-B is LSI-11 bus compatible and capable of either programmed I/O data transfers with the processor or DMA transfers with other LSI-11 bus devices.

The MSV11-B features:

- 4096 by 16-bit word.
- Fast access time—550 ns maximum.
- Lower power—12.7 W for the module, maximum.
- Dynamic MOS memory chips—Refresh is automatically controlled by the processor or by a DMA device.
- User-configured 4K addresses—Three jumpers allow user address configuration.

#### 4.5.2 Functional Description

Major functions contained on the MSV11-B module are shown in Figure 4-24. Memory data can be stored (written) or read by the LSI-11 microcomputer, or other bus master devices operating in the DMA mode, with appropriate bus cycles: DATO (16-bit word write operation); DATOB (8-bit byte write operation); DATI (16-bit read operation); or DATIOB [16-bit read-modify-write (8 or 16-bit) operation].

Addressing is initiated by a master device (either the LSI-11 processor or a DMA device) by placing the 16-bit address on BDAL0-15 L and asserting BSYNC L, latching the address (and bank select information) in the address register. Address bits are routed from the BDAL bus receivers onto the module's DAL0-15 H bus to the 13-bit address and bank select register input. Address bits BDAL13-15 L are decoded by the bank address decoder. Decoded output signal BS H will go active (high) only when the jumper-selected bank address is decoded. The active BS H signal is stored along with the 13-bit memory address for the duration of the operation.

The memory array comprises sixteen 16-pin 4K by 1-bit memory chips which require the address multiplexer to address the array with two 16-bit bytes. Address multiplexer control logic responds to the active SYNC H and stored active bank select (LBS L) signal by immediately generating an active Row Address Strobe (RAS). This signal remains active for the duration of the active SYNC H signal. Address multiplexer control AMX L is initially passive (high), multiplexing the stored row address bits (LDAL7-12 H) through the 12:6-bit address multiplexer and into all memory chips. After approximately 150 ns, address multiplexer control logic generates an active column address strobe (CAS) and an active AMX L signal. Multiplexer column address bits (LDAL1-6 H) are then strobed into all memory chips. This completes the chip addressing portion of the memory operation.

When in a memory read operation, the bus master device asserts BDIN L. The data from the accessed memory location is present on the D0-15 H bus and at bus driver inputs. Reply logic responds to BDIN L by generating an active DRIVE L signal which gates the memory read data



mains asserted if a DATOB (byte) write cycle is to be performed. Word/byte select logic responds to the DATO cycle by asserting both W0 L and W1 L for the duration of the cycle, enabling DAL0-15 H bits to be written into the addressed location in all memory chips. However, in a DATOB cycle with A0 H low (even byte), only W0 L goes active, enabling the writing of DAL0-7 H into the addressed location in the appropriate eight memory chips. Similarly, if A0 H is high (odd byte), only W1 L goes active, enabling only DAL8-15 H bits to be written into the addressed location in the appropriate eight memory chips. The reply logic also responds to the active BDOUT L signal by asserting BRPLY L, indicating that the data has been written, completing the data transfer.

The memory chips in the MSV11-B require a refresh operation once every 1.6 ms. This operation is entirely under the control of either processor microcode or a DMA device, as selected by the user. The address multiplex control logic responds to BREF L, generated by the refresh-controlling device, by simulating a "bank selected" operation. (All system memory banks are simultaneously selected during refresh.) Refresh is then accomplished by a device by executing 64 successive BSYNC L/BDIN L operations while incrementing BDAL1-6 by one on each bus transaction. Refresh is simply a series of forced memory read operations where only the row addresses are significant. Each of the 64 rows in all dynamic MOS memory chips in an LSI-11 system are simultaneously refreshed in this manner. The REF H signal inhibits all BDAL bus drivers for the duration of the refresh operation.

A dc-to-dc inverter circuit is included on the module for negative voltage generation. Output voltages include  $-9$  V for the MOS memory chips and  $-5$  V for linear devices in the address multiplex control logic. Hence, only  $+12$  V and  $+5$  V power inputs are required for module operation. The BDCOK H signal starts dc-to-dc inverter oscillation when bus power is applied.

## **4.6 DLV11 SERIAL LINE UNIT**

### **4.6.1 General**

The DLV11 is the basic interface module used for connecting asynchronous serial line devices to the LSI-11 bus. All circuits are contained on a single 8.5 by 5 inch module.

The DLV11 features:

- Either an optically isolated 20 mA current loop or an EIA interface selected by using the appropriate interface cable option.
- Selectable crystal-controlled baud rates: 50, 75, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, and an externally supplied rate.
- Jumper-selectable stop bit and data bit formats.
- LSI-11 bus interface and control logic for interrupt processing and vector generation.
- Interrupt priority determined by electrical position along the LSI-11 bus.

- Control/status register (CSR) and data registers compatible with PDP-11 software routines. CSRs and data buffer registers directly accessed via processor instructions.
- Plug, signal, and program compatible with PDP-11 DL11A, B series.

#### 4.6.2 Functional Description

**4.6.2.1 General**—Major functions contained on the DLV11 module are shown on Figure 4-25. Communications between the LSI-11 microcomputer and the DLV11 are executed via programmed I/O operations or interrupt-driven routines, as described in Chapter 3.

**4.6.2.2 UAR/T Operation**—The main function on the DLV11 module is the Universal Asynchronous Receiver/Transmitter (UAR/T) chip. This is a 40-pin LSI chip that is capable of parallel I/O with the computer bus and asynchronous serial I/O with an external device. Jumpers which allow the user to select parity functions, number of stop bits, and number of data bits are described in Paragraph 5.6. Both transmit and receive functions are totally asynchronous in operation. The transmit clock is always driven by the baud rate generator's CLK L signal. CLK L is applied to one MSPAREB backplane pin (BK1), where it is connected to MSPAREB pin BL1; this is the receive function UAR/T clock input (RCLK L) signal.

When a user application requires split transmit and receive baud rates, the MSPARE jumper can be broken from pins BK1 and BL1 and an external receive baud rate signal can be applied to BL1 (the drive frequency should be 16 times the desired baud rate).

**4.6.2.3 Baud Rate Generator**—The baud rate generator produces the desired UAR/T clock and a fixed 2.4576 MHz clock for the -12 V inverter circuit. A crystal-controlled oscillator produces the basic 2.4576 MHz frequency for the baud rate generator. A single baud rate generator chip divides this frequency to produce the available baud rates. Jumpers, which are described in Paragraph 5.6, select the desired baud rate for the CLK L output signal.

**4.6.2.4 Bus Drivers and Receivers**—Bus drivers and receivers interface directly with the LSI-11 bus. Line receivers produce RDAB0-12 H signals in response to BDAL0-12 L bus signals. When an input data or vector transfer is desired, function decoding and control logic generates an active INPUT ENABLE signal, which enables the bus drivers. When a data input operation is selected, the UAR/T receiver data buffer contents (RDO-7 H) are routed through the data selector (DDAB0-7H) to the BDAL bus. When responding to an interrupt acknowledge signal, interface control logic generates VEC L, which selects the vector address produced by jumpers W6-W10 (Paragraph 5.6). In addition, DAL0, 6, 7, and 15 are driven by CSR selection and gating circuits when a data input transfer from either the receiver (RCSR) or transmitter (XCSF) control/status registers is performed.

**4.6.2.5 Address Decoding**—Address decoding logic responds to the address present on the bus when BSYNC L is asserted. The DLV11 device address is contained on RDAB3-12 H, along with address bits

RDAB0, 1, and 2 H, which are decoded by function decoding logic. Address bits are not required for bank selection since all devices, such as any DLV11, reside in the upper 4K bank (addresses ranging from 28-32K). The processor generates an active BBS7L signal, indicating an I/O device addressing operation. Address selection jumpers A3-A12 allow the user to configure address bits 3-12, as described in Paragraph 5.6. When the DLV11 is addressed, device selection is indicated by an active ME signal. This signal remains active throughout the entire I/O cycle (while BSYNC L remains active), enabling function decoding.

**4.6.2.6 Function Decoding and Control**—Function decoding and control logic decodes DLV11 internal gating functions based upon address selection, address bits RDAB0, 1, and 2 H, bus signals BDIN L, BDOU L, and BSYNC L, and the VEC L signal generated by the interface control logic. In addition to generating function select signals, this circuit inverts BSYNC L to produce SYNC H whose leading edge clocks the address decoding logic. A truth table of function select signals is provided in Table 4-2.

**4.6.2.7 Interface Control Logic**—Interface control logic produces the BRPLY L signal in response to I/O operations, contains the interrupt control logic, and receives and distributes the BINIT L initialize signal. This function also contains the Transmit Data Interrupt Enable (TDINTEN H) flip-flop and Receiver Data Interrupt Enable (RDINTEN H) flip-flop; both flip-flops can be read or written by the LSI-11 microcomputer. RDINTEN is set or reset by BDAL6 L; the flip-flop is clocked on the leading edge of SEL0OUT L. Similarly, TDINTEN is set or reset by BDAL6 L; this flip-flop is clocked on the leading edge of SEL4OUT L.

Receiver-generated interrupts occur as a result of the RDINTEN flip-flop being set (interrupts enabled) and an active receiver Data Available (DA H) UAR/T status signal. When this condition occurs, the Receiver Data Interrupt Request flip-flop sets and generates an active BIRQ L signal. The LSI-11 microcomputer responds (if its PS bit 7 is not set) by asserting BDIN L; this enables the device requesting the interrupt to place its vector on the BDAL bus when the interrupt request is acknowledged. The processor then asserts BIAKO L, acknowledging the interrupt request. The interface control logic receives BIAKI L and responds by generating active VEC L and BRPLY L signals, placing its interrupt vector on the LSI-11 bus and clearing the BIRQ L signal. The stored BIAK signal is cleared when the next BIAKI L signal is received and the DLV11 is not requesting an interrupt.

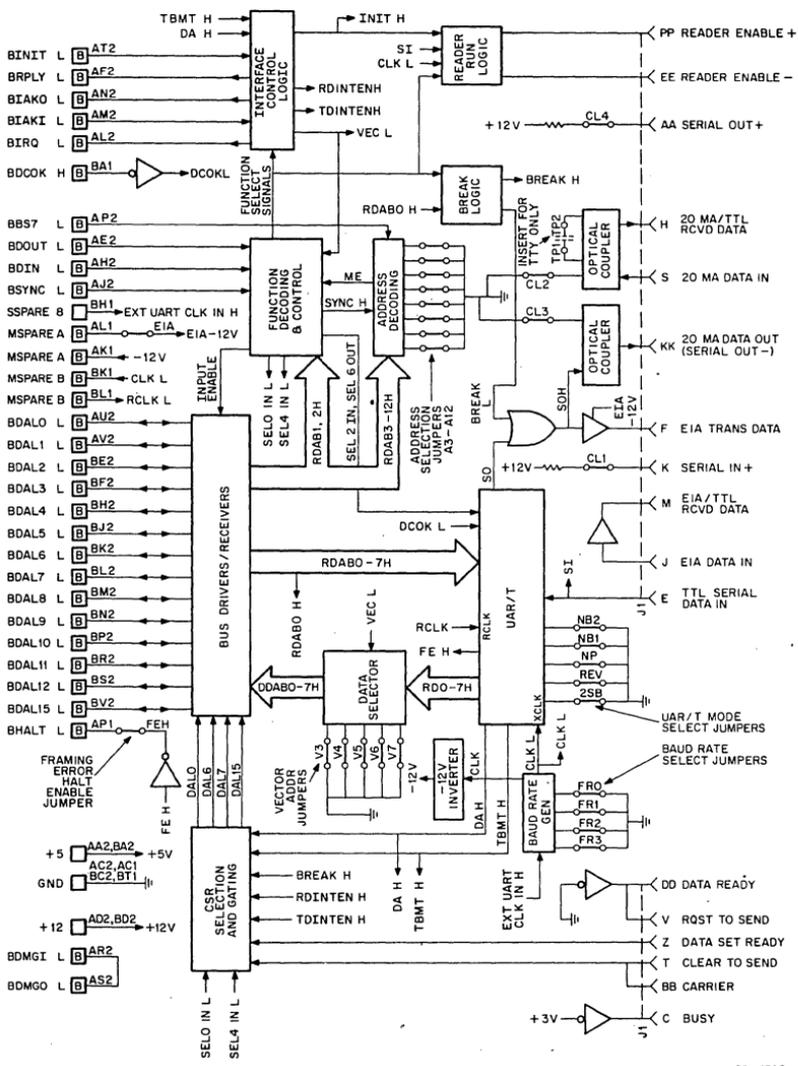


Figure 4-25 DLV11 Logic Block Diagram

Table 4-2 DLV11 Function Decoding

| Address Inputs |    | Control Inputs |         |      | Function Select Signals (low-active) |           |           |            |           |            |            |
|----------------|----|----------------|---------|------|--------------------------------------|-----------|-----------|------------|-----------|------------|------------|
| A1             | A2 | BDIN L         | BDOUT L | ME L | SEL 0IN L                            | SEL 2IN L | SEL 4IN L | SEL 0OUT L | SEL 6IN L | SEL 4OUT L | SEL 6OUT L |
| X              | X  | X              | X       | H    | H                                    | H         | H         | H          | H         | H          | H          |
| L              | L  | L              | X       | L    | L                                    | H         | H         | H          | H         | H          | H          |
| H              | L  | L              | X       | L    | H                                    | L         | H         | H          | H         | H          | H          |
| L              | H  | L              | X       | L    | H                                    | H         | L         | H          | H         | H          | H          |
| L              | L  | H              | L       | L    | H                                    | H         | H         | L          | H         | H          | H          |
| L              | H  | H              | L       | L    | H                                    | H         | H         | H          | H         | L          | H          |
| H              | H  | H              | L       | L    | H                                    | H         | H         | H          | H         | H          | L          |
| H              | H  | L              | H       | L    | H                                    | H         | H         | H          | L         | H          | H          |

Transmitter-generated interrupts occur in a manner similar to the receiver-generated interrupts. However, they occur as a result of the TDINTEN flip-flop being set (interrupts enabled) and when the Transmitter Buffer Empty (TBMT H) UAR/T signal is active (high). Note that if the transmitter and receiver functions request interrupts simultaneously, the receiver interrupt vector will be transmitted on the first interrupt cycle, and the transmitter interrupt vector will be transmitted on a subsequent (separate) interrupt sequence. If BIAKI L is received and the DLV11 is not requesting an interrupt, it passes BIAKO L to the next device in the priority chain.

The interface control logic also generates the DLV11's BRPLY L signal. It generates this signal when any function select signal is asserted or VEC L is generated.

The system initialize signal (BINIT L) is generated by the processor to reset all peripheral device registers. Interface control logic responds by clearing all control flip-flops, including the Interrupt Request, Interrupt Acknowledge, and Break flip-flop. The UAR/T's RBUF and XBUF data registers are not cleared by BINIT L; however, the initialize signal does clear the DAH signal and set the TBMT H signal.

**4.6.2.8 CSR Selection and Gating**—CSR selection and gating logic enables the LSI-11 microcomputer to read receiver and transmitter control/status bits. Functions are summarized below.

*Read RCSR (SELOIN L asserted)*

CARRIER or CLR TO SEND or DATA SET READY → BDAL15

DAH → BDAL7

ROINTEN H → BOAL6

*Read XCSR (SEL4IN L asserted)*

TBMTH → BDAL7

TDINTEN H → BDAL6

BREAK H → BDAL0

**4.6.2.9 Break Logic**—Break logic comprises the Break status flip-flop. It is set or cleared by the LSI-11 microcomputer by BDAL0 while executing a bus output cycle with the XCSR. Thus, the duration of the break signal is program controlled. The Break flip-flop is clocked on the leading edge of the SEL4OUT H signal. When set, the serial output line is continuously negated (space) or open circuit. The status of the Break flip-flop can be read in XCSR bit 0.

**4.6.2.10 Reader Run Logic**—The reader run logic enables DLV11 generation of a READER RUN pulse for 20 mA current loop teletypewriter devices. It is enabled by loading RCSR bit 0; the LSI-11 microcomputer asserts BDAL0 and causes generation of the SELOUT H signal (load RCSR). READER RUN is asserted and remains active until the received serial data has been in a mark condition for the duration of eight consecutive clock pulses. The start bit of the serial input (SI) from the low-speed reader initiates a 4-bit binary counter. When eight CLK L pulses have been counted (equivalent to one-half of the start bit), READER RUN is negated.

**4.6.2.11 EIA Interface Circuits**—An EIA interface is provided by EIA drivers and receivers. EIA signal drivers are provided for EIA TRANS DATA, RQST TO SEND, DATA TERMINAL READY (always an active high, and BUSY (always an active low. Jumper EIA applies -12 V to the EIA driver chip when the DLV11 is used with EIA-compatible devices. EIA signal receivers are provided for EIA DATA IN, CARRIER or CLEAR TO SEND, and DATA SET READY. The optional BC05C modem cable connects the output signal of the EIA DATA IN driver (EIA/TTL RCVD DATA) to the TTL SERIAL DATA IN input to the UAR/T.

**4.6.2.12 20 mA Loop Current Interface**—The 20 mA loop current interface is provided by optical isolation. An active 20 mA current loop is provided when jumpers CL1 through CL4 are installed. If the jumpers are removed, 20 mA passive current loop operation is selected. The optional BC05M cable assembly connects the 20 mA/TTL RCVD DATA optical coupler signal output to the TTL SERIAL DATA IN input of the UAR/T. When the DLV11 is used with a 110 baud teletypewriter device, a 0.005 $\mu$ F, 100 V filter capacitor should be installed between terminals TP1 and TP2.

**4.6.2.13 -12 V Inverter**—The -12 V inverter circuit generates -12 V for use by the UAR/T chip and EIA driver and receiver chips. Input to the circuit is the CLK signal (2.4576 MHz) and +12 V. The output is zener regulated to -12 V.

## **4.7 DRV11 PARALLEL LINE UNIT**

### **4.7.1—General**

The DRV11 is a general-purpose interface unit used for connecting parallel line devices to the LSI-11 bus. All circuits are contained on a single 8.5 by 5 inch module.

The DRV11 features:

- 16 diode-clamped data input lines.
- 16 latched output lines.
- 16-bit word or 8-bit byte programmed data transfers.
- User-assigned device address decoding.
- LSI-11 bus interface and control logic for interrupt processing and vector generation.
- Interrupt priority determined by electrical position along the LSI-11 bus.
- Control/status registers (CSR) and data registers that are compatible with PDP-11 software routines. Plug, signal, and program compatible with DR11-C.
- Four control lines to the peripheral device for NEW DATA READY, DATA TRANSMITTED, RQSTA, and RQSTB.
- Logic compatible with TTL or DTL devices.
- Program-controlled data transfer rate of 90K words per second (maximum).
- Maximum drive capability of 25 ft of cable.

## 4.7.2 Functional Description

**4.7.2.1 General**—Major functions contained on the DRV11 module are shown in Figure 4-26. Communications between the LSI-11 microcomputer and the DRV11 are executed via programmed I/O operations or interrupt-driven routines, as described in Chapter 3.

The DRV11 is capable of storing one 16-bit output word or two 8-bit output bytes in DROUTBUF. The stored data (OUT0-15 H) is routed to the user's device via an optional I/O cable connected to J1. Any programmed operation that loads either a byte or a word in DROUTBUF causes a NEW DATA READY H signal to be generated, informing the user's device of the operation.

Input data (DRINBUF) is gated onto the BDAL bus during a DATI bus cycle. All 16 bits are placed on the bus simultaneously; however, when the processor is involved in 8-bit byte operation, it uses only the high or low byte. When the data is taken by the processor, a DATA TRANS H pulse is sent to the user's device to inform the device of the data transfer.

**4.7.2.2 Addressing**—When addressing a peripheral device interface such as the DRV11, the processor places an address on BDAL0-15 L, which is received and distributed as BRD0-15 H in the DRV11. The address is in the upper 4K (28-32K) address space. On the leading edge of BSYNC L, the address decoder decodes the address selected by jumpers A3-A12 and sets the Device Selected flip-flop (not shown); the active flip-flop output is the ME signal, which enables function selection and I/O control logic operation. At the same time, function selection logic stores address bits BRD0-2.

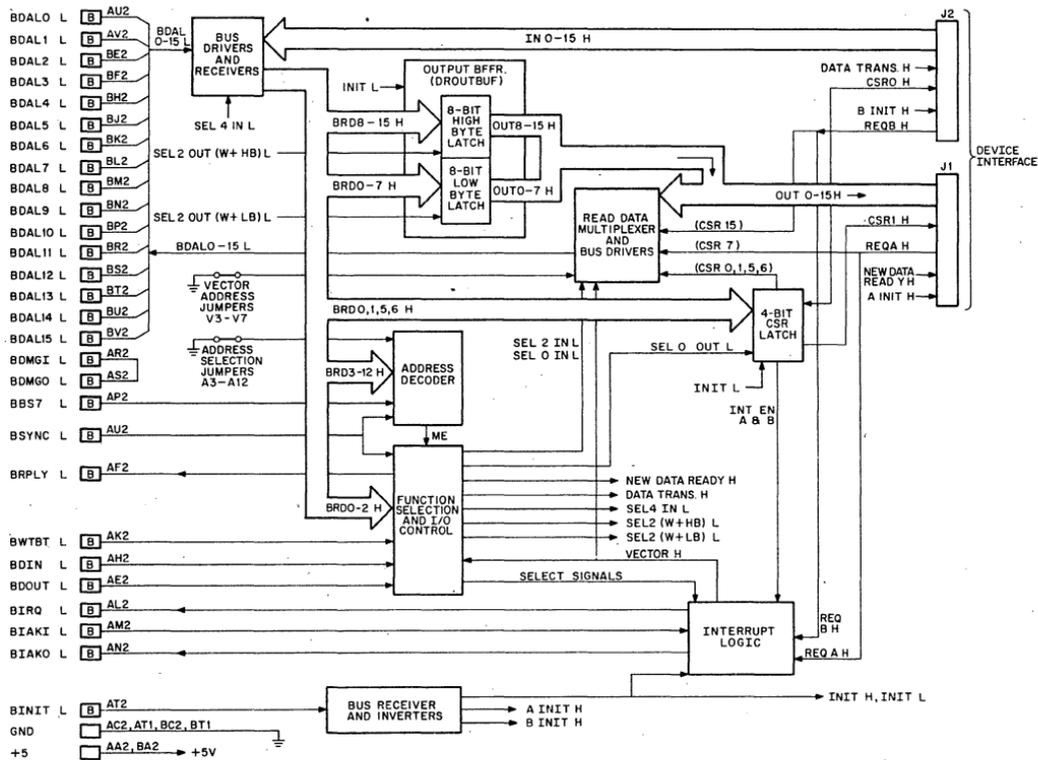


Figure 4-26 DRV11 Logic Block Diagram

**Table 4-3 DRV11 Device Function Decoding**

| <b>Programmed Operation</b> | <b>Stored Device Addr. Bits 0-2</b> | <b>BWTBT L During Data Transfer</b> | <b>BDIN L</b> | <b>BDOUT L</b> | <b>Bus Cycle Type</b> | <b>Select Signals</b>  |
|-----------------------------|-------------------------------------|-------------------------------------|---------------|----------------|-----------------------|--|
| Write DRCSR                 | 0                                   | 0                                   | H             | L              | DATO                  | SEL0OUT L  |
|                             | 0                                   | 1                                   | H             | L              | DATOB                 |  |
| Read DRCSR                  | 0                                   | 0                                   | L             | H              | DATI or DATIO         | SEL0IN L   |
| Write DROUTBUF Word         | 2                                   | 0                                   | H             | L              | DATO                  | SEL2OUT (W + HB) L, SEL2OUT (W + LB) L, and NEW DATA READY H |
| Low Byte                    | 2                                   | 1                                   | H             | L              | DATOB                 | SEL2OUT (W + LB) L and NEW DATA READY H                      |
| High Byte                   | 3                                   | 1                                   | H             | L              | DATOB                 | SEL2OUT (W + HB) L and NEW DATA READY H                      |
| Read DROUTBUF               | 2                                   | 0                                   | L             | H              | DATI or DATIO         | SEL2IN L   |
| Read DRINBUF                | 4                                   | 0                                   | L             | H              | DATI                  | SEL4IN L and DATA TRANS H                                    |

#### NOTE

When addressed, the DRV11 always responds to either BDIN L or BDOUT L by asserting BRPLY L [L = assertion].

**4.7.2.3 Function Selection**—Function selection and I/O control logic monitors the ME signal and bus signals BDIN L, BDOUT L, and BWTBT L. It responds by generating appropriate Select signals which control internal data gating, NEW DATA READY H or DATA TRANS H output signals for the user's device, and the BRPLY L bus signal which informs the processor that the DRV11 has responded to the programmed I/O operation. Since the DRV11 appears to the processor as three addressable registers (DRCSR, DROUTBUF, and DRINBUF) that can be involved in either word or byte transfers, the three low-order address bits stored during the addressing portion of the bus cycle are used for function selection. The select signals relative to I/O bus control signals and address bits 0-2 are listed in Table 4-3.

NEW DATA READY H is active for the duration of BDOUT L when in a DROUTBUF write operation. This signal is normally active for 300 ns. However, by adding an optional capacitor in the BRPLY L portion of the circuit, the leading edge of BRPLY is delayed, effectively increasing the duration of the NEW DATA READY H pulse to 1200 ns (maximum); adding the capacitor also increases the DATA TRANS H pulse width in exactly the same manner.

DATA TRANS H is active for the duration of BDIN L when in a DRINBUF read operation. This signal is normally active for 300 ns. The time, however, can be extended by adding the optional capacitor to the BRPLY L portion of the circuit as previously described.

**4.7.2.4 Read Data Multiplexer**—The read data multiplexer selects the proper data and places it on the BDAL bus when the processor inputs DRCSR, DROUTBUF or interrupt vectors; DRINBUF contents are gated onto the bus separately. The select signals (previously described) and VECTOR H, produced by the interrupt logic, control read data selection.

**4.7.2.5 DRCSR Functions**—The control/status register (DRCSR) is comprised of separate functions. Four of the six significant DRCSR bits can be involved in either write or read operations. The remaining two bits, 7 and 15, are read-only bits that are controlled by the external device via the REQ A H and REQ B H signals, respectively. The four read/write bits are stored in the 4-bit CSR latch. They represent CSR0 and CSR1 (DRCSR bits 0 and 1, respectively), which can be used to simulate interrupt requests when used with an optional maintenance cable. INT ENB A and INT ENB B (bits 6 and 5, respectively) enable interrupt logic operation. Note that CSR0 and CSR1 are available to the user's device for any user application.

**4.7.2.6 DRINBUF Input Data Transfer**—DRINBUF is an addressable 16-bit read-only register that receives data from the user's device for transmission to the LSI-11 bus. Data to be read is provided by the user's device on the IN0-15 H signal lines. Since the input buffer consists of gating logic rather than a flip-flop register, the user's device must hold

the data on the lines until the data input transaction has been completed.

The input data is read during a DAT1 sequence while bus drivers are enabled by the SEL4IN L signal. The DATA TRANSMITTED pulse that is sent to the user's device by the function select logic informs the device of the transaction. Input data can be removed on the trailing edge of this pulse.

**4.7.2.7 DROUTBUF Output Data Transfer**—DROUTBUF is comprised of two 8-bit latches, enabling either 16-bit word or 8-bit byte output transfers. Two SEL 2 signals function as clock signals for the latches. When in a DATO bus cycle, both signals clock data from the internal BRDO-15 H bus into the latches. However, when a DATOB cycle, only one signal clocks data into an 8-bit latch, as determined by address bit 0 previously stored during the addressing portion of the bus cycle.

The NEW DATA READY H pulse generated by the function select logic is sent to the user's device to inform the device of the data transaction. The data can be input to the device on the trailing edge of this pulse.

**4.7.2.8 Interrupts**—The DRV11 contains LSI-11 bus-compatible interrupt logic that allows the user's device to generate interrupt requests. Two independent interrupt request signals (REQ A H and REQ B H) are capable of requesting processor service via separate interrupt vectors. In addition, DRCSR contains two interrupt enable bits (INTEN A and INT EN B) (bits 6 and 5, respectively), which independently enable or disable interrupt requests. REQ A and REQ B status can be read by the processor in DRCSR bits 7 and 15, respectively. Since separate interrupt vectors are provided for each request, one of the requests could be used to imply that device data is ready for input and the remaining request could be used to imply that the device is ready to accept new data.

An interrupt sequence is generated when a DRCSR INT EN bit (A or B) is set and its respective REQ signal is asserted by the device. The processor responds (if its PS bit 7 is not set) by asserting BDIN L; this enables the device requesting the interrupt to place its vector on the BDAL bus when the interrupt request is acknowledged. The processor then asserts BIAKO L, acknowledging the interrupt request. The DRV11 receives BIAKI L and the interrupt logic generates VECTOR H, which gates the jumper-addressed vector information through the read data multiplexer and bus drivers and onto the LSI-11 bus. The processor then proceeds to service the interrupt request as described in Chapter 3.

**4.7.2.9 Maintenance Mode**—The maintenance mode allows the user to check DRV11 operation by installing an optional BC08R cable between connectors J1 and J2. This maintenance cable allows the contents of the output buffer DROUTBUF to be read during a DRINBUF DAT1 bus cycle. In addition, interrupts can be simulated by using DRCSR bits CSR0 and CSR1. CSR1 is routed via the cable directly to the REQ B H input and CSR0 is routed to the REQ A H input. By setting or clearing INT EN A, INT EN B, and CSR0 and CSR1 bits in the DRCSR register, a maintenance program can test the interrupt facility.

**4.7.2.10 Initialization**—BINIT L is received by a bus driver, inverted, and distributed to DRV11 logic to initialize the device interface. The buffered initialize signal is available to the user's device via the AINIT H and BINIT H signal lines. DRV11 logic functions cleared by the BINIT signal include DROUTBUF, DRCSR (bits 0, 1, 5, and 6), and interrupt logic.

#### **4.8 DRV11-B DMA INTERFACE**

A detailed description of the DRV11-B option is included in the *DRV11-B General Purpose DMA Interface User's Manual*. Refer to that publication for installation, programming, and theory of operation.

#### **4.9 DRV11-P LSI-11 BUS FOUNDATION MODULE**

A detailed description of the DRV11-P option is included in the *DRV11-P Foundation Module User's Manual*. Refer to that publication for installation, configuring user-designed logic circuits on the module, applicable programming information, and theory of operation of circuits that directly communicate with the LSI-11 bus.

#### **4.10 LSI-11 BUS ACCESSORY OPTIONS**

##### **4.10.1 General**

The options described in this section generally involve variations of the M9400 module. By including selected components and jumpers, the various options are factory produced. Figures 4-27 through 4-31 each include a simple block diagram of the options, and functional position within a system, as appropriate. A list of options and figures is provided.

| <b>Figure</b> | <b>Option</b> |
|---------------|---------------|
| 4-27          | REV11-A       |
| 4-28          | REV11-C       |
| 4-29          | TEV11         |
| 4-30          | BCV1B-XX      |
| 4-31          | BCV1A-XX      |

Detailed descriptions of functions contained within these figures are contained in the following paragraphs.

##### **4.10.2 Terminations**

Two types of terminations are provided: 120 $\Omega$  and 250 $\Omega$ . Each bus signal line termination includes two resistors as shown in Figure 4-32. Termination resistors are generally contained in 16-pin dual-inline packages which are physically identical to I.C. packages. Each package contains 14 terminations. The values used are shown in the figure. Daisy-chained grant signals are terminated and jumpered; BIAKI is jumpered to BIAKO L and BDMGI is connected to BDMGO L via a factory-installed jumper (W1). Note that the 120 $\Omega$  and 250 $\Omega$  termination values are nominal: with the resistor values shown, the actual termination values will be approximately 124 $\Omega$  and 222 $\Omega$ , respectively.

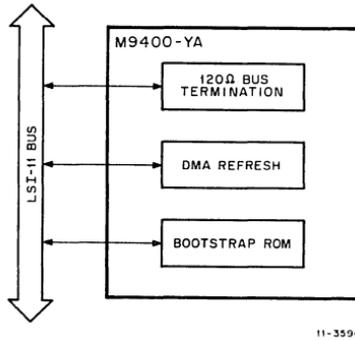


Figure 4-27 REV11-A Functions

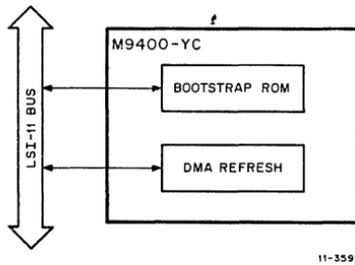


Figure 4-28 REV11-C Functions

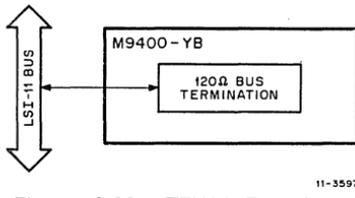


Figure 4-29 TEV11 Functions

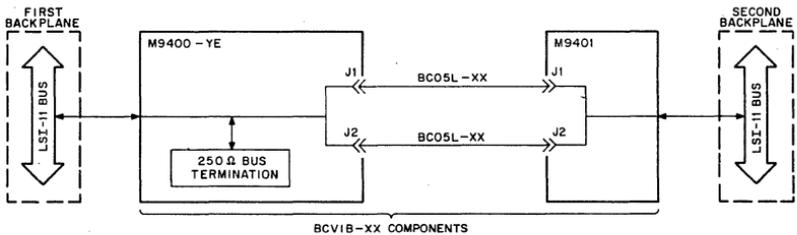


Figure 4-30 BCV1B Functions

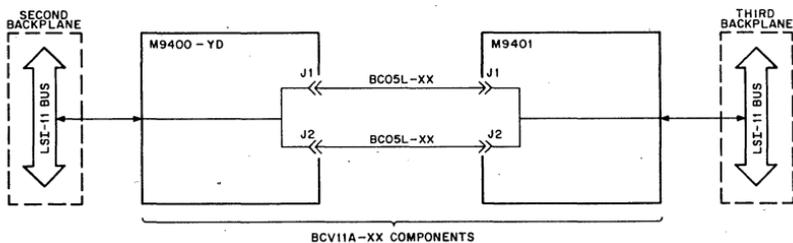


Figure 4-31 BCV1A Functions

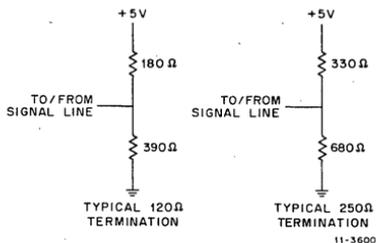


Figure 4-32 G-7 Bus Terminations

### 4.10.3 Bootstrap ROM Loader Logic

**4.10.3.1 General**—Bootstrap ROM loader logic is included in REV11-A and REV11-C options. Logic functions on their respective M9400-YA and M9400-YC modules are identical. Bootstrap programs contained in the REV11-A and REV11-C ROMs are identical. Logic functions are described below.

**4.10.3.2 Addressing**—The module includes a 512 x 16-bit ROM array that is addressed in two 256-word segments. These address segments are reserved for REV11 options and reside in the upper 4K address bank, normally used for peripheral device addresses. The reserved addresses range from 165000-165776 and 173000-173776. A power-up mode, which will cause the processor to access ROM location 173000 upon power up, is jumper selectable on the KD11-F or KD11-J processor module.

Circuits associated with bootstrap ROM logic are shown in Figure 4-33. Wired inputs to an address comparator circuit reserve the ROM addresses. The address comparator responds to any of the reserved addresses during the addressing portion of a bus I/O cycle by generating an active MY BANK H signal. MY BANK H is ANDed with DIN H to produce a BRPLY H signal when the ROM word is read by the processor.

MY BANK H is also inverted and applied to the ROM array chip enable inputs.

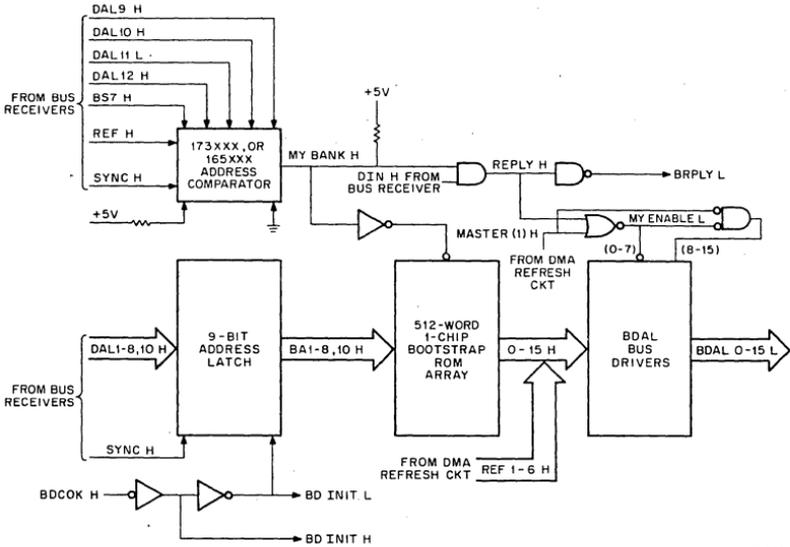


Figure 4-33 Bootstrap ROM Logic

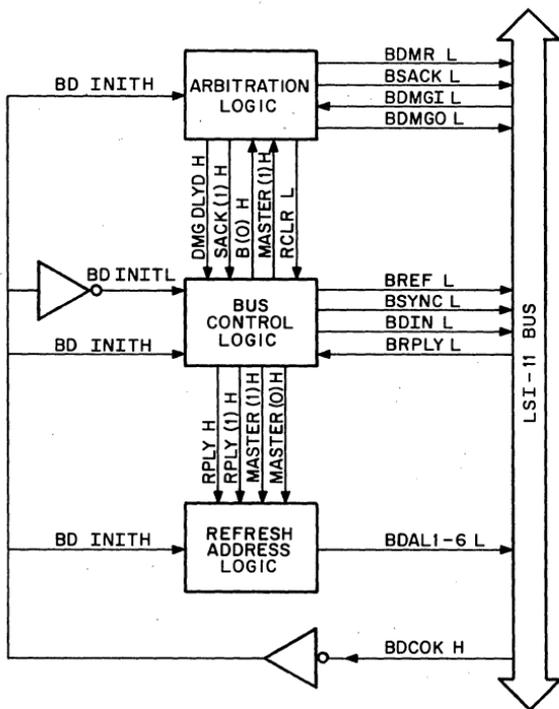
DAL1-8 H and DAL10 H address bits are stored in the 9-bit address latch on the leading-edge of SYNC H. The stored BA1-8 H and BA10 H address bits select the desired word location to be read within the two 256-word address segment detected by the address comparator.

**4.10.3.3 Data Transfer**—After the addressing portion of the bus DAT1 cycle has been completed, the ROM 0-15 H data becomes available on the BDAL bus driver inputs. Note that the ROM array consists of four 512 x 4-bit IC's. Hence, the four 4-bit outputs comprise the 16-bit LSI-11 word. MY ENABLE L strobes the 16-bit word onto the I/O bus in response to the DIN H signal. The processor then receives the data, terminates BDIN L, and the bootstrap ROM logic responds by terminating BRPLY L and inhibiting the BDAL bus drivers.

**4.10.3.4 Initialization**—the bootstrap ROM logic is initialized only when BDCOKH goes false. This condition occurs during a power failure and produces active BD INIT H and BD INIT L signals. These signals clear the 9-bit address latch and circuits contained in the DMA refresh logic. The option does not respond to the LSI-11 bus BINIT L signal.

#### 4.10.4 DMA Refresh Logic

**4.10.4.1 General**—DMA refresh logic consists of the three main functions shown in Figure 4-34. Arbitration logic requests the I/O bus once every 30  $\mu$ s (approx) and completes the required DMA signal sequence



11-3602

Figure 4-34 DMA Refresh Logic

with the processor. When it becomes bus master, it enables the bus control logic to execute a single refresh **BSYNC L**/**BDIN L** bus transaction simultaneously refreshing one row in all dynamic MOS memory chips contained in the system. The refresh address logic places a six-bit memory chip "row" address on the **BDAL 1-6** lines during the addressing portion of the I/O bus cycle. Once the cycle has been completed, the row address is incremented by one for the next refresh cycle, and the I/O bus is released. The actual refresh transaction for one row address takes approximately  $1.2 \mu\text{s}$ . Each dynamic MOS memory chip contains 64 row addresses. Hence, the DMA refresh logic is capable of refreshing all dynamic MOS memory contained in the system within the required 2 msec (maximum) period (ie.,  $64 \text{ rows} \times 30 \mu\text{s}$  between refresh bus cycles = 1.92 ms).

**4.104.2 Arbitration Logic**—Arbitration logic is shown in Figure 4-35; timing is shown in Figure 4-36. The DMA refresh sequence is initiated once every  $30 \mu\text{s}$  by a clock oscillator. **GRAB BUS H** is the clock output signal. It is inverted and gated with a ground (enable) signal, supplied via Refresh Disable jumper **W2**, to produce the **REQ H** signal. **REQ H**



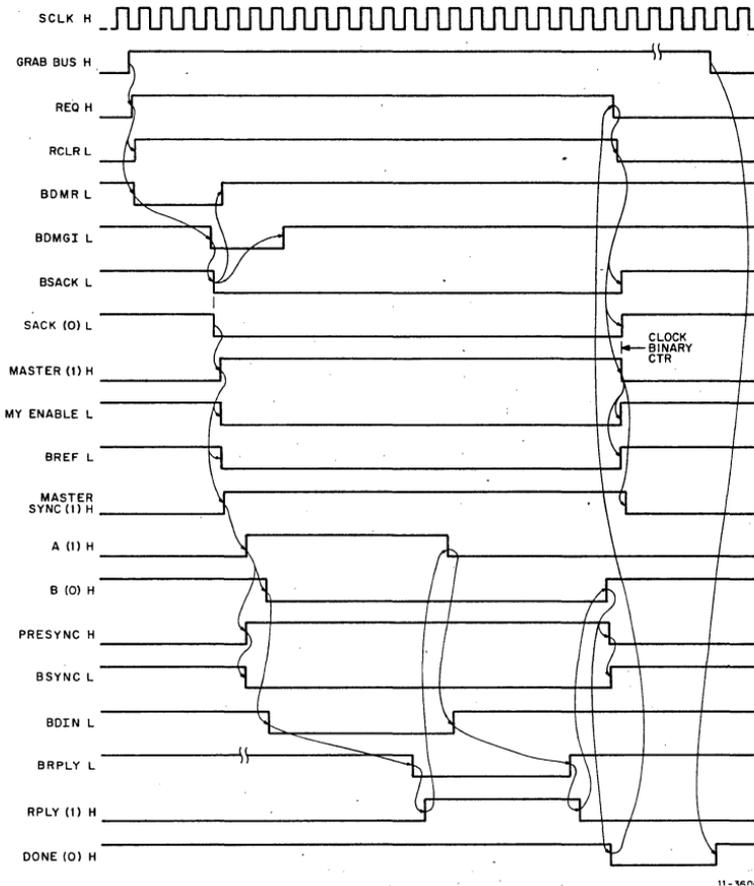
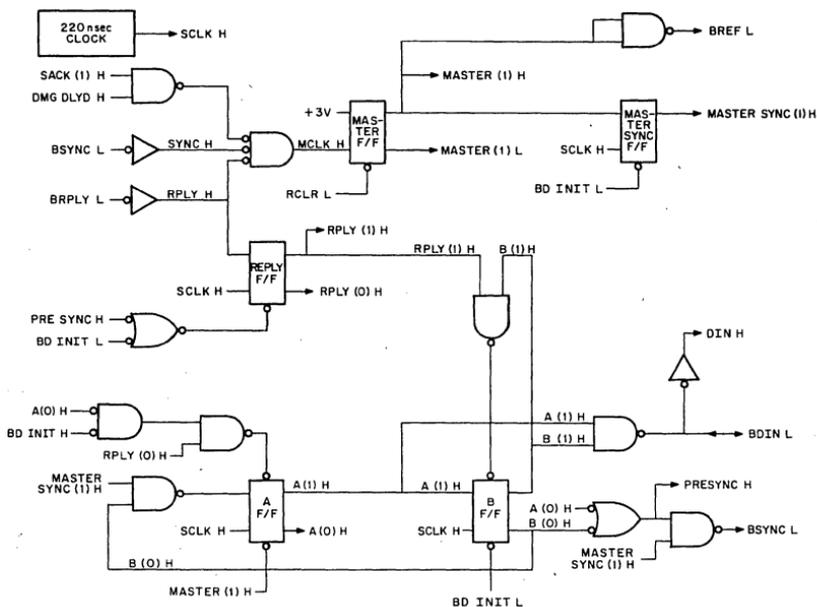


Figure 4-36 DMA Refresh Logic Signal Sequence

MASTER (1) H is applied to refresh address logic where it produces the MY ENABLE L signal. This signal enables low byte BDAL (0-7) bus drivers, and row address bits are placed on the BDAL bus.

The sequence of operations involving the bus control logic is controlled by sequence flip-flops A and B. (Note that the A flip-flop is shown inverted.) Operations are synchronized by the positive-going leading edge of the 220 ns clock SCLK H signal, as shown in Figure 4-36. On the SCLK H leading edge following the active MASTER (1) H signal, the Master-Sync flip-flop clocks to the set state, producing the active MASTER SYNC (1) H signal. The MASTER SYNC (1) H and B (0) H signals are gated to produce a low signal that clocks sequence flip-flop A to the set state on the following SCLK H pulse. The RPLY (0) H signal



11-3605

Figure 4-37 Bus Control Logic

is initially high since the refresh bus transaction with system memory has not been completed. Thus, the high RPLY (0) H, passive (low) BDINIT H, and A (0) H signals are gated, producing a low signal that keeps the A flip-flop set until the RPLY (0) H signal goes low. The low A (0) H signal is ORed with B (0) H to produce the PRE SYNC H signal. PRE SYNC H and the active MASTER (1) H signal are gated by the BSYNC L bus driver, causing that bus signal to become asserted.

On the third SCLK H leading edge, the B sequence flip-flop clocks to the set state, producing the high B (1) H and low B (0) H signals. B (1) H and A (1) H are gated by the BDIN L bus driver, causing that signal to become asserted. Bus control logic remains in this state until system memory responds to the refresh transaction by asserting BRPLY L. (This may occur one or more SCLK H signals later, depending upon system delays.)

BRPLY L is received and inverted, producing RPLY H. On the next leading edge of SCLK H, the reply flip-flop clocks to the set state, and high RPLY (1) H and low RPLY (0) H signals are produced. The low RPLY (0) H signal inhibits the A sequence flip-flop clear gate, and the flip-flop clocks to the reset state on the following SCLK H pulse. A (1) H goes low, inhibiting the BDIN L bus driver, and terminating that signal. RPLY (1) H and B (1) H are ANDed to produce a low signal which presets the B sequence flip-flop. This prevents resetting the B flip-flop as long as RPLY (1) H is in the active state.

System memory responds to the passive BDIN L signal by terminating the BRPLY L signal. On the next SCLK L pulse, the reply flip-flop clocks to the reset state; RPLY (1) H goes low and RPLY (0) H goes high. The following SCLK L pulse then clocks the B sequence flip-flop to the reset state and B (1) H goes passive; PRE SYNC H and BSYNC L go to the passive states.

Low R PENDING (1) H and DONE (0) H signals are gated to produce the active (low) RCLR L signal (Figure 4-35) which clears the Sack flip-flop; BSACK L and SACK (0) H signals go high. RCLR L also clears the Master flip-flop in the bus control logic (Figure 4-37), causing BREF L to go passive and MY ENABLE L in the refresh address logic (Figure 4-38) to go passive, inhibiting the BDAL bus drivers. On the next SCLK H pulse, MASTER SYNC (1) H goes passive (Figure 4-37), and the refresh bus transaction is completed. The passive MASTER SYNC (1) H signal resets the Done flip-flop enabling the next refresh operation.

**4.10.4.4 Refresh Address Logic**—Refresh address logic is shown in Figure 4-38. A 6-bit binary counter produces the six row address bits that are placed on the BDAL bus during the addressing portion of the refresh bus transaction. The low MASTER (0) H signal enables the counter output bits during the operation. Upon completing the transaction, RPLY (1) H goes passive, inhibiting the CK L gate; the CK L signal goes high, incrementing the 6-bit binary counter by one count. Hence, on each successive refresh operation, a new row address is used.

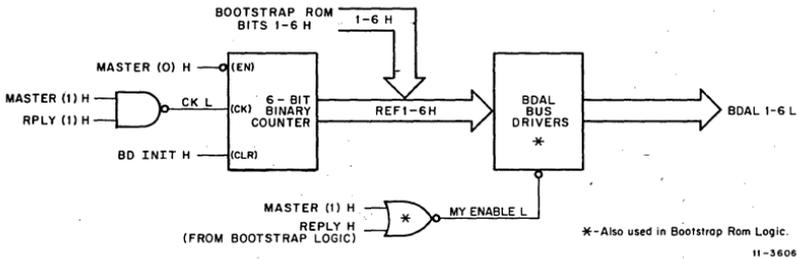


Figure 4-38 Refresh Address Logic

**4.10.4.5 Initialization**—DMA refresh logic initialization is controlled by the BDCOK H signal. Initialization occurs only during power-up or power-down conditions (when BDCOK H is in the low passive state). All flip-flops (except the Done flip-flop) are initialized by either the BD INIT H (inverted-passive BDCOK H), BD INIT L (inverted BD INIT H), or RCLR L (gated BD INIT H) signals.

## 4.11 H780 POWER SUPPLY

### 4.11.1 General

The H780 power supply provides dc operating power to all backplane slots contained in a PDP-11/03 microcomputer system. Depending on the configuration ordered, the primary power input is 115 or 230 Vac, 50 or 60 Hz. In addition to providing operating power, the H780 gen-

erates power supply status and line time clock signals that are distributed over the LSI-11 bus. Three LED indicators and three switches are on the H780's front panel. The indicators include RUN, which illuminates when the LSI-11 processor is in the run state, and DC ON, which illuminates when normal dc operating voltages are applied to the LSI-11 backplane. The DC ON indicator status is controlled by circuits contained in the H780. The DC ON/OFF switch allows the operator to turn off the H780 dc output voltages without turning off system primary power. This allows safe module installation or removal with no dc power applied to the backplane. A normal power-up/power-down sequence is produced when this switch is operated. The ENABLE/HALT switch enables the operator to manually assert the BHALT L bus signal, causing the LSI-11 microcomputer to enter the console (ODT) microcode. When in the ENABLE position, program execution can be initiated via console ODT commands. The LTC ON/OFF switch enables or disables H780 generation of the line time clock (BEVNT L) signal. One spare LED indicator is included. Two fans provide cooling air for the H780 power supply and all modules contained in the PDP-11/03 enclosure.

The H780 features:

- $+5\text{ V} \pm 3\%$ , 18 A (maximum) and  $+12\text{ V} \pm 3\%$ , 3.5 A (maximum); combined dc power must not exceed 120 W.
- Overcurrent/short circuit protection—Output voltages return to normal after removal of overload or short. Current limited to approximately 1.2 times the required maximum rating.
- Overvoltage protection— $+5\text{ V}$  limited to  $+6.3\text{ V}$  (approx);  $+12\text{ V}$  limited to  $+15\text{ V}$  (approx).
- Dual primary power configuration—Can be connected for nominal 115 V, 60 Hz or 230 V, 50 Hz input power.
- System control/indicator panel—A simple system control/indicator panel allows the user to control dc power on/off and microcomputer Run/Halt mode. Indicators display the actual dc power and processor status.
- Line Time Clock—A bus-compatible signal is generated by the power supply for the event (line time clock) interrupt input to the processor. This signal is either 50 or 60 Hz, depending upon primary power line frequency input to the power supply.
- Power Fail/Automatic Restart—Fault detection and status circuits monitor ac and dc voltages and generate bus-compatible BPOK H and BDCOK H signals (respectively) to inform the LSI-11 system modules of power supply status.
- Fans—Built-in fans provide cooling for the power supply and LSI-11 modules contained in the PDP-11/03 enclosure.

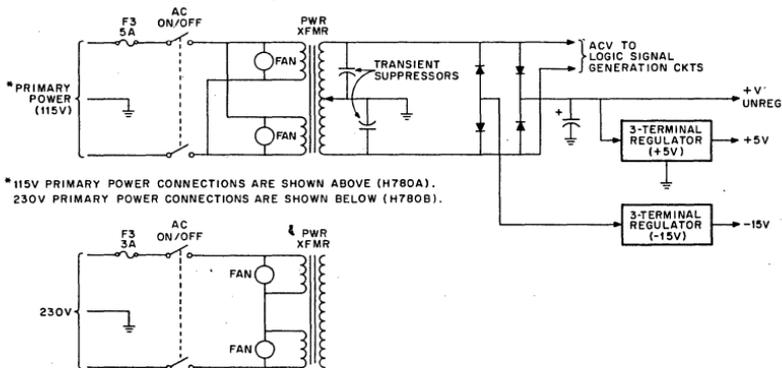
#### 4.11.2 Specifications

##### Electrical

|                          |  |
|--------------------------|--|
| Input Voltage:           | 100–127 V rms, $50 \pm 1\text{ Hz}$ or $60 \pm 1\text{ Hz}$<br>200–254 V rms, $50 \pm 1\text{ Hz}$ or $60 \pm 1\text{ Hz}$ |
| Input Power (full load): | 400 W maximum  |

|  |  |
|--|--|
| <b>Output Voltages:</b>                    | +5 V $\pm$ 3%, 1.5–18 A load (static and dynamic)<br>+12 V $\pm$ 3%, 2.5–3.5 A load (static and dynamic)<br>Maximum output power: 110 W (total)                    |
| <b>Output Protection:</b>                  | Current limited to 1.2 times maximum normal rating (approximately)<br>Voltage +5 V and +12 V outputs limited to +6.3 V (nominal) and +15 V (nominal), respectively |
| <b>Output Ripple:</b>                      |  |
| 5 V output:                                | Less than 150 mV peak-to-peak  |
| 12 V output:                               | Less than 360 mV peak-to-peak  |
| <b>Output Regulation:</b>                  |  |
| Line:                                      | +5 V, 0.5% max<br>+12 V, 0.25% max<br>(Static and dynamic ( $\Delta I < 0.1 \text{ A}/\mu\text{s}$ ):  |
| Load:                                      | +5 V, 1% max<br>+12 V, 0.5% max  |
| Load Interaction:                          | 1.0%   |
| <b>Load Term Stability:</b>                | 0.1%/1000 hr max   |
| <b>Line Protection:</b>                    |  |
| H780A (115 V input):                       | Fast blow 5 A fuse   |
| H780B (230 V input):                       | Fast blow 2.5 A fuse   |
| <b>Noise:</b>                              | AC component above 100 kHz meets DEC STD 102.7; H780B units will meet VDE N-12 limits for European environment.  |
| <b>Front Panel Control and Indicators:</b> | DC ON/OFF switch<br>HALT/ENABLE switch<br>LTC ON/OFF switch<br>RUN indicator<br>DC ON indicator<br>Spare indicator   |
| <b>Rear Panel Controls and Indicators:</b> | AC ON/OFF power switch   |
| <b>Backplane Signals:</b>                  | BPOK H<br>BDCOK H<br>BEVNT L<br>BHALT L<br>SRUN L  |
| <b>Mechanical</b>                          |  |
| <b>Cooling:</b>                            | Two self-contained fans provide 200 LFPM air flow.   |
| <b>Size:</b>                               | 6 $\frac{1}{8}$ in. w $\times$ 3 $\frac{1}{2}$ in. h $\times$ 14 $\frac{5}{8}$ in. d   |
| <b>Weight:</b>                             | 13 lbs   |
| <b>Environmental</b>                       |  |
| <b>Temperature:</b>                        | 5°–50° C operating   |
| <b>Humidity:</b>                           | 90% maximum without condensation.  |





CP-1793

Figure 4-40 Unregulated Voltage and Local dc Power

V and +12 V regulators use the same +24 V unregulated voltage for regulation and distribution to LSI-11 modules. AC voltage from one side of the transformer secondary is also routed to the line time clock (LTC) circuit, which generates a BEVNT L bus signal for a line time clock processor interrupt. When used with a 60 Hz line frequency, the interrupt occurs at 16.667 ms intervals; a 50 Hz line frequency will produce interrupts at 20 ms intervals.

**4.11.3.3 Basic Regulator Circuit**—Both +5 V and +12 V regulator circuits receive the +24 V unregulated input power. The +5 V and +12 V regulator circuits are identical except for component values. Hence, only the basic +5 V regulator is described in detail.

The basic regulator is a switching regulator which operates at approximately 20 kHz. The main controlling element is a 3-terminal regulator which operates at approximately the regulated output voltage level. Basic regulator circuits are shown in Figure 4-41. Note that the ground terminal of the 3-terminal regulator is connected to a circuit that allows factory adjustment of the terminal voltage over a  $-0.7$  to  $+0.5$  V range. Hence, the 3-terminal regulator output in the +5 V regulator circuit can range from 4.3 to 5.4 V (approx).

Normal switching regulator operation is accomplished when the control transistor is turned on. Forward bias for the control transistor is supplied via R14. It is turned off only during fault conditions (overcurrent or shorted output voltage) or when the input ac line voltage is below specifications. Its emitter supplies unregulated voltage to the 3-terminal regulator. At less than 50 mA regulator output current (approx), the 3-terminal regulator supplies the output voltage. However, as load current through the 3-terminal regulator is increased beyond this value, the voltage drop across R27 forward biases the driver transistor. The pass switch transistor then turns on and applies the unregulated +24 V to L2. The output capacitor then charges toward the +5 V value, current limited by the inductance of L2. When the output voltage rises to the 3-terminal regulator regulation voltage, the 3-terminal regulator turns off; current through R27 stops, and the driver transistor is not forward

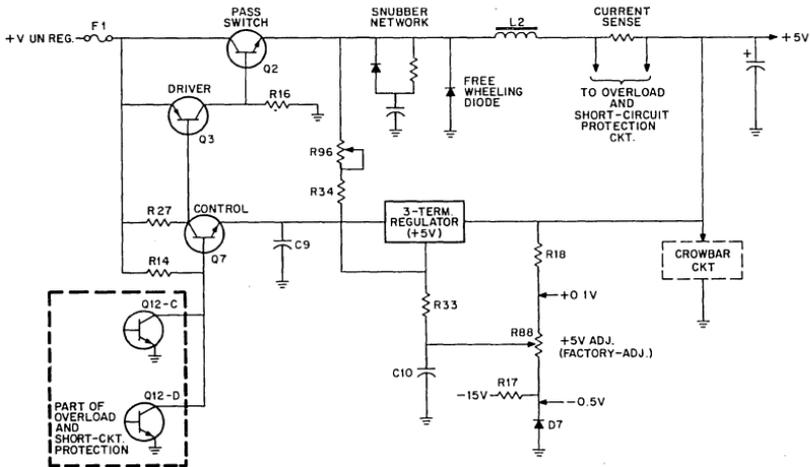


Figure 4-41 Basic Regulator Circuit

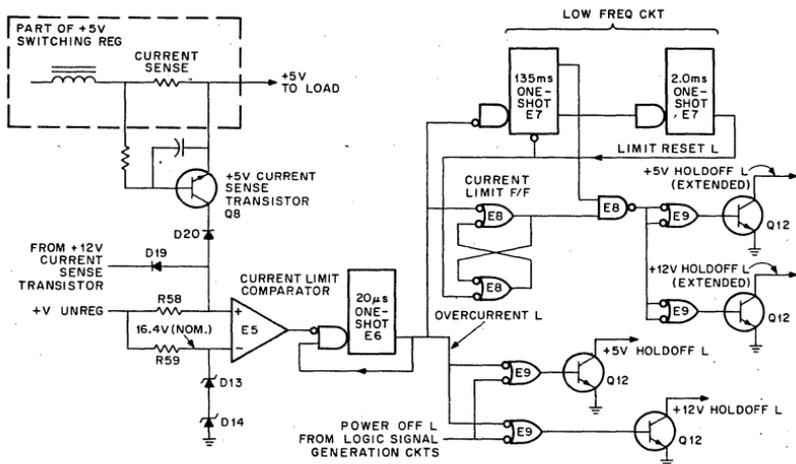
biased. Hence the driver and pass switch transistors cut off. The energy stored in L2 continues to charge the capacitor bank slightly beyond the designed output voltage via the free-wheeling diode and the current sense resistor. Once the inductor's stored energy is spent, the load discharges the output capacitor until the output voltage drops below the 3-terminal regulator's regulation voltage. At that point, current through R27 increases and turns on the driver and pass switch transistors, and the cycle repeats. Note that as the load is increased, the pass switch must remain on longer in order to charge the output capacitor to the regulated voltage value. This process repeats at a 12-20 kHz rate, producing the switching regulator operation.

Switching losses in the pass switch transistor are minimized by the snubber network. This network operates during the "off" switching transient (as the pass switch is biased off) by controlling the rate of increasing collector to emitter voltage as collector current decreases.

The control transistor is turned off during a fault condition by overload and short-circuit protection circuits. When a fault condition is detected, the control transistor's base voltage drops to nearly 0 V, causing it to cut off. When cut off, operating voltage is removed from the 3-terminal regulator and R27 current is 0, disabling the switching regulator circuit.

**4.11.3.4 Overload and Short-Circuit Protection**—Each H780 dc output is overload and short-circuit protected. When in an overload condition, excessive power supply current is sensed, causing both switching regulators to go off and then cycle on and off at a low-frequency rate (approximately 7.5 Hz) until the overload is removed. Each time the power supply cycles on, the circuit checks for the overload condition. If the load current returns to normal, the 20 kHz switching regulator operation resumes.

Overcurrent sensing circuits for +5 V and +12 Vdc outputs are identical except for component values. A 5 V power supply overcurrent condition results in an increased voltage drop across the current sense resistor (Figure 4.42), forward biasing the current sense transistor. (During nor-



CP-1796

Figure 4-42 Overload and Short-Circuit Protection

mal operation, this transistor is not forward biased.) Current sense transistor collector voltage then drops from the normal +24 V (approx) to the +5 V regulator output value; this voltage, which is less than the +16 V reference applied to the current limit comparator's inverting input, is diode-coupled to the comparator's non-inverting input, causing the comparator's output to go low; the diode coupling provides an OR logic function for both +5 V and +12 V overcurrent fault conditions. The comparator's low output signal triggers the 20  $\mu$ s one-shot whose OVERCURRENT L pulse output triggers the 135 ms one-shot and sets the Current Limit flip-flop. The OVERCURRENT L pulse is also ORed with the POWER OFF L signal, turning on the +5 V and +12 V hold-off transistors. Both switching regulators are then disabled. The high 135 ms one-shot output pulse is ANDed with the Current Limit flip-flop output, turning on +5 V and +12 V extended hold-off transistors. Hold-off signals remain in this state and inhibit switching regulator operation for the 135 ms pulse duration. At the end of this time, the 135 ms one-shot resets, terminating the delayed hold-off signals, and triggers the 2.0 ms one-shot. Its active low output resets the Current Limit flip-flop and clears the 135 ms one-shot for 2.0 ms, allowing the regulator pass switch transistors to operate for 2 ms (minimum). At the end of this time, the 135 ms one-shot is again enabled (the clear input goes high) and a new overcurrent cycle is enabled. If the overload is removed, normal operation resumes; otherwise, the overload causes a new overload condition to occur and the cycle repeats, as described above.

Switching regulator operation is suspended when the operator places the DC ON/OFF switch in the OFF position. Logic signal generation circuits respond by immediately asserting BPOK H low to initiate a processor power-fail sequence. After a 5-10 ms "pseudo delay," POWER OFF L is asserted low. This low signal is wire-ORed with OVERCURRENT L, inhibiting the switching regulator operation, and dc power is removed from the backplane.

**4.11.3.5 Crowbar Circuits**—Crowbar circuits are connected across both +5 V and +12 V power supply outputs for overvoltage protection. An overvoltage condition could occur if +12 V and +5 V outputs shorted together, or if a driver or switch transistor becomes shorted. When shorted to a higher voltage source, the crowbar fires, shorting the supply voltage that it is protecting to ground (dc return). In this condition, the overload and short-circuit protection circuits respond by limiting the duty cycle of the switch transistor until the overvoltage source is removed. However, when the overvoltage is caused by a shorted driver or switch transistor, short-circuit protection is ineffective, and the excessive current caused by the crowbar circuit firing will blow the regulator's fuse (F1 for +5 V or F2 for +12 V).

The crowbar circuit for the +5 V output is shown in Figure 4-43. It com-

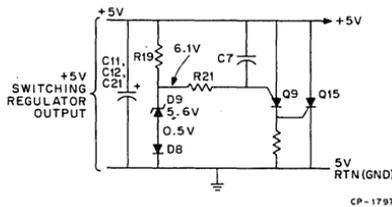


Figure 4-43 Crowbar Circuit

prises a 5.6 V zener diode D9, diode D8, programmable unijunction transistor Q9, and silicon-controlled rectifier (SCR) Q15. R19, D8, and D9 supply the 6.1 Vdc (approx) crowbar reference (threshold) voltage to the gate of Q9 via R21. Q9 is normally off and its cathode supplies a 0 V gate input to Q15. An overvoltage is coupled into the circuit via C7, causing the gate voltage of Q9 to rise; this triggers Q9 and its cathode voltage rises to the output (overvoltage) potential. Q15 then fires and shorts (crowbars) the supply output. The circuit remains in this condition until the overvoltage is removed (Q15 current goes to zero) and either the power supply switch transistor is off due to short circuit protection, or the regulator's dc fuse opens.

The +12 V crowbar circuit functions in a similar manner. However, the reference voltage for this power supply is approximately 13.5 V.

**4.11.3.6 Logic Signal Generation**—Logic signal generation circuits produce LSI-11 bus signals for power normal/power fail and line time clock interrupt functions and processor Run-Enable/Halt mode. The RUN indicator circuit monitors the SRUN L backplane (nonbused) signal and

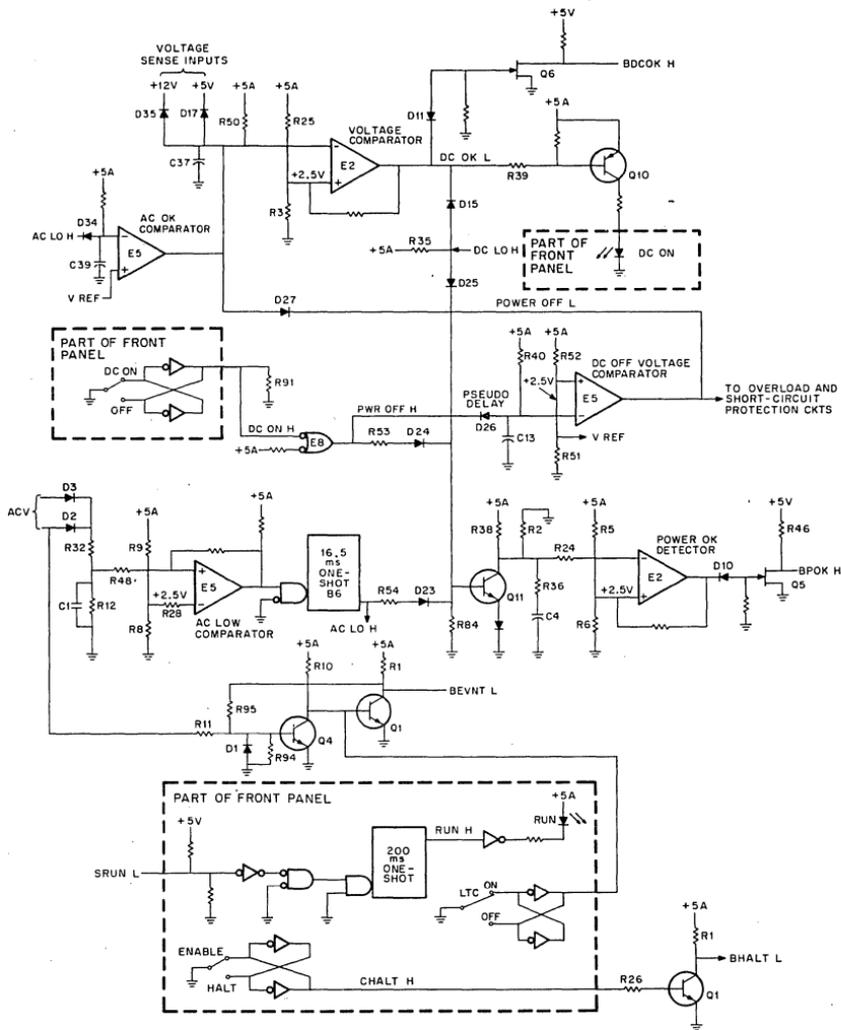
provides an active display when the processor is in the Run mode. BPOK H and BDCOK H indicate power status. When both are high, power to the LSI-11 bus is normal and no power fail condition is pending. However, if primary power goes abnormally low (or is removed) for more than 16.5 ms, BPOK H goes low and initiates a power-fail processor interrupt. If the power-fail condition continues for more than an additional 4 ms, a "pseudo-delay" circuit causes BDCOK H to go low. The circuit also causes the overload and short-circuit protection circuit to inhibit +5 V and +12 V control transistors; normal output voltages are available for 50  $\mu$ s (minimum) after BDCOK H goes low (depending on the loading of the dc output voltages). The DC ON/OFF switch simulates an AC ON/OFF operation by turning switching regulators on or off without turning system primary power off. A normal power-up/power-down sequence is produced by this circuit. The line time clock circuit produces a processor interrupt at the power line frequency (either 50 or 60 Hz). The circuit simply asserts the BEVNT L line at the line frequency.

DC voltage monitor circuits respond to both +5 V and +12 V power supply outputs. A +2.5 V reference at the voltage comparator's noninverting input is established by +5 A and a voltage divider comprised of R25 and R3, as shown in Figure 4-44. Voltages are sensed at the anodes of diodes D17 and D35.

The sensed voltage to the voltage comparator's inverting input is normally 5 V, causing the comparator's output to go low. The low signal forward biases DC ON panel indicator driver transistor Q10, producing a DC ON indication, and reverse biases the BDCOK H FET bus driver Q6. As a result, Q6 cuts off, and its source voltage rises to +5 V, producing the active BDCOK H signal.

When either (or both) power supply output is 0 V, the voltage at the voltage comparator's inverting input is less than the +2.5 V reference. Hence, the comparator's output goes high, turning off the DC ON indicator and allowing Q6 to conduct. Q6 asserts the BDCOK H signal low, indicating that a dc power-fail condition exists. When normal power is restored, as during the power-up sequence, C37 charges via R50, and its voltage exceeds the +2.5 V reference; the comparator's output then goes high (normal).

AC voltage monitor circuits include an ac low comparator, 16.5 ms delay, and a BPOK H bus driver circuit which is enabled only when BDCOK H is in the active (dc voltage normal) state. Rectifiers D2 and D3 produce positive-going dc voltage pulses at twice the ac line frequency. R32, R12, and C1 produce nominal +3.9 V (peak) normal line voltage pulses which are coupled to the noninverting input of the ac low comparator via R48. R8 and R9 produce a +2.5 V reference for the comparator's inverting input. The comparator's normal output is a series of pulses occurring at twice the ac power line frequency. Each positive-going leading edge retriggers the 16.5 ms one-shot, keeping it in the set state. The 16.5 ms one-shot output is diode-ORed with DCOK L via diodes D25 and D23 and PWR OFF H via D24. Normally, the three signals are low and Q11 remains cut off. In this condition, C4 charges to +3.125 V via R36 and R38. This signal is then applied to the power OK comparator's inverting input via R24. Since the noninverting input is referenced to +2.5 V by



CP-1750

Figure 4-44 Logic Signal Generation

voltage divider R5 and R6, the comparator's output goes low, biasing off FET Q5. Q5's source voltage then rises toward +5 V via R46 producing the active BPOK H signal. Power-up/power-down sequence timing is shown in Figure 4-45.

A power failure is first detected when the pulsating dc voltage at the ac low comparator's noninverting input is less than +2.5 V (peak). The comparator's output then remains low, allowing the 16.5 ms one-shot to go out of the retrigger mode. The one-shot resets 16.5 ms after the leading edge of the last valid ac voltage alternation; the 16.5 ms delay is equivalent to a full line cycle (two-alternation) failure. The high one-

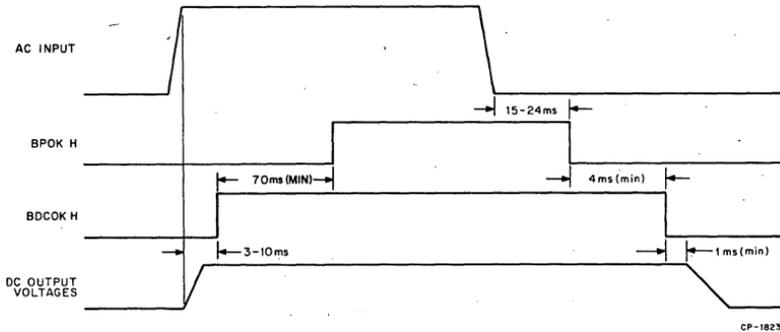


Figure 4-45 Power-Up/Power-Down Sequence

shot output is then coupled via D23 to the base of Q11, forward biasing it. Q11 conducts and rapidly discharges C4; R36 limits peak discharge current. The low voltage thus produced is less than the +2.5 V reference at the power OK comparator's input, and its output goes high. Q5 then conducts and asserts the BPOK H signal low (power fail). The AC LO H signal produced by the 16.5 ms one-shot is coupled via D34 to C39 on the inverting input of AC OK comparator E5. When C39's voltage rises above 2.5 V, the comparator's output goes low, turning off the DC ON indicator and negating BDCOK via the DC voltage monitor circuit, and turns off the regulator circuits by asserting POWER OFF L via D27.

When normal power is restored, the 16.5 ms one-shot returns to the retrigger (set) mode. AC LO H goes low and enables the DC voltage monitor and regulator circuits. The low AC LO H signal also removes forward bias from the base of Q11, cutting it off. Its collector voltage then rises as C4 charges at a relatively slow rate. R38 controls the charging rate of C4 and ensures that ac voltage and dc output voltages are normal for approximately 100 ms (70 ms minimum) before BPOK H goes high.

The DC ON/OFF switch simulates a power failure when it is placed in the OFF position. Cross-coupled inverters provide switch debounce protection and a low (false) DC ON H signal is produced. This signal is inverted to produce a high PWR OFF H signal that is coupled via D26 to the "pseudo delay" circuit, causing a power fail sequence to occur, and to Q11 via R53 and D24, causing BPOK H to go low (power fail indication). After a 5-10 ms (approximately) "pseudo delay," C13's voltage rises above the dc off voltage comparator's +2.5 V reference (nonin-

verting) input. The comparator's output goes low, asserting POWER OFF L low and turning off the switching regulators (Paragraph 4.11.3.4). When the DC ON/OFF switch is returned to the ON position, PWR OFF H goes low, rapidly discharging C13. POWER OFF L then goes high and switching regulator operation resumes. Approximately 100 ms later, BPOK H goes high and normal processor operation is enabled. DC ON/OFF circuit timing is shown in Figure 4-46.

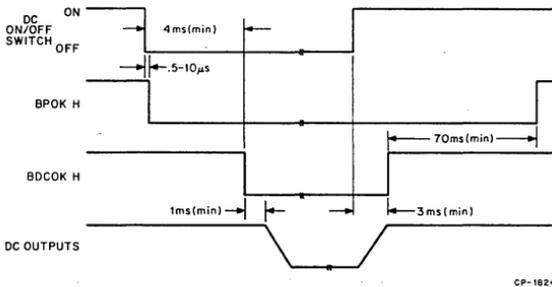


Figure 4-46 DC ON/OFF Circuit Timing

BEVNT L is the bused EVENT line which is normally used for line time clock interrupts. Q4 is cut-off and Q1 is forward-biased during negative alternations of the ac line, producing low-active BEVNT L signals. D1 clips negative alternations and limits Q4's reverse base to emitter voltage. The LTC ON/OFF switch must be in the ON position for BEVNT L signal generation. When the LTC function is not desired, the LTC switch is set to the OFF position; CSPARE2 goes low, Q1 remains cut-off, and BEVNT L remains passive (high).

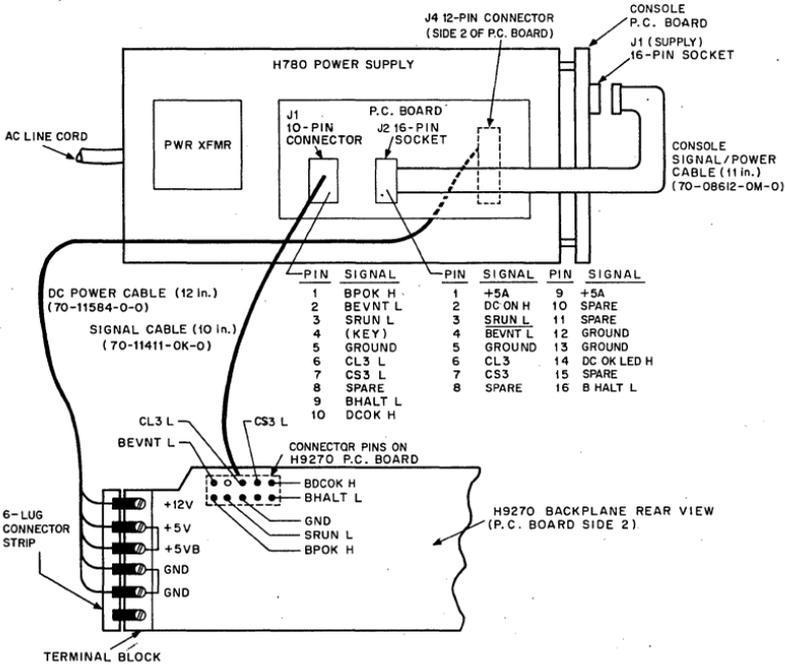
The RUN indicator is illuminated whenever the processor is executing programs. SRUN L, a non-bused backplane signal, is a series of pulses which occur at 3-5  $\mu$ s intervals whenever the processor is in the Run mode. The pulses trigger a 200 ms one-shot on each SRUN L pulse leading edge, keeping it in the retrigger mode. Its high RUN H output is then inverted, producing a low signal that turns on the RUN indicator. When the processor is in the Halt mode, SRUN L pulses cease and the 200 ms one-shot resets after the 200 ms delay. The RUN indicator turns off, indicating the Halt mode.

The HALT/ENABLE switch allows the operator to manually assert the BHALT L signal low, causing the processor to execute console ODT microcode. When in the ENABLE position, BHALT L is not asserted, and the Run mode is enabled. Cross-coupled inverters provide a switch debounce function.

#### 4.11.4 H780 Connections

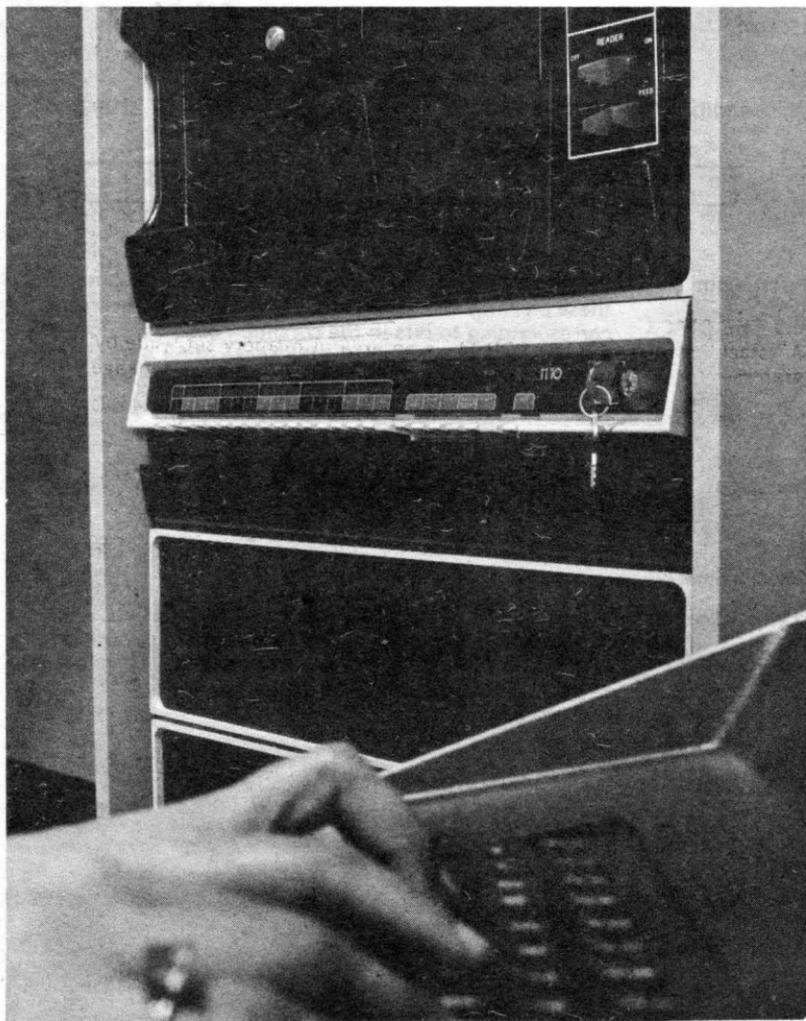
H780 connections are shown in Figure 4-47. The H9270 backplane connections and interconnecting cables are also shown. Note that cable connectors are wired 1:1. Both connectors on the H780/H9270 signal cable are 10-pin connectors which are wired in exactly the same manner,

as listed in Figure 4-47. Similarly, both ends of the panel signal/power cable are wired to 16-pin connectors in the same pin/signal configuration.



CP-1825

Figure 4-47 H780 Connections



section4  
system  
software

**CHAPTER 1 INTRODUCTION**

**CHAPTER 2 OPERATING SYSTEMS**

**CHAPTER 3 LANGUAGE PROCESSORS**

**CHAPTER 4 FOREGROUND/BACKGROUND  
OPERATING SYSTEM RT-11**

**CHAPTER 5 REAL-TIME MULTIPROGRAMMING  
RSX-11S**

**CHAPTER 6 MACRO**

**CHAPTER 7 FORTRAN IV**

**CHAPTER 8 BASIC**

## CONFIGURING LSI-11 MODULES

## 5.1 GENERAL

LSI-11 modules each contain jumpers or switches that allow the user to configure the module for a specific system application. All LSI-11 and PDP-11/03 systems will normally require some configuration of jumpers and switches. PDP-11V03 systems include LSI-11 modules which are factory configured for that system application.

LSI-11 modules that are included in this chapter are listed in Table 5-1. Note that all modules are factory configured and can be used as is in many system applications. Refer to the paragraphs listed for detailed information on factory configurations and jumper and switch functions. Configure the modules for your system application, as required.

Table 5-1 LSI-11 Module Configuration Summary

| Model No.<br>(Module(s))            | Factory Configured Application   | Reference<br>(Para.) |
|-------------------------------------|--|----------------------|
| KD11-F<br>(M7264)                   | Processor and resident 4K semiconductor read/write memory: Processor power-up mode 0 selected; resident memory bank 0 selected and reply enabled (except during refresh); event line (LTC) interrupt enabled; processor controlled memory refresh enabled. | 5.2<br>5.2           |
| KD11-J<br>(M7264-YA,<br>G653, H223) | Processor and 4K core memory: Processor power-up mode 0 selected; resident memory functions not enabled; processor controlled memory refresh disabled; event line (LTC) interrupt enabled.   | 5.2<br>5.4           |
| MSV11-B<br>(M7944)                  | 4K dynamic MOS read/write memory: Bank 0 selected; reply to refresh enabled.   | 5.3                  |
| MMV11-A<br>(G653, H223)             | 4K core memory: Bank must be selected by setting switches to appropriate positions.  | 5.4                  |
| MRV11-AA<br>(M7942)                 | 4K read-only memory: 512 × 4 chips; bank 0 selected; all BRPLY jumpers installed.  | 5.5                  |
| DLV11<br>(M7940)                    | Serial line unit: Addresses and vectors for console device use are configured; 110 baud, active current loop (Teletype) I/O selected.  | 5.6                  |
| DRV11<br>(M7941)                    | Parallel line unit: device address = 16777X; vector address = 300 and 304.   |                      |
| DRV11-B<br>(M7950)                  | DMA interface unit: Device and vector addresses must be user configured by   | 5.7                  |

**Table 5-1 LSI-11 Module Configuration Summary (Cont.)**

| <b>Model No.<br/>(Module(s))</b> | <b>Factory Configured Application</b>   | <b>Reference<br/>(Para.)</b> |
|----------------------------------|---|------------------------------|
|                                  | setting switches to appropriate positions. Summary information is contained in referenced paragraph. Refer to detailed information included with the option for complete user instructions.                         |                              |
| DRV11-P<br>(M7948)               | LSI-11 bus foundation module: Extensive user configuring required. Summary information is contained in referenced paragraph. Refer to detailed information included with the option for complete user instructions. | 5.8                          |

## 5.2 LSI-11 PROCESSOR MODULE

### 5.2.1 General

Before installing and using the KD11-F or KD11-J processor in the LSI-11 or PDP-11/03 system, the user must select certain processor features (jumper-selected), determine where the processor and option modules should be installed on the backplane, be aware of trap and interrupt functions, and ensure the conditions for bus initialization. These items are discussed in detail in the following paragraphs.

### 5.2.2 Processor Module Jumpers

**5.2.2.1 General**—The processor module contains 11 wire wrap posts that allow the user to configure the module for a specific system application. KD11-F and KD11-J processor modules are factory configured as shown in Table 5-2. Jumpers can be user-configured as described in the following paragraphs. Jumpers are located on the processor module as shown in Figure 5-1.

**Table 5-2 KD11-F and KD11-J Factory-Installed Jumpers**

| <b>Jumper</b> | <b>KD11-F (M7264)</b> |   | <b>KD11-J (M7264-YA)</b> |  |
|---------------|-----------------------|---|--------------------------|--|
|               | <b>Status</b>         | <b>Function</b>                             | <b>Status</b>            | <b>Function</b>                              |
| W1            | R                     | Resident memory bank 1 not selected         | R                        | Resident memory bank 1 not selected          |
| W2            | I                     | Resident memory bank 0 selected             | R                        | Resident memory bank 0 not selected          |
| W3            | R                     | Event line (LTC) interrupt enabled          | R                        | Event line (LTC) interrupt enabled           |
| W4            | R                     | Processor-controlled memory refresh enabled | I                        | Processor-controlled memory refresh disabled |

**Table 5-2 KD11-F and KD11-J Factory-Installed Jumpers (Cont.)**

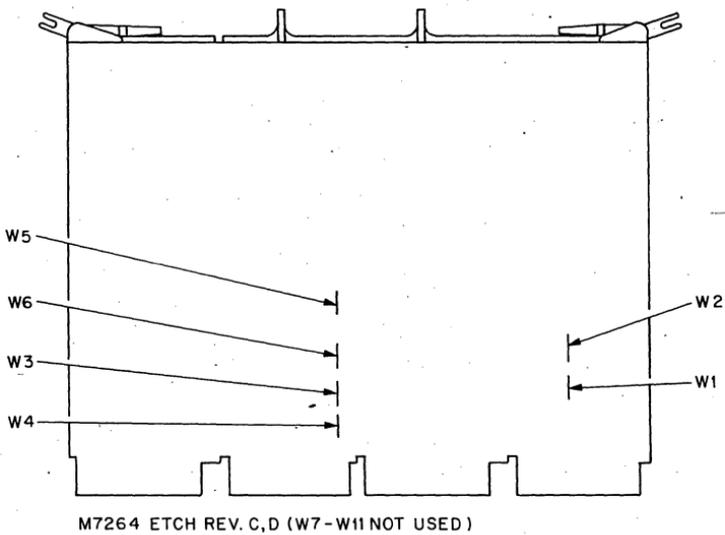
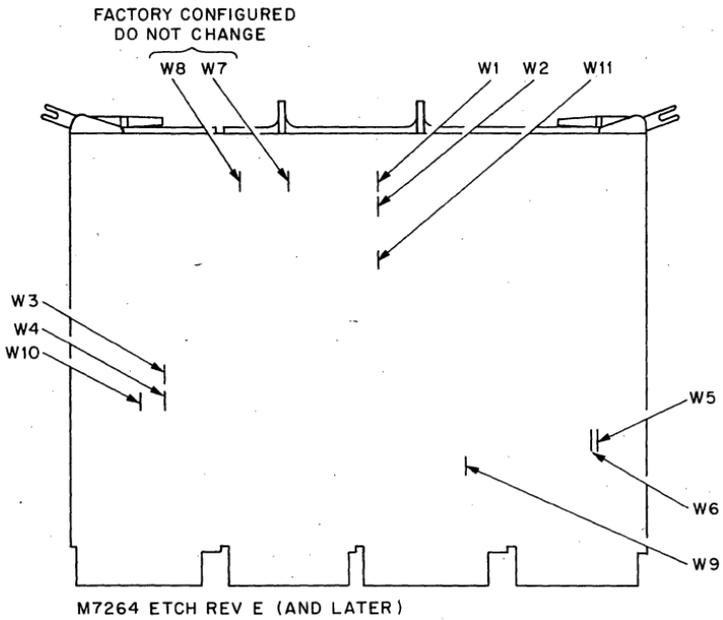
| Jumper | KD11-F (M7264) |   | KD11-J (M7264-YA) |   |
|--------|----------------|---|-------------------|---|
|        | Status         | Function  | Status            | Function  |
| W5     | R              | Power-up mode 0 selected                          | R                 | Power-up mode 0 selected                          |
| W6     | R              |   | R                 |   |
| W7     | —              | Factory-configured bias voltage (do not change)   | —                 | Factory-configured bias voltage (do not change)   |
| W8     | —              |   | —                 |   |
| W9     | R              | Enable reply from resident memory                 | I                 | Disable reply from resident memory                |
| W10    | R              | Disable reply from resident memory during refresh | R                 | Disable reply from resident memory during refresh |
| W11    | I              | Enable on-board memory select                     | R                 | Disable on-board memory select                    |

Note: I = Installed; R = Removed.

**5.2.2.2 Memory Refresh**—The LSI-11 processor has the capability of completely controlling the refreshing of all dynamic MOS memories in a system when jumper W4 is removed. Memory refresh is always required when dynamic MOS memory devices are used in the LSI-11 system, such as the KD11-F resident memory and the MSV11-B 4K by 16-bit read/write memory module. The refresh operation can be controlled by a device other than the LSI-11 processor, if available, such as the REV11-A, REV11-C, and REV11-H options. If such a device is used, or if no dynamic MOS memory devices are present in the system (KD11-J), install W4. The refresh sequence is described below.

The processor's memory refresh sequence is controlled by resident microcode in the processor which is initiated by an interrupt that occurs once every 1.6 ms. It is the highest priority processor interrupt, and cannot be disabled by software using PSW bit 7. Once the sequence is initiated, the processor will execute 64 BSYNC L/BDIN L bus transactions while asserting BREF L. The BREF L signal overrides memory bank address bits 13—15 and allows all memory units to be simultaneously enabled. After each bus transaction, BDAL1—6 L is incremented by 1 until all 64 rows have been refreshed by the BSYNC L/BDIN L transactions. This process takes approximately 130 $\mu$ s during which external interrupts (BIRQ L and BEVNT L) are ignored. However, DMA requests can be granted between each of the 64 refresh transactions.

**5.2.2.3 Line Time Clock**—LTC (or external event) interrupts are enabled when jumper W3 is removed and the processor is running. The jumper can be inserted to disable this feature. The LTC interrupt is initiated by an external device when it asserts the BEVNT L signal. This is the highest priority external interrupt request; processor interrupts



11-4290

Figure 5-1 KD11-F and KD11-J Jumper Locations

have higher priorities. If external interrupts are enabled (PS bit 7 = 0), the processor PC (R7) and PS word are pushed onto the processor's stack. The LTC (or external event device) service routine is entered by vector address 100; the usual interrupt vector address input operation by the processor is not required since vector 100 is generated by the processor.

The first instruction of the service routine will typically be fetched within 16  $\mu$ s from the time BEVNT L is asserted; however, if optional EIS/FIS instructions are being executed, this time could extend to 44.1  $\mu$ s maximum. This time could also be extended by processor trap execution (memory refresh, T-bit, power fail, etc.), or by asserting the BHALT L signal.

**5.2.2.4 Power-Up Mode Selection**—Since the LSI-11 can be used in a variety of system applications that have either (or both) volatile (semiconductor read/write) or nonvolatile (PROM or core) memory, one of four power-up mode features are available for user selection. These are selected (or changed) by wire-wrap jumpers W5 and W6 on the KD11-F or KD11-J processor (M7264) module. Note that the jumpers affect only the power-up mode (after BDCOK H and BPOK H have been asserted); they do not affect the power-down sequence.

The state of the BHALT L signal is significant during the power-up sequence. When this signal is asserted, it causes the processor's ODT console microcode (a subset of an Octal Debugging Technique program) to become invoked after the power-up sequence. The console device must be properly installed for correct use of the BHALT L signal.

The power-up modes are listed in Table 5-3. Detailed descriptions of each mode are provided in the paragraphs which follow.

**Table 5-3 Power-Up Modes**

| Mode | Jumpers |    | Mode Selected                                    |
|------|---------|----|--|
|      | W6      | W5 |  |
| 0    | R       | R  | PC at 24 and PS at 26, or Halt mode              |
| 1    | R       | I  | ODT Microcode                                    |
| 2    | I       | R  | PC at 173000 for user bootstrap                  |
| 3    | I       | I  | Special processor microcode<br>(not implemented) |

Note: R = Jumper Removed; I = Jumper Installed.

#### **Power-Up Mode 0**

This option places the processor in a microcode sequence that fetches the contents of memory locations 24 and 26 and loads their contents into R7 and the PS, respectively. A microcode service translation at this point interrogates the state of the BHALT L signal; depending on the state of this signal, the processor either enters ODT microcode (BHALT L asserted low) or begins program execution with the current contents of R7 as the starting address (BHALT L not asserted).

Note that the T-bit (PS bit 4) is loaded with the contents of PS bit 4 in location 26. This mode should be used only with nonvolatile memory locations 24 and 26 or with BHALT L asserted. This power-up sequence is shown in Figure 5-2.

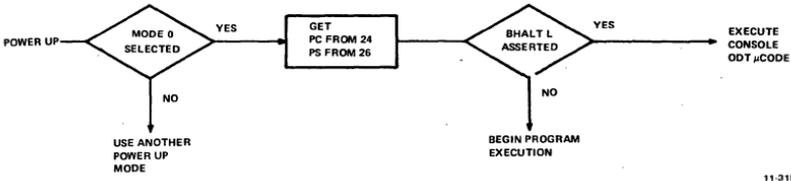


Figure 5-2 Model 0 Power-Up Sequence

**Power-Up Mode 1**

This mode immediately places the processor in the console microcode regardless of the state of the BHALT L signal. This mode assumes a console interface device at bus address 177560.

**Power-Up Mode 2**

This mode places the processor in a microcode sequence that loads a starting address of 173000 into R7 and begins program execution at this location if the BHALT L signal is not asserted.

Note that before 173000 is loaded into R7, PS bit 4 (T-bit) is cleared and bit 7 (interrupt disable) is set. The user's program must set these bits, as desired, and set up a valid stack pointer (R6). This option should be used with nonvolatile memory (ROM, PROM, or core) at address 173000. A time-out trap through location 4 will occur if no device exists at location 173000.

If BHALT L is asserted, the processor will not execute the instruction at location 173000 and will immediately execute the console microcode. This power-up mode sequence is shown in Figure 5-3.

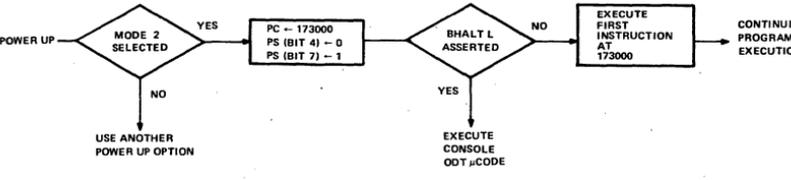


Figure 5-3 Mode 2 Power-Up Sequence

**Power-Up Mode 3**

This microcode sequence allows access to future microcode expansion in the fourth microm page (microlocations 3000 to 3777). After BDCOK H and BPOK H are asserted and the internal flags are cleared, a micro

jump is made to microlocation 3002. If this option is selected and no microm responds to the fourth page microaddress, a microtrap will occur through microlocation 0 which will, in turn, cause a reserved user instruction trap through location 10.

Note that the state of BHALT L is not checked before control is transferred to the fourth microm page.

**5.2.2.5 Resident Memory 4K Address Selection**—Jumpers W1 and W2 are used for selecting the 4K (bank) address for the KD11-F resident memory. Only one jumper must be installed, as follows.

W1 installed = Bank 1 (addresses 20000—37776)

W2 installed = Bank 0 (address 0—17776)

**NOTE**

If no jumper is installed, the 4K resident memory will not respond to any address.

**5.2.2.6 Disable Resident Memory Reply**—Jumper W9, when removed, enables the KD11-F processor module's resident memory to assert the BRPLY L signal when the resident memory is accessed. When W9 is installed, the resident memory will not assert BRPLY L. This jumper is normally installed only on the KD11-J processor module.

**5.2.2.7 Enable Reply During Refresh**—Jumper W10, when installed, enables the processor module's resident memory to assert BRPLY L during memory refresh bus cycles; W9 must not be installed. W10 is normally installed when no optional dynamic MOS memory modules are present in the system, other than the KD11-F processor module's resident memory. When optional dynamic MOS memory modules are installed in the system, only the slowest memory module should assert BRPLY L to ensure proper refresh of that module. For example, when the processor is controlling memory refresh (W4 is removed) and optional MSV11-B memory modules are installed in the system, the slowest memory module will typically be the module located at the greatest electrical distance from the processor module.

**5.2.2.8 Enable On-Board Memory Select**—Jumper W11 is normally installed on the KD11-F processor module to enable normal bank selection, as configured by W1 and W2. W11 is removed on the KD11-J processor module. In special applications, the user can remove W11 from the KD11-F module and supply a low-active SMENB L signal to processor module backplane pin CF1 from an external source. The processor module's resident memory will be enabled whenever this signal is asserted low; bank address decoding must be provided by the external source.

### **5.2.3 Installation**

Prior to installation, the processor module jumpers must be configured as directed in Paragraph 5.2.2. PDP-11/03 systems are shipped from the factory with the KD11-F or KD11-J processor installed. Refer to Chapter 6 for LSI-11 processor module installation details.

## 5.2.4 Using The LSI-11 Microcomputer

**5.2.4.1 General**—Most of the operational characteristics are discussed in Sections II and III and related software publications. This discussion includes the use of the LTC (external event interrupt) feature, bus initialization, and trap and interrupt priority.

**5.2.4.2 Interrupts and Trap Priority**—Interrupts and traps are quite similar in their operation. Interrupts are service requests from devices external to the processor; traps are interrupts that are generated within the processor. Their main operational difference, however, is that external interrupts can only be recognized when PS priority (bit 7) is zero; traps can be executed at any time, regardless of the PS priority bit status.

The highest priority trap is memory refresh, when enabled (Paragraph 5.2.2.2). Memory refresh does not require an interrupt vector since it is entirely controlled by processor microcode; memory refresh operations are completely transparent to the user programs and PS bits are not altered in any way. The remaining traps, including EMT, BPT, IOT, and TRAP instructions, and hardware-generated Trace Trap, Bus Error, Power Fail, etc., are described in Section III. The LTC (external event) interrupt, has the highest priority of all external interrupts, when enabled (Paragraph 5.2.2.3). It is acknowledged (serviced) only when PS priority bit 7 = 0. This interrupt always uses vector address 100. It loads a new PC from location 100 and a new PS from location 102. All other external interrupts are requested by a device asserting the BIRQ signal. If PS bit 7 = 0, the request is acknowledged and the processor inputs a user-assigned vector address to the device's service routine PC (starting address) and PS. For example, when the requesting device is the console device, vectors 60 (console input) or 64 (console output) are used. These vectors are reserved for the console device by most DIGITAL software systems.

**5.2.4.3 Halt Mode**—The LSI-11 microcomputer can operate in either a Run or Halt mode. When in the Halt mode, normal program execution is not performed and the processor executes ODT console microcode. However, the processor will execute memory refresh in a normal manner and arbitrate DMA requests; all external interrupts are ignored.

The Halt mode can be entered in one of six ways:

1. When the BHALT L signal is asserted.
2. When a HALT instruction has been executed.
3. By power-up sequence.
4. When a double bus error has occurred [a bus error trap with SP (R6) pointing to non-existent memory].
5. No Reply received from a device (bus time-out error) when the processor attempts to input a vector during an interrupt transaction.
6. A bus error (time-out) occurs when the processor refreshes one of 64 memory rows.

The LSI-11 microcomputer does not use conventional control panel lights and switches. Instead, the ODT console microcode routine pro-

vides all control panel features on a peripheral device that can be interfaced at bus address 177560 and interpret ASCII characters. In a typical configuration there is no bus device that responds to address 177570 (the PDP-11 SWR address). The peripheral device used with the ODT console microcode is called the console device, which can be any device capable of interpreting ASCII characters. The prompt character sequence and detailed use of console ODT commands are contained in Section II.

### **5.2.5 Initialization and Power Fail**

Initialization occurs during a power-up or power-fail sequence, or when a RESET instruction is executed. The processor responds to these conditions by asserting the BINIT L bus signal. BINIT L can be used to clear or initialize all device registers on the bus. In addition, the DRV11 parallel line unit applies the buffered initialize signal to pins on both of its device interface connectors for initializing the user's device.

During the power-up sequence, the processor asserts BINIT L in response to a passive (low) power supply-generated BDCOK H signal. When BDCOK H goes active (high), the processor terminates BINIT L and the jumper-selected power-up sequence is executed. Similarly, if power fails, the power supply-generated BPOK H signal goes passive (low) and causes the processor to push the PC and PS onto the stack and enter a power-fail routine via vector location 24. The processor will execute a user power-fail routine until either BDCOK H goes passive (low), indicating that dc operating power may not sustain processor operation, or BPOK H returns to the active state. BINIT L will go active if BDCOK H goes passive.

Note that if a HALT instruction is executed after entering the power-fail routine, the ODT microcode will not be executed until BPOK H is re-asserted. If BPOK H goes passive while the processor is in the Halt mode, the power-fail routine will not be executed.

## **5.3 MSV11-B READ/WRITE MEMORY**

### **5.3.1 General**

The MSV11-B(4K) read/write memory (Figure 5-4) provides temporary storage of user programs and data in an inexpensive, compact, low-power memory subsystem. The user can select the 4K address space (bank) in which the module is addressed by installing or removing jumpers.

The MSV11-B module is factory configured to respond to addresses in bank 0 addresses 0-17776). The module will also reply to refresh signals.

### **5.3.2 Address Jumpers**

MSV11-B address jumpers are located as shown in Figure 5-5. The module is supplied with all address jumpers installed. Figure 5-6 illustrates a 16-bit address and how jumpers are assigned for the MSV11-B module.

### **5.3.3 Reply To Refresh Jumpers**

Only one dynamic memory module in a system is required to reply to the refresh bus transactions initiated by the processor. The module

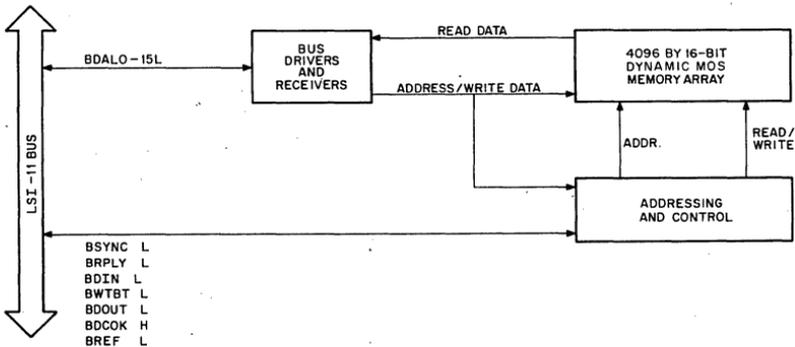
selected to reply should be the module with the slowest access time. Jumper W4 enables or inhibits the MSV11-B reply as follows:

**W4 installed:** MSV11-B will not assert BRPLY in response to refresh bus signals.

**W4 removed:** MSV11-B will reply to refresh bus BSYNC/BDIN transactions by asserting BRPLY L.

### 5.3.4 Refresh Requirements

The MSV11-B module contains dynamic MOS memory integrated circuits. Hence, memory refresh cycles are required. Refresh cycles can be either provided automatically by the LSI-11 processor module or by the DMA refresh circuits contained on the REV11-A, REV11-C, or REV11-H options. One complete refresh operation consists of 64 refresh bus cycles. When refresh is controlled by the processor, 64 successive bus cycles are executed; a new refresh operation is initiated by the processor at 1.6 ms (approximately) intervals. The REV11 options execute single refresh bus cycles at 27  $\mu$ s (approximately) intervals via DMA bus cycles. A complete refresh operation must be completed every 2 ms maximum.



CP-1748

Figure 5-4 MSV11-B 4K by 16-Bit Read/Write Memory

| W1 | W2 | W3 | Bank No. | Address Range | Octal Address Range |
|----|----|----|----------|---------------|---------------------|
| I  | I  | I  | 0        | 0-4K          | 000000-017776       |
| I  | I  | R  | 1        | 4-8K          | 020000-037776       |
| I  | R  | I  | 2        | 8-12K         | 040000-057776       |
| I  | R  | R  | 3        | 12-16K        | 060000-077776       |
| R  | I  | I  | 4        | 16-20K        | 100000-117776       |
| R  | I  | R  | 5        | 20-24K        | 120000-137776       |
| R  | R  | I  | 6        | 24-28K        | 140000-157776       |
| R  | R  | R  | 7        | 28-32K        | 160000-177776       |

NOTE: I = Installed, R = Removed

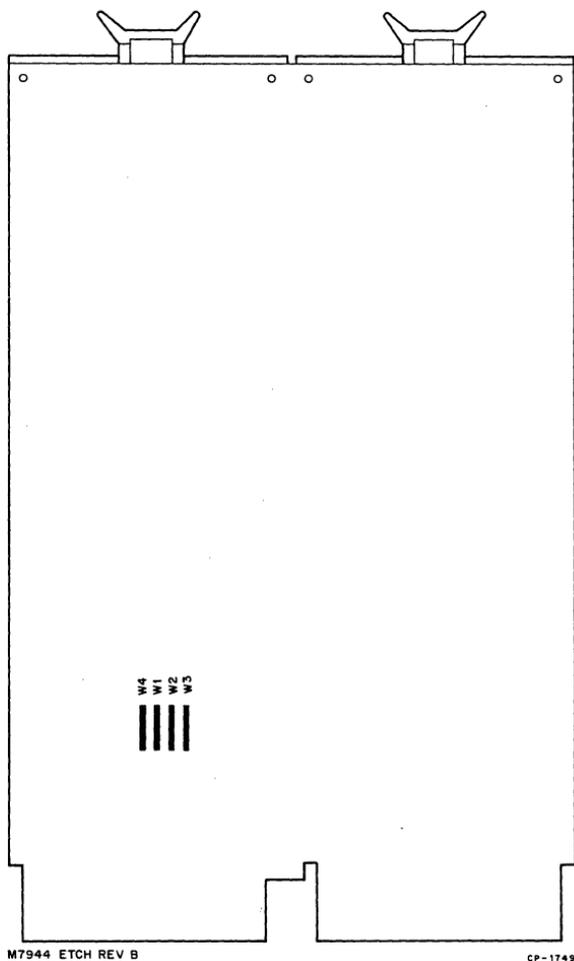


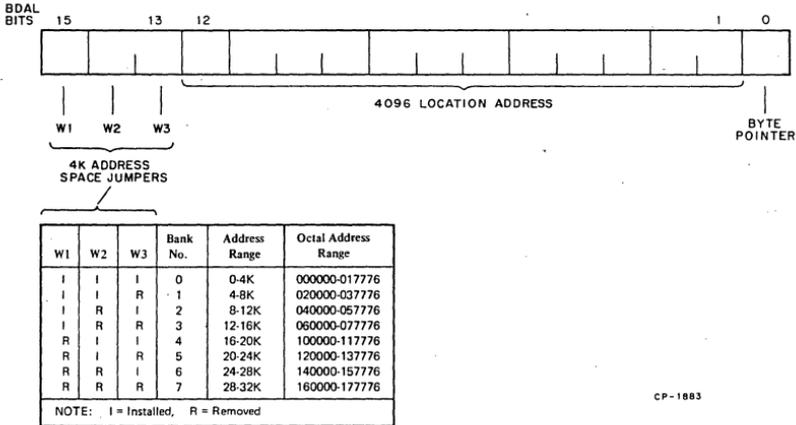
Figure 5-5 MSV11-B Jumper Locations

## 5.4 MMV11-A CORE MEMORY

### 5.4.1 General

The MMV11-A core memory option comprises two modules (G653 and H223) that are mated by connector pins in a single 8.5 x 10 by 0.9 assembly. It requires two device locations (electrical positions) on the backplane when installed in H9270 slots A4-D4; otherwise, because of its total thickness (0.9 in.), the MMV11-A requires four physical device locations when installed in any other backplane slot. (Refer to Paragraph 6.3.3 for installation considerations.) Memory capacity is 4096 16-bit

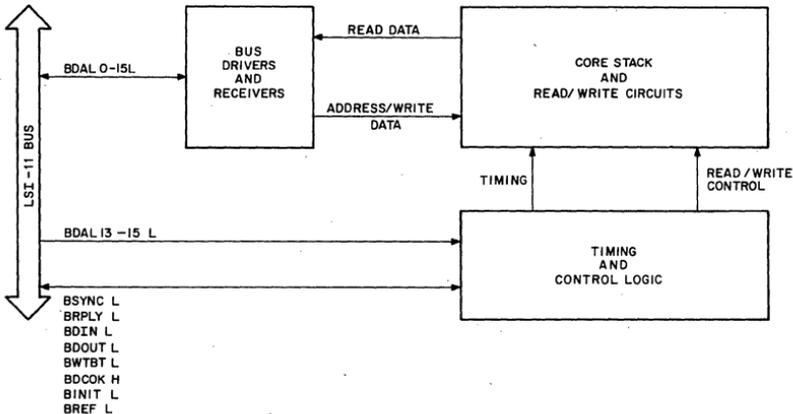
words. Switches select the 4K bank address to which the MMV11-A will respond.



CP-1883

Figure 5-6 MSV11-B Address Format/Jumpers

The MMV11-A is fully LSI-11 bus-compatible and can be accessed by the LSI-11 microcomputer or any DMA device that becomes bus master. It interfaces with the bus as shown in Figure 5-7.



CP-1752

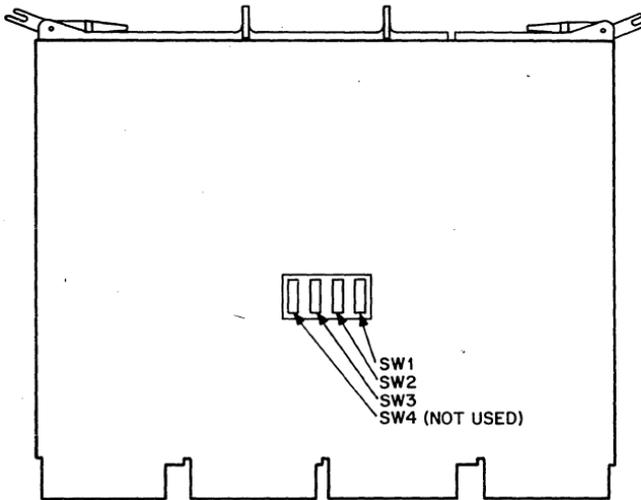
Figure 5-7 MMV11-A 4K by 16-Bit Core Memory

#### 5.4.2 Switch-Selected Addressing

The only preparation required for the MMV11-A before it is installed in the backplane is to select its bank address. This is accomplished

by opening or closing switches in appropriate address bit locations to produce the desired bank address decoding.

MMV11-A bank address switches are used as shown in Figure 5-9. The figure illustrates a 16-bit address and how switches are assigned to each address bit. Open or close switches to produce the desired bank address as directed in the figure. Switches are located on the G653 module (component side) as shown in Figure 5-8.



CP - 1754

Figure 5-8 Bank Address Switch Locations

### 5.4.3 Backplane Jumpers

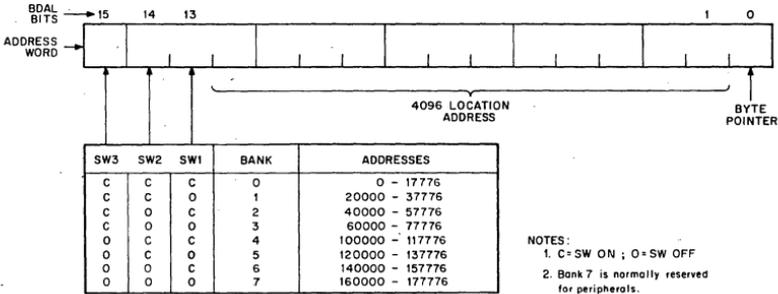
The BDMGI L and BIAKI L bus lines must be jumpered to BDMGO L and BIAKO L lines, respectively, under the H223 module when installed between the processor and I/O device interface modules in order to maintain daisy-chain signal continuity.

Pins which must be connected are:

| From  | To    | Signal   |
|-------|-------|----------|
| C0IN2 | C04M2 | BIAKI/OL |
| C01S2 | C04R2 | BDMGI/OL |

Bus pins can be identified as shown in Figures 3-2 and 3-3.

Memory refresh is not required for this memory option. If memory refresh is used for other memory options, such as the KD11-F's resident memory and the MSV11-B semiconductor memory, the MMV11-A will not respond to the refresh operation.



CP - 1753

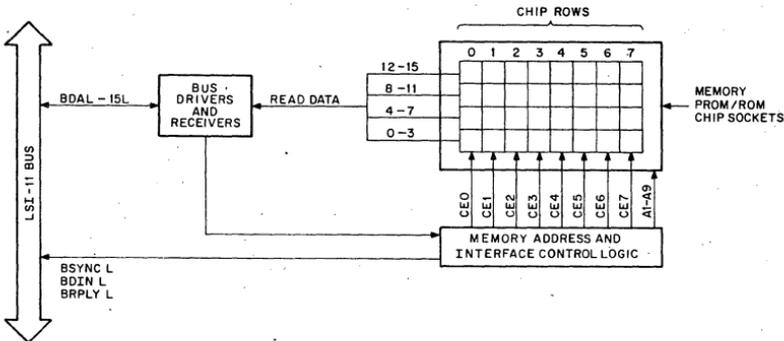
Figure 5-9 MMV11-A Addressing

## 5.5 MRV11-AA READ-ONLY MEMORY

### 5.5.1 General

The MRV11-AA (Figure 5-10) is a read-only memory module that allows the use of user-supplied, preprogrammed, programmable read-only memory (PROM) and masked read-only memory (ROM) chips in a compact, nonvolatile memory subsystem. Depending on chip type, the module's capacity is either 4096 16-bit words or 2048 16-bit words, using 512 by 4-bit or 256 by 4-bit chips, respectively. Full address decoding is provided on the module. The user can select the 4K address bank in which the module resides by installing (or removing) jumpers on the module. Similarly, when using 256 by 4-bit chips, the user can jumper-select the upper or lower 2K segment within the selected 4K address bank. Note that 512 by 4-bit and 256 by 4-bit chips cannot be mixed on a MRV11-AA module; the user configures jumpers on the module for the chip type being used.

A partial listing of manufacturer's chips that will operate in the MRV11-AA is given in Table 5-4.



CP - 1755

Figure 5-10 MRV11-AA Read-Only Memory

**Table 5-4 MRV11-AA Chips**

| <b>Manufacturer or Source</b> | <b>512 by 4-Bit Chips</b> | <b>256 by 4-Bit Chips</b> |
|-------------------------------|---------------------------|---------------------------|
| Digital Equipment Corp.       | MRV11-AC                  | —                         |
| Intersil                      | IM5624                    | IM5623                    |
| Signetics                     | 82S131                    | 82S129                    |
| MMI                           | 6306                      | 6301                      |

Chips used must be tristate output devices that conform to the device pinning, data, and addressing described in the remainder of this chapter.

The user can install chips in increments of four chips each. When using 512 by 4-bit chips, memory expansion is in 512-word increments. When using 256 by 4-bit chips, memory expansion is in 256-word increments. Jumpers on the MRV11-AA can be cut by the user to prevent an incorrect BRPLY L signal from being generated when unpopulated locations are addressed on the module.

The information contained in the remainder of this chapter will enable the user to prepare the MRV11-AA for use (jumper-selected addressing and chip selection) and includes information required for correct PROM and ROM programming.

### **5.5.2 Chip Type Jumpers**

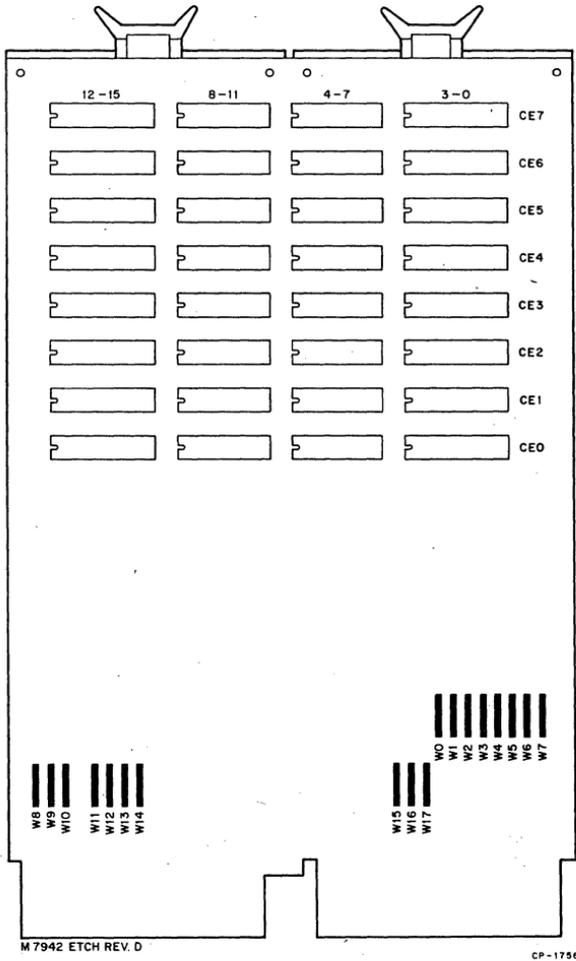
The module is supplied with jumpers W8, W9, and W10 installed for use with 512 by 4-bit chips. When using 256 by 4-bit chips, W8, W9, and W10 must be cut or removed and jumpers W11 and W12 installed; in addition, either W13 (lower 2K) or W14 (upper 2K) must be installed to properly address the lower 2K or upper 2K address segment within the 4K memory bank. Jumpers are located as shown in Figure 5-11.

### **5.5.3 Address and Reply Jumpers**

The user must consider both 4K bank address selection and BRPLY L signal generation when configuring a module for use. Chips (either PROM or ROM, 512 by 4 or 256 by 4) are arranged in eight physical rows (CE0-CE7) of four chips each. Entire rows can be unpopulated, allowing those addressed locations to be used by read/write memory contained on another module. When this is done, the BRPLY L jumpers (W0-W7) associated with the unused rows should be cut or removed to prevent the MRV11-AA from returning a BRPLY L signal when those rows are addressed. A listing of octal addresses (within a 4K bank), physical rows, and BRPLY L jumpers is provided in Table 5-5; use data listed for the chip type being used.

The 4K bank in which the MRV11-AA resides is programmed by connecting bank address jumpers W15-W17, as appropriate. The module is supplied with all bank address jumpers installed (bank 0). Jumpers installed represent logical 0s; jumpers not installed represent logical 1s.

Figure 5-12 illustrates addressing words used with the MRV11-AA. Refer to the addressing format for the type of PROM or ROM chips being used.



M7942 ETCH REV. D

CP-1756

Figure 5-11 MRV11-AA Jumper Locations

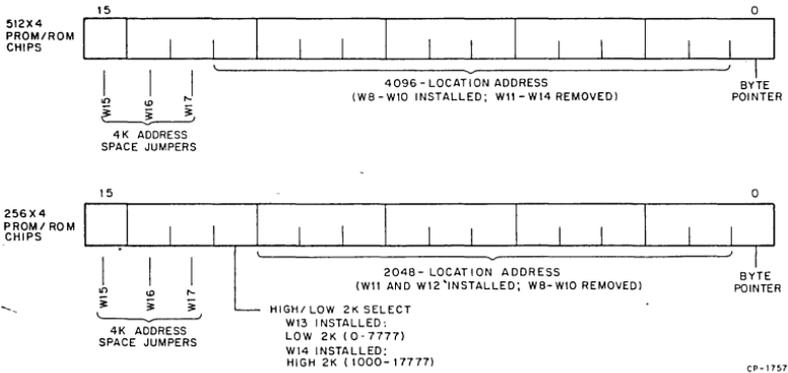


Figure 5-12 MRV11-AA Address Word Formats

### 5.5.4 PROM Chips

The actual procedure for loading data into PROM chips or writing specifications for masked ROM chips will vary, depending on the chip manufacturer. Those procedures are beyond the scope of this document. (See chip manufacturer's data sheets.) However, the user must be aware of the chip pins versus LSI-11 data bit relationship, and the chip pins versus memory address bits. Address and data pins are described below.

As previously discussed, chips are arranged in rows of four chips each. Each chip contains locations of four bits each. Hence, four chips are used to provide the 16-bit data word formats for each row. Rows are designated by their respective Chip Enable (CE0-CE7) signals. Depending upon the chip type used, a row of four chips contains 512 or 256 16-bit read-only memory locations. The actual chip within a row is designated by one additional digit (0, 1, 2, or 3). Hence, the data pins are assigned to LSI-11 bus bits as listed in Table 5-6.

Table 5-6 Data Pin Assignments

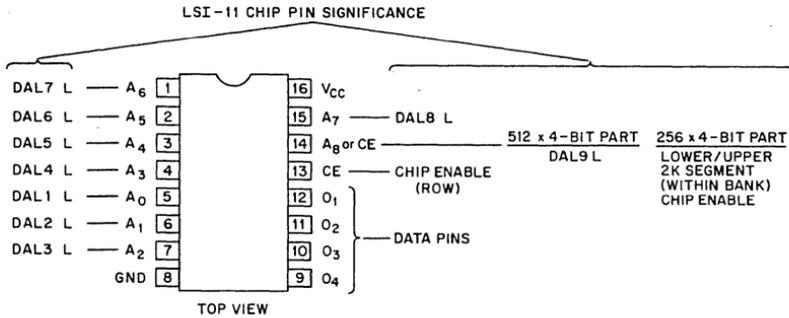
| Chip Pin | Chip 0 | Chip 1 | Chip 2 | Chip 3 |
|----------|--------|--------|--------|--------|
| 9        | BDAL3  | BDAL7  | BDAL11 | BDAL15 |
| 10       | BDAL2  | BDAL6  | BDAL10 | BDAL14 |
| 11       | BDAL1  | BDAL5  | BDAL9  | BDAL13 |
| 12       | BDAL0  | BDAL4  | BDAL8  | BDAL12 |

Addressing of chips is shown in Figure 5-13. All chips used on the MRV11-AA must conform to this information. Observe that the only difference between 512 by 4-bit and 256 by 4-bit chip pins is pin 14. The

Table 5-5 PROM/ROM Chip Addressing Data

| Bank Addr.<br>Jumpers* |     |     | 512 by 4 Bit Chips   |                 |                   | 256 by 4 Bit Chips |               |                 |                   |
|------------------------|-----|-----|----------------------|-----------------|-------------------|--------------------|---------------|-----------------|-------------------|
|                        |     |     | Word/Byte<br>Address | Physical<br>Row | BRPLY L<br>Jumper | Word/Byte Address  |               | Physical<br>Row | BRPLY L<br>Jumper |
| W15                    | W16 | W17 |                      |                 |                   | W13 Installed      | W14 Installed |                 |                   |
| I                      | I   | I   | 0-1777               | CE0             | W0                | 0-777              | 10000-10777   | CE0             | W0                |
| I                      | I   | R   | 2000-3777            | CE1             | W1                | 1000-1777          | 11000-11777   | CE2             | W2                |
| I                      | R   | I   | 4000-5777            | CE2             | W2                | 2000-2777          | 12000-12777   | CE4             | W4                |
| I                      | R   | R   | 6000-7777            | CE3             | W3                | 3000-3777          | 13000-13777   | CE6             | W6                |
| R                      | I   | I   | 10000-11777          | CE4             | W4                | 4000-4777          | 14000-14777   | CE1             | W1                |
| R                      | I   | R   | 12000-13777          | CE5             | W5                | 5000-5777          | 15000-15777   | CE3             | W3                |
| R                      | R   | I   | 14000-15777          | CE6             | W6                | 6000-6777          | 16000-16777   | CE5             | W5                |
| R                      | R   | R   | 16000-17777          | CE7             | W7                | 7000-7777          | 17000-17777   | CE7             | W7                |

\* R = Jumper/removed: I = Jumper installed



**NOTE:**

Designations immediately adjacent to pins are typical designations used by chip manufacturers — not LSI-11 designations. LSI-11 designations for correct addressing are located away from the chip. Observe that these signals are low — active; they are double-inverted bus signals (low = logical "1").

IC - 0169

Figure 5-13 PROM/ROM Chip Pin Addressing

512 by 4-bit part uses this pin for address bit DAL9; the 256 by 4-bit part uses this pin for a chip enable when both bank address and 2K segment address are true. Also note that bus address bits do not follow in sequence with chip manufacturer's address designations. The pinning arrangement shown allows for the use of commonly available PROM and ROM chips and optimum (compact) MRV11-AA module layout.

**5.5.5 Programming PROM Chips**

Complete information for programming PROM chips is contained in Chapter 7. Do not attempt to program PROMs until you are thoroughly familiar with the information contained in that chapter.

**5.5.6 I/O Timing and Bus Restrictions**

Addressed memory read data is available within 120 ns after the BSYNC L signal is received by the MRV11-AA. Logic on the module responds to DAT1 bus cycles only. DAT0 or DATOB bus cycles will result in a bus time-out error. Logic functions on the module are not affected by the bus initialize (BINIT L) signal.

**5.6 DLV11 SERIAL LINE UNIT**

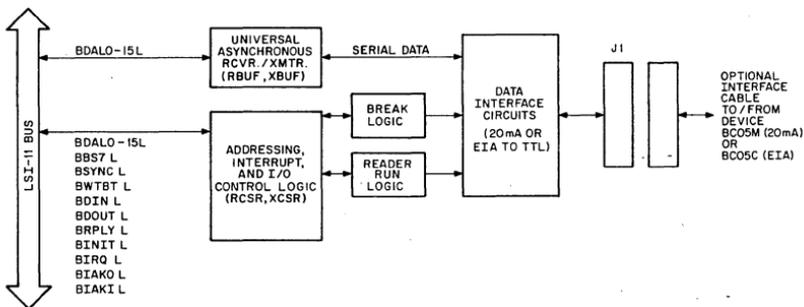
**5.6.1 General**

The DLV11 Serial Line Unit (SLU) interfaces serial I/O devices to the LSI-11 bus, as shown in Figure 5-14.

**5.6.2 Jumper-Selected Addressing, Vectors, and Module Operations**

**5.6.2.1 General**—As shown in Figure 5-15, the DLV11 SLU module is equipped with thirty jumpers that can be configured to satisfy the operating requirements. A DLV11 module is configured at the factory to serve as a console SLU. This configuration is summarized in Table 5-7.

**5.6.2.2 Addressing**—Jumpers involved with addressing include A3 through A12. Only address bits 03 through 12 are programmed by the jumpers for correct DLV11 addressing, producing the 16-bit address word shown in Figure 5-16. The appropriate jumpers are removed to produce logical 1 bits; jumpers installed will produce logical 0 bits.



CP-1800

Figure 5-14 DLV11 Serial Line Unit

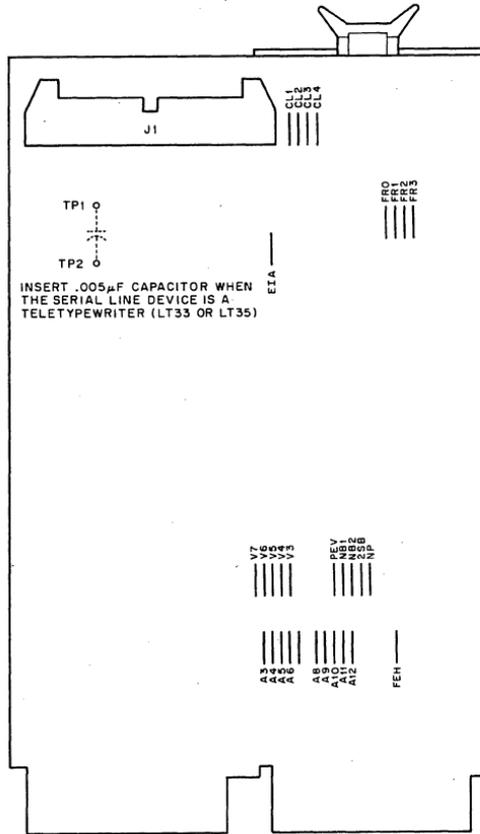
Table 5-7 DLV11 SLU Factory Jumper Configuration

| Jumper Designation | Jumper State | Function Implemented  |   |
|--------------------|--------------|---|---|
| A3                 | I            | This arrangement of jumpers A3 through A12 implements the octal device address 17756X, which is the assigned address for the console device SLU. The least significant digit is hard-wired on the module to address the four SLU device registers as follows: |   |
| A4                 | R            |   |   |
| A5                 | R            |   |   |
| A6                 | R            |   |   |
| A7                 | I            |   |   |
| A8                 | R            |   |   |
| A9                 | R            |   |   |
| A11                | R            |   | X = 0, RCSR address   |
| A10                | R            |   | X = 2, Receive data register address  |
| A12                | R            |   | X = 4, XCSR address   |
|                    |              |   | X = 6, Transmit data register address   |
| V3                 | I            |   | This jumper arrangement implements the interrupt vector addresses 60 for received data and 64 for transmitted data. |
| V4                 | R            |   |   |
| V5                 | R            |   |   |
| V6                 | I            |   |   |
| V7                 | I            |   |   |
| NP                 | R            | No parity   |   |
| 2SB                | R            | Two stop bits   |   |
| NB2                | R            | Eight data bits   |   |
| NB1                | R            |   |   |
| PEV                | R            | Even parity if NP installed   |   |
| FEH                | I            | Halt on framing error   |   |
| EIA                | R            | 12 V EIA operation disabled   |   |

**Table 5-7 DLV11 SLU Factory Jumper Configuration (Cont.)**

| Jumper Designation | Jumper State | Function Implemented  |
|--------------------|--------------|---|
| FR0                | R            | 110 Baud rate selected                                      |
| FR1                | R            |   |
| FR2                | R            |   |
| FR3                | R            |   |
| CL1                | I            | 20 mA current loop active receiver and transmitter selected |
| CL2                | I            |   |
| CL3                | I            |   |
| CL4                | I            |   |

Note: R = removed, I = installed



CP-1801

**Figure 5-15 DLV11 Jumper Locations**

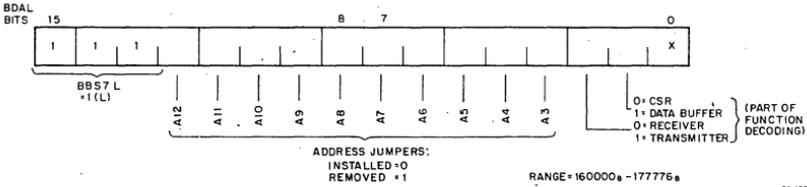


Figure 5-16 DLV11 Addresses

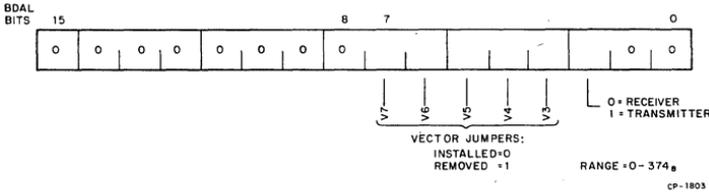


Figure 5-17 DLV11 Interrupt Vectors

**5.6.2.3 Vectors**—Jumpers involved with vector addressing include V3 through V7. Only vector bits 03 through 07 are programmed by the jumpers for correct DLV11 vector addressing, producing the 16-bit address shown in Figure 5-17. The appropriate jumpers are removed to produce logical 1 bits; jumpers installed will produce logical 0 bits.

**5.6.2.4 UAR/T Operation**—UAR/T operation is programmed via jumpers NP, 2SB, NB1, NB2, and PEV as shown below.

**Number of Data Bits**

|   | NB1       | NB2       |
|---|-----------|-----------|
| 5 | Installed | Installed |
| 6 | Removed   | Installed |
| 7 | Installed | Removed   |
| 8 | Removed   | Removed   |

**Number of Stop Bits Transmitted**

- 2SB installed = One stop bit
- 2SB removed = Two stop bits

**Parity Transmitted**

- NP removed = No parity bit
- NP and PEV installed = Odd parity
- NP installed and PEV removed = Even parity

**5.6.2.5 Baud Rate Selection**—Baud rate is programmed via jumpers FR0 through FR3 as shown in Table 5-8.

**Table 5-8 Baud Rate Selection**

| Baud Rate                 | FR3 | FR2 | FR1 | FR0 |
|---------------------------|-----|-----|-----|-----|
| 50                        | I   | I   | R   | I   |
| 75                        | I   | I   | R   | R   |
| 110                       | R   | R   | R   | R   |
| 134.5                     | I   | R   | I   | I   |
| 150                       | R   | R   | R   | I   |
| 200                       | I   | R   | I   | R   |
| 300                       | R   | R   | I   | R   |
| 600                       | I   | R   | R   | I   |
| 1200                      | R   | I   | R   | R   |
| 1800                      | R   | I   | R   | I   |
| 2400                      | I   | R   | R   | R   |
| 2400                      | R   | R   | I   | I   |
| 4800                      | R   | I   | I   | R   |
| 9600                      | R   | I   | I   | I   |
| External<br>(via pin BH1) | I   | I   | I   | X   |

NOTE:

I = installed                      X = don't care  
R = removed

**5.6.2.6 EIA Interface**—EIA drivers are enabled when jumper EIA is installed. This jumper applies  $-12$  V to the EIA driver chip. It should be removed during 20 mA current loop operation.

**5.6.2.7 20 mA Current Loop Interface**—Jumpers CL1 through CL4 are associated with 20 mA current loop interface operation. Remove EIA and remove or install jumpers as desired for the functions listed below:

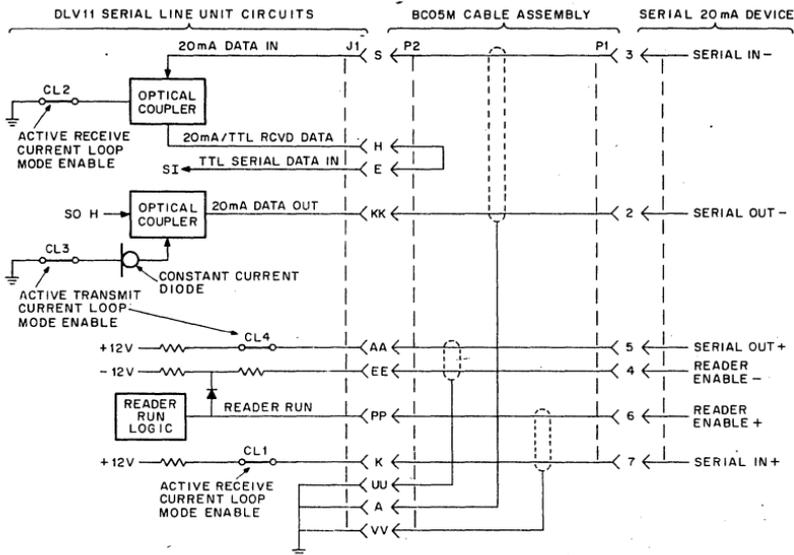
**Active Current Loop** (Jumper configuration is shown in Figures 5-18 and 5-19.)

Transmit = CL3 and CL4 installed  
Receive = CL1 and CL2 installed

**Passive Current Loop** (Jumpers configured as shown in Figures 5-20 and 5-21.)

Transmit = CL3 and CL4 removed  
Receive = CL1 and CL2 removed

The DLV11 is supplied with jumpers CL1 through CL4 wired for the active transmit, active receive mode (Figure 5-18). When in this mode, serial current limiting to 23 mA is provided by resistors (one each for transmit and receive functions) connected to the  $+12$  V source. Note that when module power is removed, the 20 mA transmit optical coupler closes the serial loop (active or passive mode). When the DLV11 is used in the passive 20 mA mode (Figure 5-20), the serial device must produce the 20 mA current. Current limiting must be provided for transmit and receive currents in the serial device.



CP-1804

Figure 5-18 Active 20 mA Current Loop Interface

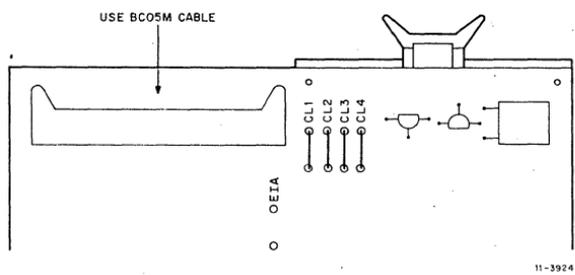
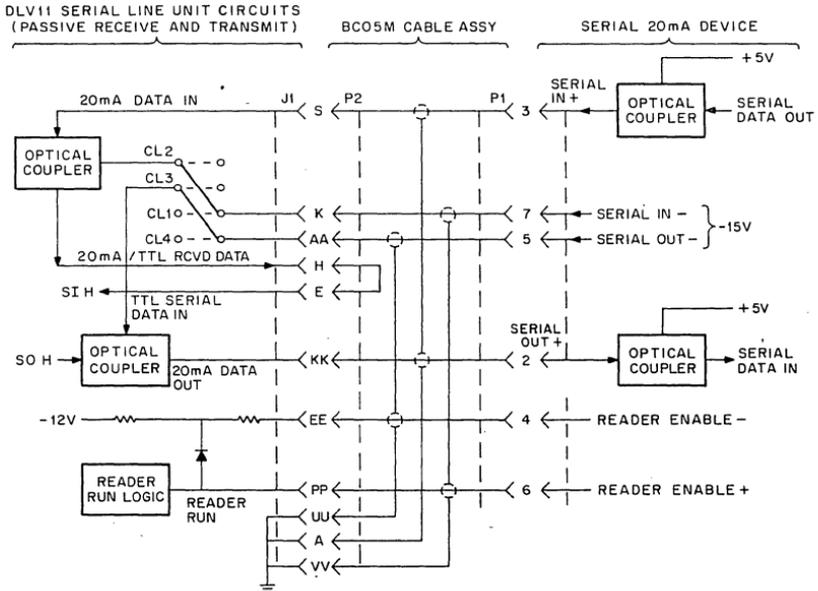


Figure 5-19 20 mA Active Current Loop Jumper Configuration

**5.6.2.8 Framing Error Halt**—A framing error halt allows entry to console microcode directly from the console device by pressing the BREAK key, producing a framing error. A framing error occurs when the received character has no valid stop bit. This error condition is detected by the UAR/T. FEH is factory-installed, causing the assertion of BHALT L when the framing error is detected. The processor then executes console microcode.

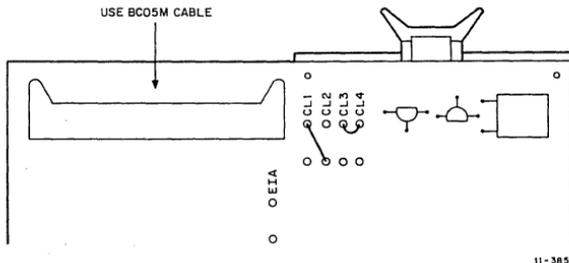
**5.6.3 Installation**

Prior to installing the DLV11 on the backplane, first establish the desired priority level (Chapter 3) to determine the backplane slot in which the module will be installed. Then, check that jumpers are removed or installed as described for your application (Paragraph 5.6.2). Connection



CP-1805

Figure 5-20 Passive 20 mA Loop Interface



11-3851

Figure 5-21 20 mA Passive Current Loop Jumper Configuration

to the peripheral device is via an optional data interface cable. Cables are listed below.

| Application        | Cable Type*            |
|--------------------|------------------------|
| EIA Interface      | BC05C-X Modem Cable    |
| 20 mA Current Loop | BC05M-X Cable Assembly |

\* The -X in the cable number denotes length in feet, as follows: -1, -6, -10, -20, -25. For example, a 10-ft EIA interface cable would be ordered as BC05C-10.

### 5.6.4 Interfacing with 20 mA Current Loop Devices

When interfacing with 20 mA current loop devices, the BC05M cable assembly provides the correct connections to the 40-pin connector on the DLV11. The peripheral device end of the cable is terminated with a Mate-N-Lok connector that is pin-compatible with the following peripheral options:

- LA36 DECwriter
- LT33 Teletypewriter
- LT35 Teletypewriter
- VT05B Alphanumeric Terminal
- VT50 DECscope
- VT52 DECscope
- RT02 Alphanumeric Terminals
- DF01-A Acoustic Telephone Coupler

The complete interface circuit provided by the BC05M cable and the associated DLV11 jumpers is shown in Figure 5-18.

#### NOTE

When the DLV11 is used with teletypewriter devices, a 0.005  $\mu$ F capacitor must be installed between split lugs TP1 and TP2.

After configuring the module jumpers and installing the proper interface cable, the DLV11 can be installed in the backplane.

### 5.6.5 Interfacing with EIA-Compatible Devices

When interfacing with EIA devices, the BC05C modem cable provides the correct connection to the 40-pin connector on the DLV11. The peripheral device end of the cable is terminated with a Cinch DB25P connector that is pin-compatible with Bell 103, 113 modems. Connector pinning and signal levels conform to EIA Specification RS232C. The complete EIA interface circuit is shown in Figure 5-22; jumpers are shown in Figure 5-23.

### 5.6.6 Programming

**5.6.6.1 Addressing**—Addresses for the DLV11 can range from 160000 through 17777X<sub>6</sub>. The least significant three bits (only bits 01 and 02 are used; bit 0 is ignored) address the desired register in the DLV11, as follows:

| Address | Addressed Register             |
|---------|--------------------------------|
| 1XXX0   | RCSR (Receiver control/status) |
| 1XXX2   | RBUF (Receiver data buffer)    |
| 1XXX4   | XCSR (Transmit control/status) |
| 1XXX6   | XBUF (Transmit data buffer)    |

Address bits 03 through 12 are jumper-selected as directed in Paragraph 5.6.2.2.

Since each DLV11 module has four registers, each requires four addresses. Addresses 177560—177566 are reserved for the DLV11 used with the console peripheral device. Additional DLV11 modules should be assigned addresses from 175610 through 176176, allowing up to 30 additional DLV11 modules to be addressed.

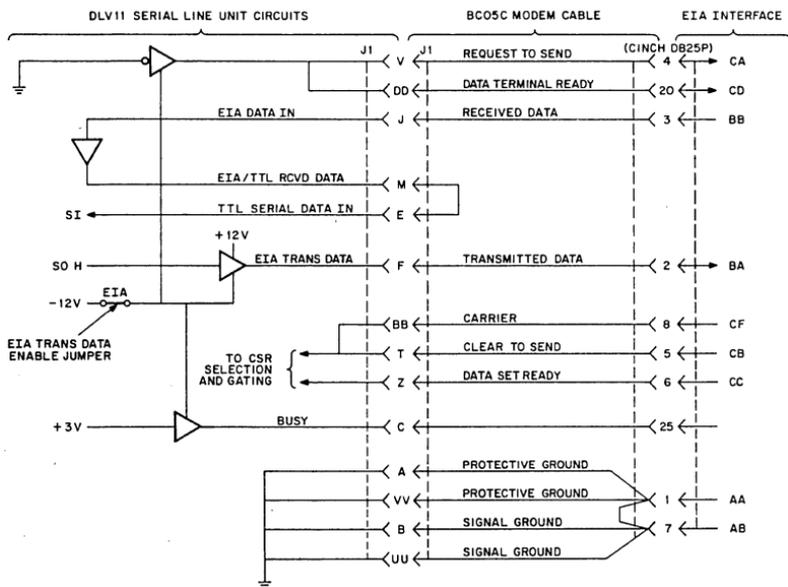


Figure 5-22 EIA Interface

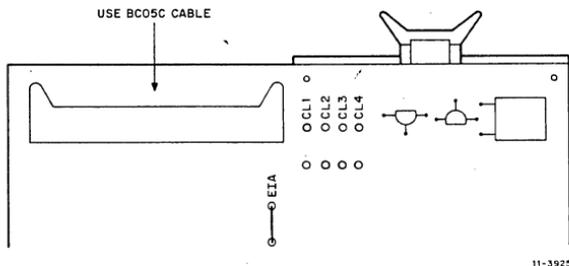


Figure 5-23 EIA Jumper Configuration

**5.6.6.2 Interrupt Vectors**—Two interrupt vectors are jumper-selected on each DLV11 as described in Paragraph 5.6.2.3:

- 000XX0 Receiver interrupt vector
- 000XX4 Transmitter interrupt vector

Vectors can range from addresses 0 through 37X<sub>8</sub>. Vectors 60 and 64 are reserved for the console peripheral device. Additional DLV11 modules should be assigned vectors following any DRV11 modules installed in the system starting at address 300.

**5.6.6.3 Word Formats**—The four word formats associated with the DLV11 are shown in Figure 5-24 and are described in Table 5-9.

**5.6.7 Console Device**

The console device is a serial line device, such as the LA36 DECwriter, that uses a DLV11 Serial Line Unit. The following device addresses must be used for the console device:

| Register | Address |
|----------|---------|
| RCSR     | 177560  |
| RBUF     | 177562  |
| XCSR     | 177564  |
| XBUF     | 177566  |

Vector addresses must be assigned as follows:

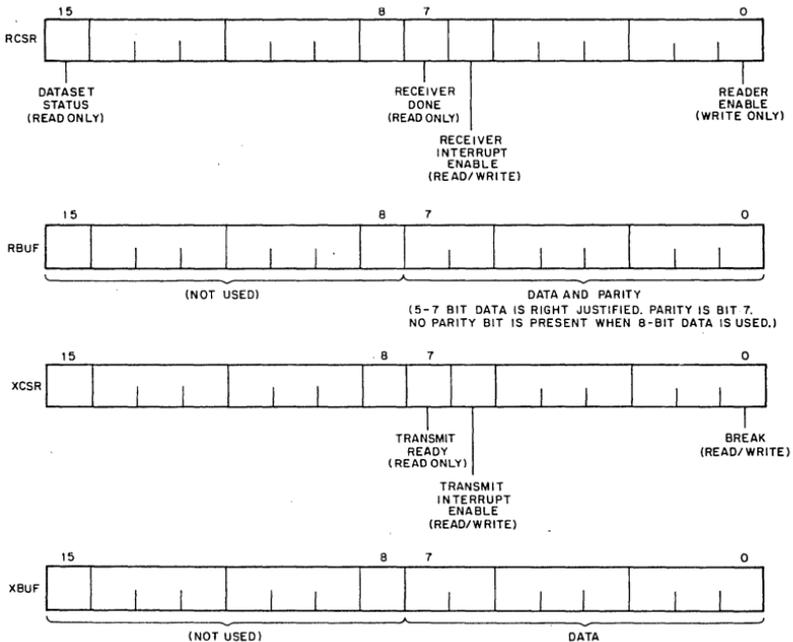
| Interrupt Vector    | Address |
|---------------------|---------|
| Console Receiver    | 000060  |
| Console Transmitter | 000064  |

**Table 5-9 Word Formats**

| Word | Bit(s) | Function  |
|------|--------|---|
| RCSR | 15     | Dataset Status—Set when CARRIER or CLEAR TO SEND and DATA SET READY signals are asserted by an EIA device. Read-only bit.   |
|      | 14—08  | Not used. Read as 0.  |
|      | 07     | Receiver Done—Set when an entire character has been received and is ready for input to the processor. This bit is automatically cleared when RBUF is addressed or when the BDCOK H signal goes false (low). A receiver interrupt is enabled by the DLV11 when this bit is set and receiver interrupt is enabled (bit 6 is also set). Read-only bit. |
|      | 06     | Interrupt Enable—Set under program control when it is desired to generate a receiver interrupt request when a character is ready for input to the processor (bit 7 is set). Cleared under program control or by the BINIT signal. Read/write bit.   |
|      | 05—01  | Not used. Read as 0.  |
|      | 00     | Reader Enable—Set by program control to advance the paper tape reader on a teletypewriter device to input a new character. Automatically cleared by the new character's start bit. Write-only bit.  |
| RBUF | 15—08  | Not used. Read as 0.  |
|      | 07—00  | Contains five to eight data bits in a right-justified format. MSB is the optional parity bit. Read-only bit.  |
| XCSR | 15—08  | Not used. Read as 0.  |
|      | 07     | Transmit Ready—Set when XBUF is empty and can   |

**Table 5-9 Word Formats (Cont.)**

| Word | Bit(s) | Function  |
|------|--------|---|
|      |        | accept another character for transmission. It is also set during the power-up sequence by the BDCOK H signal. Automatically cleared when XBUF is loaded. When transmitter interrupt is enabled (bit 6 also set), an interrupt request is asserted by the DLV11 when this bit is set. Read-only bit. |
|      | 06     | Interrupt Enable—Set under program control when it is desired to generate a transmitter interrupt request when the DLV11 is ready to accept a character for transmission. Reset under program control or by the BINIT signal. Read/write bit.   |
|      | 05—01  | Not used. Read as 0.  |
|      | 00     | Break—Set or reset under program control. When set, a continuous space level is transmitted. BINIT resets this bit. Read/write bit.   |
| XBUF | 15—08  | Not used.   |
|      | 07—00  | Contains five to eight right-justified data bits. Loaded under program control for serial transmission to a device. Write only.   |



**Figure 5-24 DLV11 Word Formats**

CP-1807

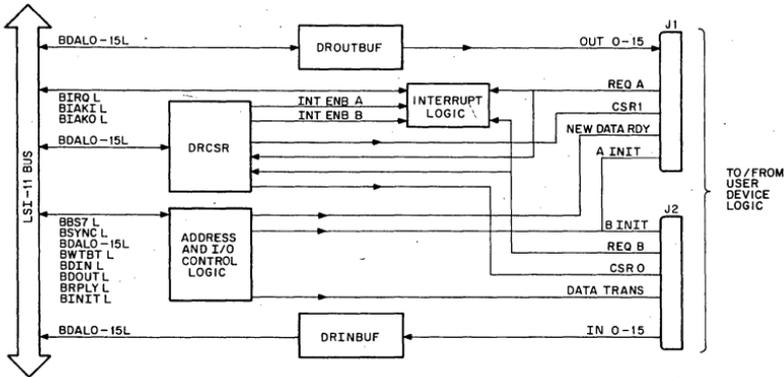
## 5.7 DRV11 PARALLEL LINE UNIT

### 5.7.1 General

The DRV11 Parallel Line Unit (PLU) is a general-purpose device interface module that connects parallel I/O devices to the LSI-11 bus, as shown in Figure 5-25.

### 5.7.2 Jumper-Selected Addressing and Vectors

The DRV11 Parallel Line (PLU) module is equipped with 15 jumpers that can be configured to select device and interrupt vector addresses. In addition, the board is equipped with a set of split lugs for installing an optional capacitor to adjust certain signal pulse widths that control interfacing with external devices. The location of these jumpers and the split lugs for the optional capacitor are shown in Figure 5-26. This unit is jumper configured to implement recommended device and vector addresses, and is not equipped with the split lug-mounted capacitor. The factory-installed jumper configuration is summarized in Table 5-10.



CP-1808

Figure 5-25 DRV11 Parallel Line Unit

Table 5-10 DRV11 PLU Factory Jumper Configuration

| Jumper Designation | Jumper State | Function Implemented  |
|--------------------|--------------|---|
| A3                 | R            | This arrangement of jumpers A3 through A12 assigns the device address 16777X to the PLU. This address is the starting address of a reserved block in memory bank 7 which is recommended for user device address assignments. The least significant digit X is hardwired on the module to implement the three PLU device addresses as follows: |
| A4                 | R            |   |
| A5                 | R            |   |
| A6                 | R            |   |
| A7                 | R            |   |
| A8                 | R            |   |
| A9                 | R            |   |
| A10                | R            |   |
| A11                | R            |   |
| A12                | I            |   |

**Table 5-10 DRV11 PLU Factory Jumper Configuration (Cont.)**

| Jumper Designation | Jumper State | Function Implemented  |
|--------------------|--------------|---|
|                    |              | X = 0 DRCSR address   |
|                    |              | X = 2 Output buffer address   |
|                    |              | X = 4 Input buffer address  |
| V3                 | I            | This factory installed jumper configuration implements the two interrupt vector addresses 300 and 304 for use as defined by application requirements. |
| V4                 | I            |   |
| V5                 | I            |   |
| V6                 | R            |   |
| V7                 | R            |   |

NOTE: R = Removed, I = Installed

**5.7.2.1 Locations**—Jumpers for device address and vector selection are provided on the DRV11 as shown in Figure 6-10. Factory installed jumpers can be cut or removed by the user to program the module for a particular system application, as described in the following paragraphs.

**5.7.2.2 Addressing**—Jumpers involved with addressing include A3 through A12. Only address bits 03 through 12 are programmed by jumpers for DRV11 addressing producing the 16-bit address word shown in Figure 5-27. The appropriate jumpers are removed to produce logical 1 bits; jumpers installed will produce logical 0 bits.

**5.7.2.3 Vectors**—Jumpers involved with vector addressing include V3 through V7. Only vector bits 03 through 07 are programmed by the jumpers for DRV11 vector addressing, producing the 16-bit word shown in Figure 5-28. The appropriate jumpers are removed to produce logical 1 bits; jumpers installed will produce logical 0 bits.

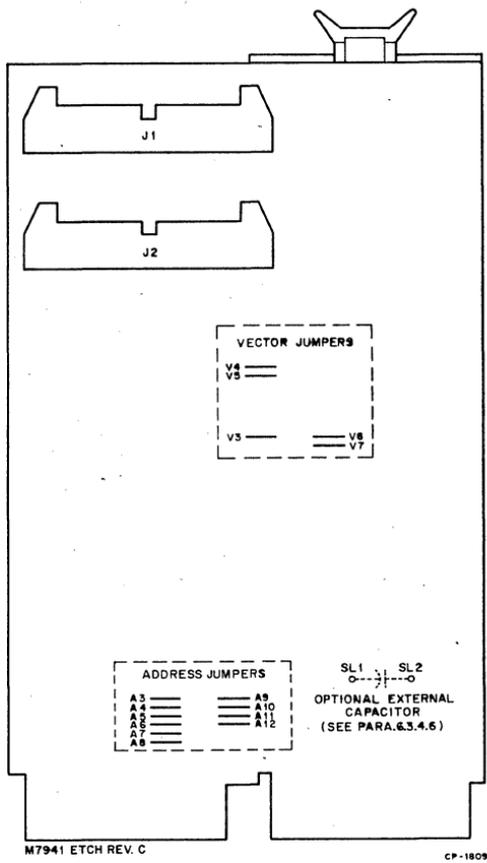


Figure 5-26 DRV11 Jumper Locations

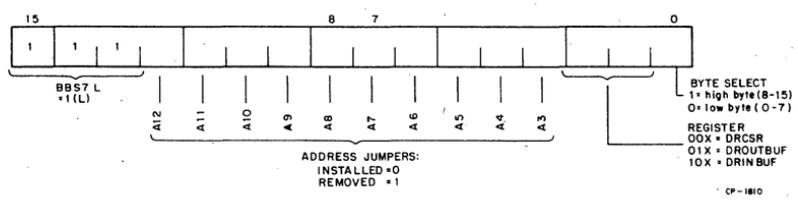


Figure 5-27 DRV11 Device Address

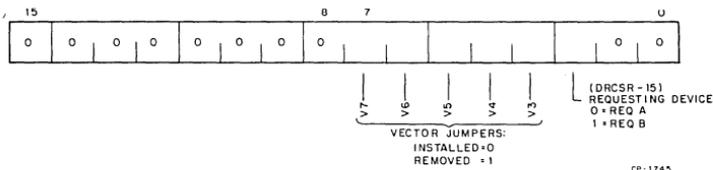


Figure 5-28 DRV11 Vector Address

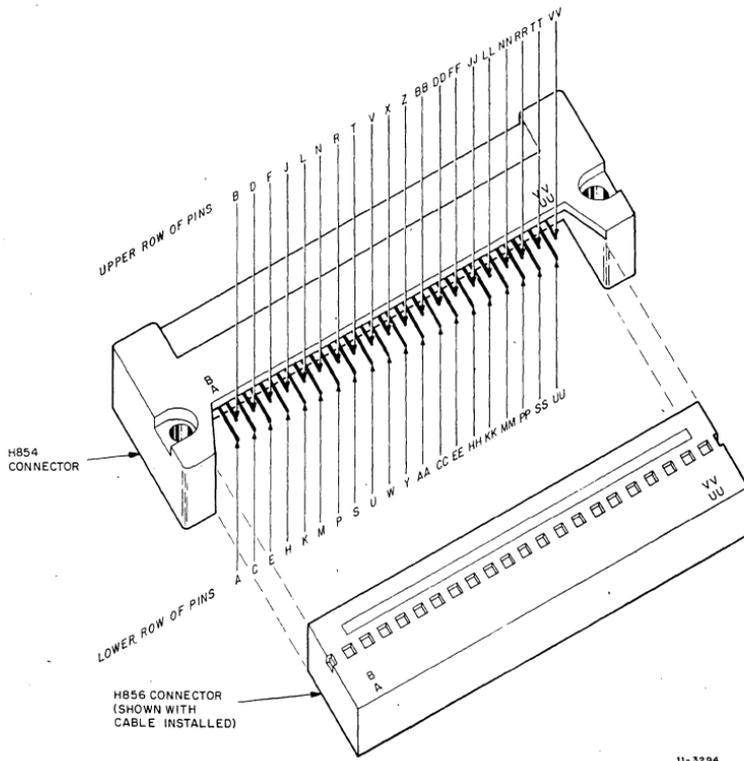


Figure 5-29 J1 or J2 Connector Pin Locations

### 5.7.3 Installation

Prior to installing the DRV11 on the backplane, first establish the desired priority level (Chapter 3) for the backplane slot installation. Check that proper device address vector jumpers are installed, as directed in Paragraph 5.7.2. The DRV11 can then be installed on the backplane. Connection to the user's device is via optional cables.

## 5.7.4 Interfacing to the User's Device

**5.7.4.1 General**—Interfacing the DRV11 to the user's device is via the two board-mounted H854 40-pin male connectors. Pins are located as shown in Figure 5-29. Signal pin assignments for input interface J2 (connector No. 2) and output interface J1 (connector No. 1) are listed in Table 5-11. Optional cables and connectors for use with the DRV11 include:

**BC08R-X\***—Maintenance cable, 40-conductor flat with H856 connectors on each end. Available in lengths of 1, 6, 10, 20, and 25 feet.

**BC07D-X\***—Signal cable, two 20 conductor ribbon cables with a single H856 connector on one end; remaining end is terminated by the user. Available in lengths of 10, 15, and 25 feet.

**H856**—Socket, 40-pin female, for user-fabricated cables.

\* The -X in the cable number denotes length in feet. -10, -12, -20. For example, a 10-ft BC07D cable would be ordered as BC07D-10.

When using the BC07D cable, connect the free end of the ribbon cables using the wiring data contained in Table 5-12. Refer to the *Hardware/Accessories Catalog* for additional optional interface accessories.

**5.7.4.2 Output Data Interface**—The output interface is the 16-bit buffer (DROUTBUF). It can be either loaded or read under program control. When loaded by a DATO or DATOB bus cycle, the NEW DATA READY H 750 ns pulse is generated to inform the user's device of the data transfer. The trailing edge of this positive-going pulse should be used to strobe the data into the user's device in order to allow data to settle on the interface cable. The system initialize signal (BINIT L) will clear DROUTBUF.

All output signals are TTL levels capable of driving eight unit loads except for the following:

NEW DATA READY = 10 unit loads

DATA TRANSMITTED = 30 unit loads

INIT (Initialize) = 10 units per connector

**Table 5-11 DRV11 Input and Output Signal Pins**

| Inputs |           |      | Outputs |           |     |
|--------|-----------|------|---------|-----------|-----|
| Signal | Connector | Pin  | Signal  | Connector | Pin |
| IN00   | J2        | TT   | OUT00   | J1        | C   |
| IN01   | J2        | LL   | OUT01   | J1        | K   |
| IN02   | J2        | H, E | OUT02   | J1        | NN  |
| IN03   | J2        | BB   | OUT03   | J1        | U   |
| IN04   | J2        | KK   | OUT04   | J1        | L   |
| IN05   | J2        | HH   | OUT05   | J1        | N   |
| IN06   | J2        | EE   | OUT06   | J1        | R   |
| IN07   | J2        | CC   | OUT07   | J1        | T   |

**Table 5-11 DRV11 Input and Output Signal Pins (Cont.)**

| Inputs |           |     | Outputs       |           |        |
|--------|-----------|-----|---------------|-----------|--------|
| Signal | Connector | Pin | Signal        | Connector | Pin    |
| IN08   | J2        | Z   | OUT08         | J1        | W      |
| IN09   | J2        | Y   | OUT09         | J1        | X      |
| IN10   | J2        | W   | OUT10         | J1        | Z      |
| IN11   | J2        | V   | OUT11         | J1        | AA     |
| IN12   | J2        | U   | OUT12         | J1        | BB     |
| IN13   | J2        | P   | OUT13         | J1        | FF     |
| IN14   | J2        | N   | OUT14         | J1        | HH     |
| IN15   | J2        | M   | OUT15         | J1        | JJ     |
| REQ A  | J1        | LL  | NEW DATA RDY* | J1        | VV     |
| REQ B  | J2        | S   | DATA TRANS*   | J2        | C      |
|        |           |     | CSR0          | J2        | K      |
|        |           |     | CSR1          | J1        | DD     |
|        |           |     | INIT          | J1        | P      |
|        |           |     | INIT          | J2        | RR, NN |

\*Pulse signals, approximately 750 ns wide. Width can be changed by user.

**Table 5-12 BC07D Signal Cable Connections**

| Wire Color | Cable 1 (connector pins B-VV) |              |           | Cable 2 (connector pins A-UU) |           |            |
|------------|-------------------------------|--------------|-----------|-------------------------------|-----------|------------|
|            | Pins                          | J1 Signal    | J2 Signal | Pins                          | J1 Signal | J2 Signal  |
| blk        | B                             | open         | open      | A                             | open      | open       |
| brn        | D                             | open         | open      | C                             | OUT00     | DATA TRANS |
| red        | F                             | open         | open      | E                             | open      | IN02       |
| orn        | J                             | GND          | GND       | H                             | open      | IN02       |
| yel        | L                             | OUT04        | GND       | K                             | OUT01     | CSR0       |
| grn        | N                             | OUT05        | IN14      | M                             | GND       | IN15       |
| blu        | R                             | OUT06        | GND       | P                             | INIT      | IN13       |
| vio        | T                             | OUT07        | GND       | S                             | GND       | REQ B      |
| gry        | V                             | GND          | IN11      | U                             | OUT03     | IN12       |
| wht        | X                             | OUT09        | GND       | W                             | OUT08     | IN10       |
| blk        | Z                             | OUT10        | IN08      | Y                             | GND       | IN09       |
| brn        | BB                            | OUT12        | IN03      | AA                            | OUT11     | GND        |
| red        | DD                            | CSR1         | GND       | CC                            | GND       | IN07       |
| orn        | FF                            | OUT13        | open      | EE                            | GND       | IN06       |
| yel        | JJ                            | OUT15        | GND       | HH                            | OUT14     | IN05       |
| grn        | LL                            | REQ A        | IN01      | KK                            | GND       | IN04       |
| blu        | NN                            | OUT02        | INIT      | MM                            | GND       | GND        |
| vio        | RR                            | OUT02        | INIT      | PP                            | GND       | GND        |
| gry        | TT                            | open         | IN00      | SS                            | GND       | GND        |
| wht        | VV                            | New DATA RDY | open      | UU                            | GND       | GND        |

**5.7.4.3 Input Data Interface**—The input interface is the 16-bit DRINBUF read-only register, comprising gated bus drivers that transfer data from the user's device onto the LSI-11 bus under program control. DRINBUF is not capable of storing data; hence the user must keep input data on the IN lines until read by the LSI-11 microcomputer. When read, the DRV11 generates a positive-going 750 ns DATA TRANSMITTED H pulse which informs the user's device that the data has been accepted. The trailing edge of the pulse indicates that the input transfer has been completed.

All input signals are one standard TTL unit load; inputs are protected by diode clamps to ground and +5 V.

**5.7.4.4 Request Flags**—Two signal lines (REQ A H and REQ B H) can be asserted by the user's device as flags in the DRCSR word. REQ B is available via Connector No. 2, and it can be read in DRCSR bit 15. REQ A is available via Connector No. 1, and it can be read in DRCSR bit 7. Two DRCSR interrupt enable bits, INT ENB A (bit 6) and INT ENB B (bit 5), allow automatic generation of an interrupt request when their respective REQ A or REQ B signals are asserted. Interrupt enable bits can be set or reset under program control.

In a typical application, REQ A and REQ B are generated by Request flop-flops in the user's device. The user's Request flip-flop should be set when servicing is required and cleared by NEW DATA READY or DATA TRANSMITTED when the appropriate data transaction has been completed.

**5.7.4.5 Initialization**—The BINIT L processor-generated initialize signal is applied to DRV11 circuits for interface logic initialization. It is also available to the user's circuits via connectors J1 and J2 as follows:

| Connector/Pin | Signal  |
|---------------|---------|
| J1/P          | AINIT H |
| J2/RR         | BINIT H |
| J2/NN         | BINIT H |

An active BINIT L signal will clear: DROUTBUF data; DRCSR bits 6, 5, 1, 0; bits 16 and 7 (when the maintenance cable is connected); and Interrupt Request and Interrupt Acknowledge flip-flops.

**5.7.4.6 NEW DATA READY and DATA TRANSMITTED Pulse Width Modification**—An optional capacitor can be added by the user to the DRV11 module to extend the pulse width of both the NEW DATA READY and DATA TRANSMITTED pulse widths. The module without external capacitance (as shipped) will produce 750 ns pulses. The capacitor can be added in the location shown in Figure 5-26 to produce the approximate pulse widths listed below.

| Optional External Capacitance ( $\mu$ F) | Approximate Pulse Width (ns) |
|--|------------------------------|
| None                                     | 750                          |
| 0.0047                                   | 1150                         |
| 0.01                                     | 1750                         |
| 0.02                                     | 2650                         |
| 0.03                                     | 3850                         |

**5.7.4.7 BC08R Maintenance Cable**—When using the optional BC08R maintenance cable, the connections listed in Table 5-13 are provided. Cable connectors P1 and P2 are connected to DRV11 connectors J1 and J2, respectively. Note that CSR0 (J2-K), which can be set or reset under program control, is routed to the REQ A input (J1-LL); similarly, CSR1 (J1-DD) is routed to REQ B (J2-S). Hence, a maintenance program can output data to DROUTBUF and read the same data via the cable and DRINBUF. DRCSR bits 0 (CSR0) and 1 (CSR1) can be used to simulate REQ A and REQ B signals, respectively. If the appropriate INT ENB bit (DRCSR bits 5 or 6) is set, the simulated signal will generate an interrupt request.

### 5.7.5 Programming

**5.7.5.1 Addressing**—Addresses for the DRV11 can range from 16000 through 17777X<sub>8</sub>. The least significant three bits address the desired DRV11 register as follows:

| Address | Device Register |
|---------|-----------------|
| 1XXXX0  | DRCSR           |
| 1XXXX2  | DROUTBUF        |
| 1XXXX4  | DRINBUF         |

Addresses 177560-177566 are reserved for the console device and should not be used for DRV11 addressing. The following address assignments are normally used:

*First DRV11*

DRCSR = 167770  
 DROUTBUF = 167772  
 DRINBUF = 167774

*Second DRV11*

167760 to 167764

*Third DRV11*

167750 to 167754

**Table 5-13 BC08R Maintenance Cable Signal Connections**

| J2  |        | J1    |     |
|-----|--------|-------|-----|
| Pin | Name   | Name  | Pin |
| VV  | OPEN   | OPEN  | A   |
| UU  | GND    | OPEN  | B   |
| TT  | IN00   | OUT00 | C   |
| SS  | GND    | OPEN  | D   |
| RR  | INIT H | OPEN  | E   |
| PP  | GND    | OPEN  | F   |
| NN  | INIT H | OPEN  | H   |
| MM  | GND    | GND   | J   |
| LL  | IN01   | OUT01 | K   |
| KK  | IN04   | OUT04 | L   |
| JJ  | GND    | GND   | M   |

**Table 5-13 BC08R Maintenance Cable Signal Connections (Cont.)**

| J2  |            | J1           |     |
|-----|------------|--------------|-----|
| Pin | Name       | Name         | Pin |
| HH  | IN05       | OUT05        | N   |
| FF  | OPEN       | INIT H       | P   |
| EE  | IN06       | OUT06        | R   |
| DD  | GND        | GND          | S   |
| CC  | IN07       | OUT07        | T   |
| BB  | IN03       | OUT03        | U   |
| AA  | GND        | GND          | V   |
| Z   | IN08       | OUT08        | W   |
| Y   | IN09       | OUT09        | X   |
| X   | GND        | GND          | Y   |
| W   | IN10       | OUT10        | Z   |
| V   | IN11       | OUT11        | AA  |
| U   | IN12       | OUT12        | BB  |
| T   | GND        | GND          | CC  |
| S   | REQ B      | CSR1         | DD  |
| R   | GND        | GND          | EE  |
| P   | IN13       | OUT13        | FF  |
| N   | IN14       | OUT14        | HH  |
| M   | IN15       | OUT15        | JJ  |
| L   | GND        | GND          | KK  |
| K   | CSR0       | REQ A        | LL  |
| J   | GND        | GND          | MM  |
| H   | IN02       | OUT02        | NN  |
| F   | OPEN       | GND          | PP  |
| E   | IN02       | OUT02        | RR  |
| D   | OPEN       | GND          | SS  |
| C   | DATA TRANS | OPEN         | TT  |
| B   | OPEN       | GND          | UU  |
| A   | OPEN       | NEW DATA RDY | VV  |

**5.7.5.2 Interrupt Vectors**—Two interrupt vectors are jumper-selected in the range of 0 through 37X<sub>8</sub>. The least significant three bits identify the interrupting function.

|        |             |
|--------|-------------|
| 000XX0 | Interrupt A |
| 000XX4 | Interrupt B |

Vectors 60 and 64 are reserved for the console device and should not be used for DRV11 vectors.

**5.7.5.3 Word Formats**—The three word formats associated with the DRV11 are shown in Figure 5-30 and are described in Table 5-14.

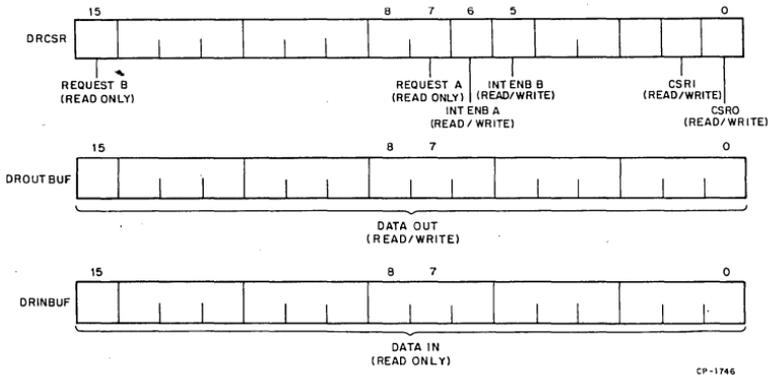


Figure 5-30 DRV11 Word Formats

Table 5-14 Word Formats

| Word  | Bit(s) | Function   |
|-------|--------|--|
| DRCSR | 15     | <p>REQUEST B—This bit is under control of the user's device and may be used to initiate an interrupt sequence or to generate a flag that may be tested by the program.</p> <p>When used as an interrupt request; it is asserted by the external device and initiates an interrupt provided the INT ENB B bit (bit 05) is also set. When used as a flag, this bit can be read by the program to monitor external device status.</p> <p>When the maintenance cable is used, the state of this bit is dependent on the state of CSRI (bit 01). This permits checking interface operation by loading a 0 or 1 into CSRI and then verifying that REQUEST B is the same value.</p> <p>Read-only bit. Cleared by INIT when in maintenance mode.</p> |
|       | 14-08  | Not used. Read as 0.   |
|       | 07     | <p>REQUEST A—Performs the same function as REQUEST B (bit 15) except that an interrupt is generated only if INT ENB A (bit 06) is also set.</p> <p>When the maintenance cable is used, the state of REQUEST A is identical to that of CSRO (bit 00).</p> <p>Read-only bit. Cleared by INIT when in maintenance mode.</p>   |

**Table 5-14 Word Formats (Cont.)**

| <b>Word</b> | <b>Bit(s)</b> | <b>Function</b>  |
|-------------|---------------|--|
|             | 06            | INT ENB A—Interrupt enable bit. When set, allows an interrupt request to be generated, provided REQUEST A (bit 07) becomes set.<br>Can be loaded or read by the program (read/write bit). Cleared by BINIT.  |
|             | 05            | INT ENB B—Interrupt enable bit. When set, allows an interrupt sequence to be initiated, provided REQUEST B (bit 15) becomes set.   |
|             | 04-02         | Not used. Read as 0.<br>Can be loaded or read by the program (read/write bit). Cleared by INIT.  |
|             | 01            | CSR1—This bit can be loaded or read (under program control) and can be used for a user-defined command to the device (appears only on Connector No. 1).<br><br>When the maintenance cable is used, setting or clearing this bit causes an identical state in bit 15 (REQUEST B). This permits checking operation of bit 15 which cannot be loaded by the program.<br><br>Can be loaded or read by the program (read/write bit). Cleared by INIT. |
| DRSCR       | 00            | CSR0—Performs the same functions as CSR1 (bit 01) but appears only on Connector No. 2.<br>When the maintenance cable is used, the state of this bit controls the state of bit 07 (REQUEST A).<br><br>Read/write bit. Cleared by INIT.  |
| DROUTBUF    | 15-00         | Output Data Buffer—Contains a full 16-bit word or one or two 8-bit bytes: High Byte = 15-8; Low Byte = 7-0.<br><br>Loading is accomplished under a program-controlled DATO or DATOB bus cycle. It can be read under a program-controlled DATI cycle.   |
| DRINBUF     | 15-00         | Input Data Buffer—Contains a full 16-bit word or one or two 8-bit bytes. The entire 16-bit word is read under a program-controlled DATI bus cycle.   |

## **5.8 DRV11-B DIRECT MEMORY ACCESS (DMA) INTERFACE**

### **5.8.1 General**

The DRV11-B is a general purpose Direct Memory Access (DMA) interface used to transfer data directly between the LSI-11 system memory and an

I/O device as shown on Figure 5-31. The interface is programmed by the processor to move variable length blocks of 16-bit data words to or from specified locations in memory by means of the LSI-11 bus. Once programmed, no processor intervention is required. The DRV11-B can transfer up to 250K, 16-bit words per second and is capable of operating in burst modes, with byte addressing. The control structure also allows read-modify-restore operations.

The interface consists of five registers: Word Count Register (WCR), Bus Address Register (BAR), Control Status Register (CSR), Input Data Buffer Register (IDBR), and Output Data Buffer Register (ODBR). The module also includes bus transceivers and logic for interrupt requests, address control and protocol, and DMA requests.

The DRV11-B contains one switch bank used to assign an appropriate device address to the DMA interface and one switch bank to select an interrupt vector address in the LSI-11 memory where the DMA routine is stored.

Two 40-pin connectors, mounted near the edge of the module, facilitate the connection of the I/O device with the DMA, using any two of several cable assemblies available from DIGITAL. The module may be inserted into any available slot of the LSI-11 backplane.

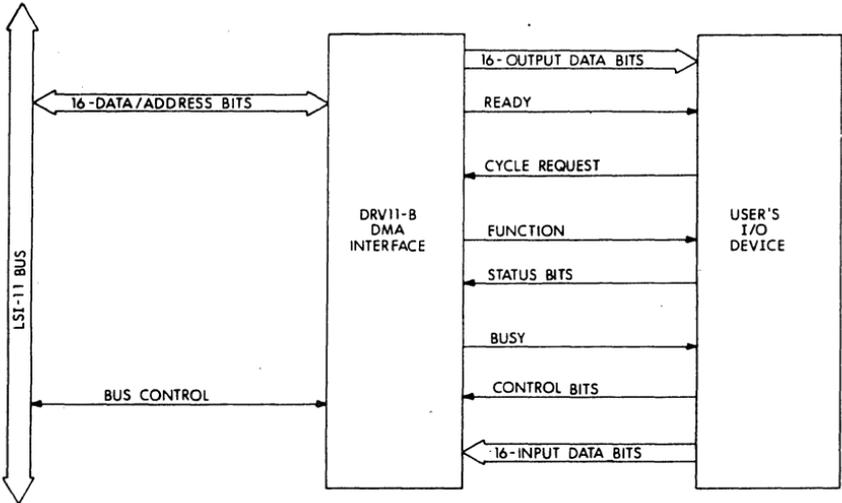


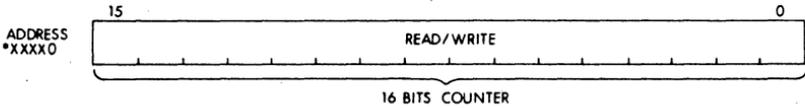
Figure 5-31 DRV11-B Interface Diagram

### 5.8.2 Registers

Each of the five registers can be addressed by the processor. The IDBR and ODBR are assigned the same address, and are read-only and write-only, respectively.

The register bit format and functions are described as follows.

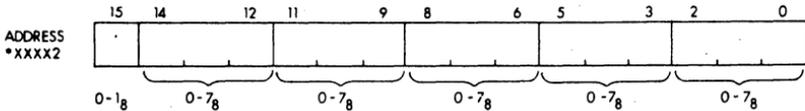
### Word Count Register (WCR)



\* All logic "ones" decoded by bus master to assert BBS7 L signal

The WCR is a 16-bit read/write counter which is loaded by the program with the two's complement of the number of words or bytes to be transferred at one time between memory and the I/O device. At the end of each transfer, the WCR is incremented. When the count becomes zero (all 16 bits = 0), the DMA generates an interrupt request. The contents of the WCR can be monitored by the processor program.

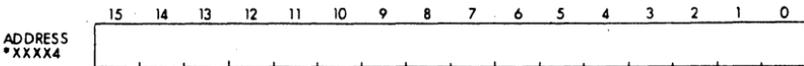
### Bus Address Register (BAR)



\* All logic "ones" decoded by bus master to assert BBS7 L signal

The BAR is a 15-bit read/write register used to generate the bus address which specifies the location to or from which data is to be transferred. The register is incremented after each transfer. It will increment across 32K boundary lines via the extended address bits in the control status register. Bus address bit 00 is driven by the user device.

### Control and Status Register (CSR)



\* All logic "ones" decoded by bus master to assert BBS7 L signal

The CSR contains 16 bits of information used to control the function and monitor the status of the DMA transfers. The information in the CSR can be modified or read by the processor program in either 8-bit bytes or 16-bit words. Table 5-15 lists and defines each of the 16 bits.

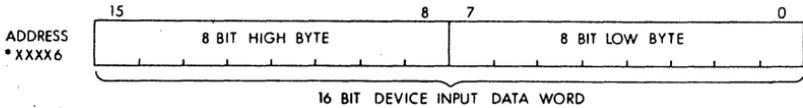
**Table 5-15 Control Status Register Bit Description**

| Bit | Name                 | Description  |
|-----|----------------------|--|
| 15  | Error<br>(Read Only) | 1. Indicates a special condition.<br>a. NEX (bit 14)<br>b. ATTN (bit 13) |

**Table 5-15 Control Status Register Bit Description (Cont.)**

| Bit | Name                     | Description  |
|-----|--------------------------|--|
|     |                          | <ol style="list-style-type: none"> <li>2. Sets READY (bit 7) and causes interrupt if IE (bit 6) is set.</li> <li>3. Cleared by removing the special condition.                             <ol style="list-style-type: none"> <li>a. NEX is cleared by writing to zero.</li> <li>b. ATTN is cleared by the user device.</li> </ol> </li> </ol> |
| 14  | NEX<br>(Read/Write Zero) | <ol style="list-style-type: none"> <li>1. Non-existent memory indicates that as bus master, the DRV11-B did not receive BRPLY or that a DATIO cycle was not completed.</li> <li>2. Sets Error (bit 15).</li> <li>3. Cleared by INIT or by writing to zero.</li> </ol>  |
| 13  | ATTN<br>(Read Only)      | <ol style="list-style-type: none"> <li>1. Indicates the state of the ATTN user signal.</li> <li>2. Sets Error (bit 15).</li> </ol>   |
| 12  | MAINT<br>(Read/Write)    | <ol style="list-style-type: none"> <li>1. Maintenance bit used with Diagnostic Program.</li> </ol>   |
| 11  | STAT A<br>(Read Only)    | <ol style="list-style-type: none"> <li>1. Device Status bits that indicate the state of the DSTAT A, B, and C user signals.</li> <li>2. Set and cleared by user control only.</li> </ol>   |
| 10  | STAT B<br>(Read Only)    |  |
| 09  | STAT C<br>(Read Only)    |  |
| 08  | CYCL<br>(Read/Write)     |  |
| 07  | READY<br>(Read Only)     | <ol style="list-style-type: none"> <li>1. Indicates that the DRV11-B is able to accept a new command. Requests an interrupt if IE (bit 6) is set.</li> <li>2. Set by NINT.</li> </ol>  |
| 06  | IE<br>(Read/Write)       | <ol style="list-style-type: none"> <li>1. Enables interrupts to occur when READY (bit 07) is set.</li> <li>2. Cleared by INIT.</li> </ol>  |
| 05  | XAD 17<br>(Read/Write)   | EXTENDED Address bit 17, cleared by INIT.  |
| 04  | XAD 16<br>(Read/Write)   | EXTENDED Address bit 16, cleared by INIT.  |
| 03  | FNCT 3<br>(Read/Write)   | <ol style="list-style-type: none"> <li>1. Three bits made available to the user device. User defined.</li> <li>2. Cleared by init.</li> </ol>  |
| 02  | FNCT 2<br>(Read/Write)   |  |
| 01  | FNCT 1<br>(Read/Write)   |  |
| 00  | GO<br>(Write Only)       | <ol style="list-style-type: none"> <li>1. Causes "NOT READY" to be sent to the user device indicating a command has been issued. Clear READY (bit 7). Enables DMA transfers.</li> </ol>  |

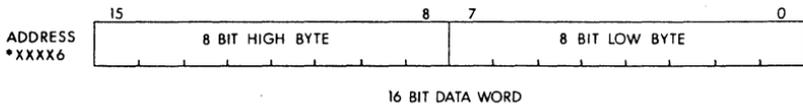
## Input Data Buffer Register (IDBR)



\* All logic "ones" decoded by bus master to assert BBS7 L signal

The IDBR is used for read-only operations. Data is loaded into the register by the user's device. The data may be read from the IDBR as a 16-bit word, an 8-bit high byte or an 8-bit low byte. Transfers are usually via DATO or DATOB DMA bus cycles. The register input connects to J2 mounted on the module.

## Output Data Buffer Register (ODBR)



\* All logic "ones" decoded by bus master to assert BBS7 L signal

The ODBR is used during write-only operations. Data from the LSI-11 bus is loaded into the register under program control and read from the register by the user's device. The register can be loaded with a 16-bit data word or with an 8-bit high byte, or as an 8-bit low byte. Transfers are usually via DATI or DATIO DMA bus cycles. The output of the register connects to J1 on the module.

### 5.8.3 Device And Vector Address Selection

**5.8.3.1 General**—The address of the DRV11-B interface and the interrupt vector address in memory is selected by the position of the switches in switch bank S1 and S2, respectively. The location of the switches on the module is shown on Figure 5-32. The switches are set to the OFF position (open) to select a zero bit in the address format and the ON position (closed) to select a one.

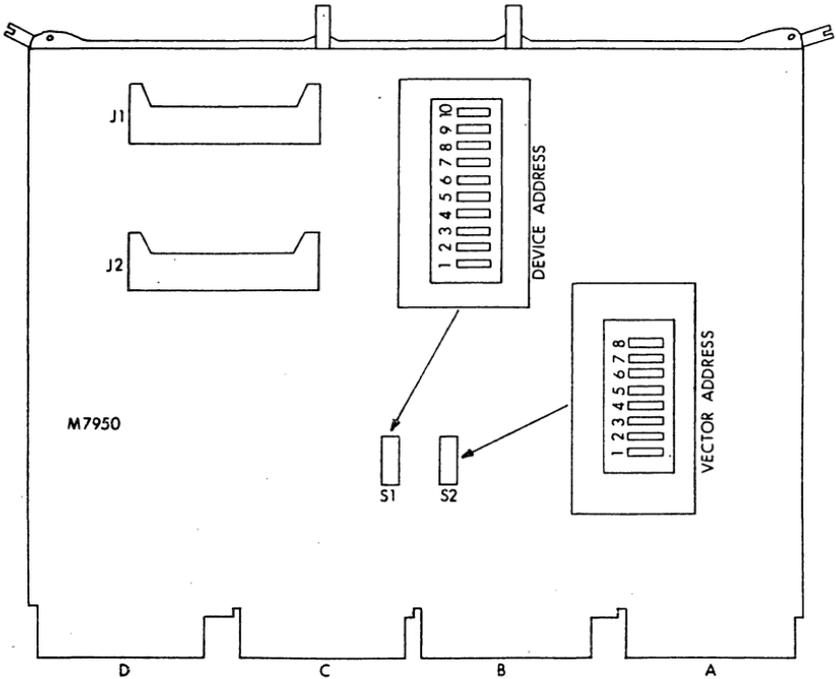
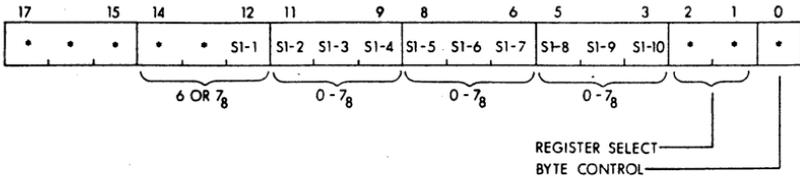


Figure 5-32 Connector and Switch Locations

**5.8.3.2—Device Address Format**—The DRV-11B decodes four address, one for each of the registers listed:

| Register | Octal Address |
|----------|---------------|
| WCR      | *XXXX0        |
| BAR      | *XXXX2        |
| CSR      | *XXXX4        |
| DBR      | *XXXX6        |

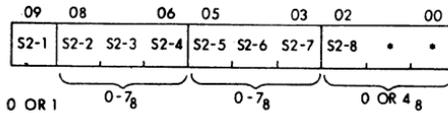
Normally, the addresses assigned to the DMA start at  $772410_8$  and progress upward. Switches S1-1 through S1-10 select the base address as indicated by the X portion of the octal code and the individual registers are decoded by the DMA interface. The relationship between the address format and the switches are shown on Figure 5-33.



\* = All logic "ones" decoded by processor as BBS7-L signal

Figure 5-33 Device Address Switch (S1) Selection

**5.8.3.3 Interrupt Vector Address Selection**—The interrupt vector addresses for the LSI-11 systems are allocated memory locations from  $0-774_8$ . The recommended location assigned to the DRV11-B is  $124_8$ . Switches S2-1 through S2-8 are used to select the octal address and the relationship between the switches and address format is shown on Figure 5-34.



\* = Preassigned as zero

Figure 5-34 Interrupt Vector Address Switch (S2) Selection

## 5.8.4 Functions

**5.8.4.1 General**—The DRV11-B interface operates as both a slave and master device. Prior to becoming bus master, all Data Transfers Out (DATO) or Data Transfer In (DATI) are in respect to the processor. Once DMA is granted bus mastership by the processor, all data transfers are in respect to the DMA.

DMA operation is initialized under program control by: (1) loading the WCR with the two's complement of the number of words to be transferred; (2) loading the BAR with the first address to or from which data is to be transferred; (3) loading the CSR with the desired function bits. After the interface is initialized, data transfers are under control of the DMA logic.

**5.8.4.2 Program Control Transfers**—Data transfers may be performed under program control by addressing the IDBR or ODBR and reading or writing data.

**5.8.4.3 DMA Control Transfers**—DMA input (DATI) or output (DATO) data transfers occur when the processor clears READY. For a DATO cycle (DRV11-B to memory transfer), the user's I/O device presets the CONTROL BITS [word count increment enable (WC INC ENB), bus address increment enable (BA INC ENB), C1, CO, A00, and ATTN], and asserts CYCLE REQUEST to gain use of the LSI-11 bus. When CYCLE REQUEST is asserted, input data is latched into the input DBR, the CONTROL BITS are latched into the DRV11-B DMA control, and BUSY goes low. A DATI cycle—memory to DRV11-B transfer—is handled in a similar manner, except that the output data is latched into the output DBR at the end of the bus cycle.

When the DRV11-B becomes bus master, a DATO or DATI cycle is performed directly to or from the LSI-11 memory location specified by the BAR. At the end of each cycle, the WCR and BAR are incremented and BUSY goes high while READY remains low. A second DATO or DATI cycle is performed when the user's I/O device again asserts CYCLE REQUEST. DMA transfers will continue until the WCR increments to zero, at which time READY goes high and the DRV11-B generates an interrupt (if interrupt enable is set) to the LSI-11 processor.

If burst mode is selected (SINGLE CYCLE low), only one CYCLE REQUEST is required for the complete transfer of the specified number of data words.

## 5.8.5 Device Cables and Signals

Data, status and control signals are transferred between the user's I/O device and DMA by an input and an output cable assembly. The input cable attaches to connector J2 and the output cable attaches to connector J1 as shown on Figure 5-32, Table 5-16 and 5-17 list the connector pin and designations for each signal. Table 5-18 lists several recommended cable assemblies that are available from DIGITAL in the lengths indicated. The H856 female connector mates with either J1 or J2 on the DRV11-B. To order cable assemblies in lengths not listed, contact a DIGITAL sales office.

**Table 5-16 Input Connector Signals**

| <b>J2*</b><br>Connector Pin | <b>Signal Name</b> | <b>Unit Loads</b> |
|-----------------------------|--------------------|-------------------|
| B                           | BUSY H             | 10 (drive)        |
| D                           | ATTN H             | 1                 |
| F                           | A00 H              | 1                 |
| J                           | BA INC ENB H       | 1                 |
| K                           |                    |                   |
| L                           | FNCT 3 H           | 10 (drive)        |
| N                           | C0 4               | 1                 |
| R                           | FNCT 2 H           | 10 (drive)        |
| T                           | C1 H               | 1                 |
| V                           | FNCT 1 H           | 10 (drive)        |
| DD                          | 08 IN H            | }                 |
| FF                          | 09 IN H            |                   |
| JJ                          | 10 IN H            |                   |
| LL                          | 11 IN H            |                   |
| NN                          | 12 IN H            |                   |
| RR                          | 13 IN H            |                   |
| TT                          | 14 IN H            |                   |
| VV                          | 15 IN H            |                   |
| CC                          | 07 IN H            |                   |
| EE                          | 06 IN H            |                   |
| HH                          | 05 IN H            |                   |
| KK                          | 04 IN H            |                   |
| MM                          | 03 IN H            |                   |
| PP                          | 02 IN H            |                   |
| SS                          | 01 IN H            |                   |
| UU                          | 00 IN H            |                   |

\*All remaining pins connect in common to logic ground by board etch.

**Table 5-17 Output Connector Signals**

| <b>J1*</b><br>Connector Pin | <b>Signal Name</b> | <b>Unit Loads</b> |
|-----------------------------|--------------------|-------------------|
| B                           | CYCLE REQUEST H    | 1                 |
| D                           | INIT V2 H          | 10 (drive)        |
| F                           | INI H              | 10 (drive)        |
| J                           | WC INC ENB H       | 1                 |
| K                           | SINGLE CYCLE H     | 1                 |
| L                           | STATUS A           | 1                 |
| N                           | READY H            | 10 (drive)        |
| R                           | STATUS B           | 1                 |
| T                           |                    |                   |
| V                           | STATUS C           | 1                 |
| DD                          | 08 OUT H           | }                 |
| FF                          | 09 OUT H           |                   |
| JJ                          | 10 OUT H           |                   |
| LL                          | 11 OUT H           |                   |
| NN                          | 12 OUT H           |                   |
| RR                          | 13 OUT H           |                   |

**Table 5-17 Output Connector Signals (Cont.)**

| J1* | Connector Pin | Signal Name | Unit Loads |
|-----|---------------|-------------|------------|
|     | TT            | 14 OUT H    | 10 (drive) |
|     | VV            | 15 OUT H    |            |
|     | CC            | 07 OUT H    |            |
|     | EE            | 06 OUT H    |            |
|     | HH            | 05 OUT H    |            |
|     | KK            | 04 OUT H    |            |
|     | MM            | 03 OUT H    |            |
|     | PP            | 02 OUT H    |            |
|     | SS            | 01 OUT H    |            |
|     | UU            | 00 OUT H    |            |

\*All remaining pins connect in common to logic ground by board etch.

**Table 5-18 Recommended Cable Assemblies**

| Cable No. | Connectors       | Type                   | Standard Lengths (ft.)   |
|-----------|------------------|------------------------|--------------------------|
| BC07D-XX  | H856 to open end | 2, 20 conductor ribbon | 10, 15, 25               |
| BC08R-XX  | H856 to H856     | Shielded flat          | 1, 6, 10, 12, 20, 25, 50 |
| BC04Z-XX  | H856 to open end | Shielded flat          | 6, 10, 15, 25, 50        |

## 5.9 LSI-11 BUS FOUNDATION MODULE

### 5.9.1 General

The DRV11-P (Figure 5-35) is a versatile wire wrap module that contains the bus interface logic for operation with the LSI-11 or PDP-11/03 system and provides adequate board area for mounting and connecting integrated circuits (IC's) or discrete components. Because the bus interface logic is included, the module can be efficiently configured by the user to satisfy a variety of device interface logic applications.

A 40-pin connector, conveniently mounted at the board edge, facilitates the connection to a device through several cable assembly types available from DIGITAL.

Except for the bus interface connections, all signals and voltages are terminated to wire wrap pins for user connections. The bus control logic is provided with wire wrap test points for monitoring the internal signals. The test points are spaced at 0.1 in. (0.254 cm) between pins to allow a 40-pin connector to be inserted over the wire wrap pins for automated test functions.

Approximately 2/3 of the surface area on the module consists of plated-through holes, each connected to a wire wrap pin. The user can mount three different types of dual-in-line IC's or a variety of discrete components into the holes and connect the proper voltages and signals by wire wrapping leads on the board.

The DRV11-P module can be inserted into any one of the available interface option locations of the LSI-11, PDP-11/03 backplane, or back-

plane extender unit. The module occupies four vertical slots. Refer to the documentation supplied with the option for detailed information.

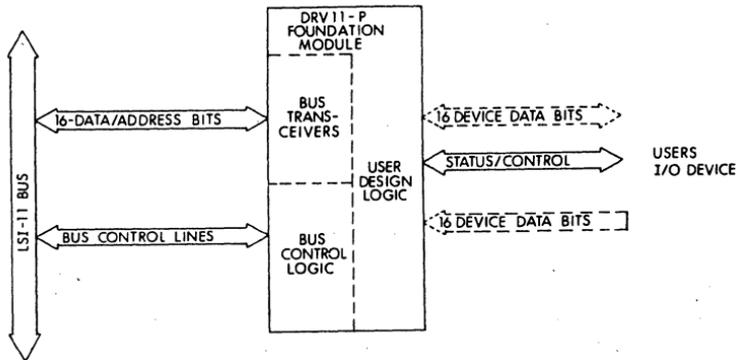


Figure 5-35 Typical DRV11-P Interface

## 5.9.2 Functions

**5.9.2.1 General**—The DRV11-P contains 16 bus transceivers, device selection and vector address generation logic, interrupt control, and control and status register functions. The device data inputs and outputs of the bus transceivers and the device control signals are made available to user to complement control of up to four 16-bit registers.

**5.9.2.2 Address Selection Logic**—The address selection logic consists of a device address comparator and the protocol control logic. Up to four discrete addresses are made available with the existing logic on the DRV11-P and can be assigned to data registers, status and control registers, or word counters. By adding additional IC's, the user can increase the total number of addresses available. The main address of the DRV11-P is selected by monitoring the BBS7 bus line and decoding the address information D03-D12 from the bus. The main device address is assigned by the configuration of jumper leads (A03-A08) attached to wire wrap pins. When the selected and input bus addresses are the same, the device address comparator provides an ENB H level to the protocol control logic. The protocol control logic receives bus signals and address bits D01, D02 to assert one of the four available output lines—SEL DEV 0L, SEL DEV 2L, SEL DEV 4L, and SEL DEV 6L. In addition, the protocol control logic provides output signals to specify word or byte transfers.

Table 5-19 lists and defines the function of the control signals required or available for the user logic.

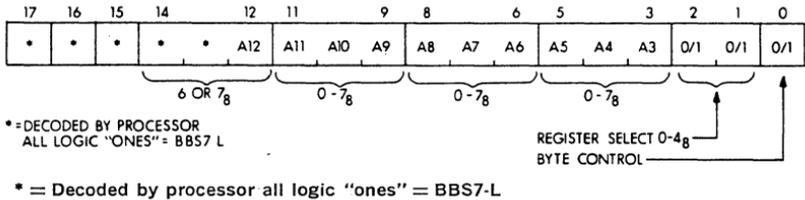
Table 5-19 Protocol Control Logic Signals

| Signal     | Function  |
|------------|---|
| SEL DEV 0L | Select Device 0 through 4—One of four lines asserted by decoding the device address and available to select |
| SEL DEV 2L |   |

**Table 5-19 Protocol Control Logic Signals (Cont.)**

| Signal                   | Function  |
|--------------------------|---|
| SEL DEV 4L<br>SEL DEV 6L | one of four user word registers.  |
| OUT LB L<br>OUT HB L     | Out Low Byte, Out High Byte—Used to load (write) data into low byte (8 bits) or high byte (8 bits) or both bytes (16 bits) of the selected word register. |
| IN WD L                  | In Word—Used to gate (read) data from the selected word register to the bus.  |

The format for the device address selection is shown on Figure 5-36. A logic one is specified when no jumper lead is installed between the appropriate wire wrap pin from A3-A12. A logic zero is specified when a jumper lead is installed.



**Figure 5-36 Device Address Selection**

**5.9.2.3 Interrupt Control Logic**—The interrupt control provides the circuits necessary to allow a program interrupt transaction between the LSI-11 and device. Two interrupt channels (A and B) are available to the user with channel A assigned the highest priority. Table 5-20 lists and defines the user available signals associated with the interrupt control logic.

**Table 5-20 Interrupt Control Logic Signals**

| Signal       | Function  |
|--------------|---|
| RQST A H     | Interrupt Request A—Asserted by device logic and sets the channel A Interrupt Request flip-flop when the channel A Interrupt Enable flip-flop is set. |
| ENB DATA A H | Interrupt Enable A Data—Asserted by device logic and sets the channel A Interrupt Enable flip-flop when the ENB CLK A signal is asserted.             |
| ENB CLK A    | Interrupt Enable A Clock—Asserted by device logic to cause the channel A Interrupt Enable flip-flop to be set when ENB DATA A signal is asserted.     |
| ENB A ST H   | Interrupt Enable A Status—Indicates the status of the channel A Interrupt Enable flip-flop.   |

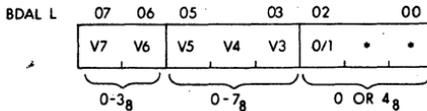
**Table 5-20 Interrupt Control Logic Signals (Cont.)**

| Signal       | Function   |
|--------------|--|
| RQST B H     | Interrupt Request B—Same as RQST A H signal except controls channel B interrupts.  |
| ENB DATA B H | Interrupt Enable B Data—Same as ENB DATA A H signal except controls channel B interrupts.  |
| ENB CLK B    | Interrupt Enable B Clock—Same as ENB CLK A signal except controls channel B interrupts.  |
| ENB B ST H   | Interrupt Enable B status—Same as ENB A ST H except controls channel B interrupts.   |
| VECTOR H     | Interrupt Vector Gate—Used by device logic to gate vector address onto the bus and to generate B RPLY signal.  |
| VEC RQST H   | Vector Request—Asserted by device logic to specify that channel A vector address is required; negated to specify channel B vector address is required. |
| INIT O L     | Initialize Out—Buffered B INIT L signal from bus used for general initialization.  |

**5.9.2.4 Vector Address and CSR Logic**—The vector address logic is used in conjunction with the interrupt control logic to generate a vector address on bus lines BDAL 00L-BDAL 07. The vector address is specified by the user and selected by installing jumper leads between wire wrap pins on the M7948 module. The addresses available are from 000<sub>8</sub> to 374<sub>8</sub>. The vector address range can be increased from 000<sub>8</sub> to 774<sub>8</sub> with additional logic and wiring.

When the VECTOR H signal is asserted as a result of a device interrupt request, the vector address is placed on the bus lines.

Wire wrap pins V3 through V7 are used to assign the vector address. A jumper lead installed selects a logic “zero” address bit for its associated line and no lead selects a logic “one” address bit according to the format on Figure 5-37.



\* = Preset by M7948 to 0 bit

**Figure 5-37 Vector Address Select Format**

Bit BDAL 02 L can be connected to the device interrupt request signal RQST A signal to specify a separate vector address for channel A and channel B.

Status and control information can be multiplexed through the same logic used to generate the vector address. Up to eight status and control bits can be assigned by the user and transferred to bus lines BDAL 00 L-BDAL 07 L. The information can be gated onto the bus lines using a select level generated by the address decoding logic.

### 5.9.3 Component Mounting Area

**5.9.3.1 General**—Twelve vertical areas (A-L) are available on the M7948 module for mounting integrated circuits or discrete components as shown on Figure 5-32. Each area has a double row of wire wrap pins that connect to an associated plated through hole located at 0.1 in. (.254 cm) vertical spacing. Area A is for multi-use and is capable of accepting IC's with pin centers at 0.3 in. (.762 cm), 0.4 in. (1.01 cm) or 0.6 in. (1.52 cm). Area K will also accept IC's with pin centers at 0.3 in. 0.4 in. All remaining areas will only accept IC's with pin centers at 0.3 in.

Table 5-21 lists the total number of IC's with 0.3 in. spacing that can be mounted in the user areas of the module, A through L.

**Table 5-21 IC MOUNTING**

| IC Type | Total Number |
|---------|--------------|
| 14-pin  | 60           |
| 16-pin  | 52           |
| 18-pin  | 44           |
| 20-pin  | 44           |

**5.9.3.2 Connector Wire Wrap Pins**—The 36 contact pins in row C and D at the edge of the module connect to a double row of wire wrap pins. These two rows are made available to the user for connecting signals and voltages from the backplane to the user installed logic circuits. The following pins of row C and D are normally dedicated to +5V and GND.

The user can connect the power to the IC or components using the row C and row D wire wrap pins.

|      |               |
|------|---------------|
| +5 V | CA2, DA2      |
| GND  | CJ1, CM1, CT1 |
|      | DJ1, DM1, DT1 |
|      | CC2, DC2      |

**5.9.3.3 Device Signals**—Input and output data, status and control signals can be transferred between the device and the DRV11-P module using any one of several cable assemblies listed on Table 5-21 and available from DIGITAL. One end of each cable is terminated with a 40-pin female connector which mates with the 40-pin male connector J1 mounted on the M7948 module. The pins of J1 connect to the user installed logic through a series of wire wrap pins.

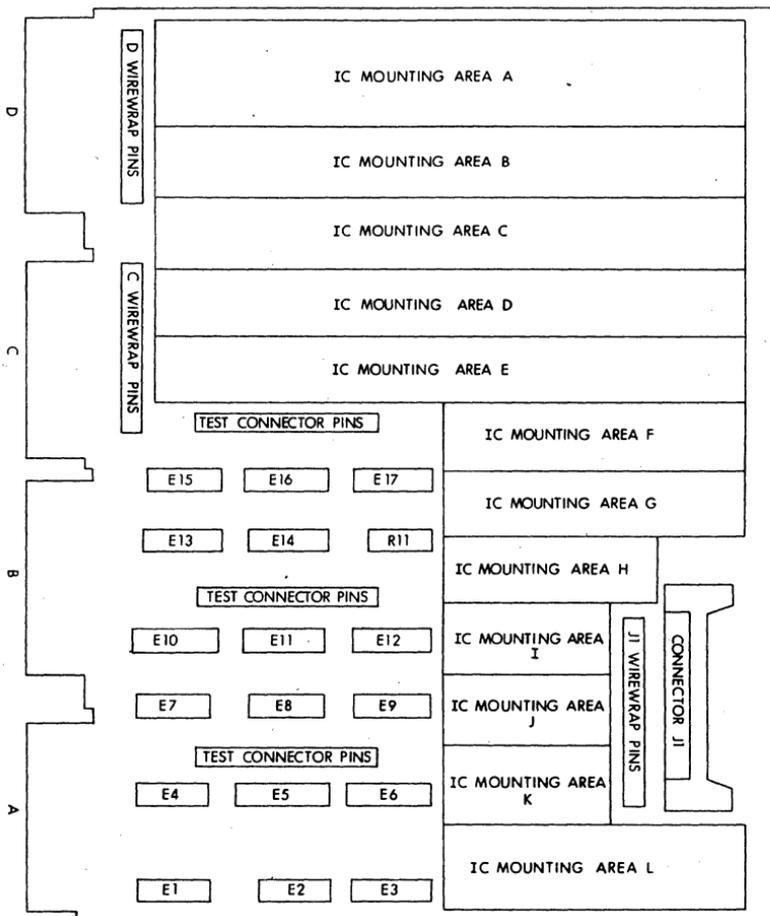


Figure 5-38 DRV11-P Component Mounting Locations

Table 5-21 Recommended Cable Assemblies

| Cable No. | Connectors       | Type                   |                                   |
|-----------|------------------|------------------------|-----------------------------------|
| BC07A-XX  | H856 to open end | 20-twisted pair        | 10, 15, 25                        |
| BC07D-XX  | H856 to open end | 2, 20 conductor ribbon | 10, 15, 25                        |
| BC08R-XX  | H856 to H856     | Shielded flat          | 1, 6, 10, 12, 20, 25, 50, 75, 100 |
| BC04Z-XX  | H856 to open end | Shielded flat          | 6, 10, 15, 25, 50                 |

## INSTALLATION

**6.1 GENERAL**

This chapter contains the basic considerations and requirements for configuring and installing LSI-11 or PDP-11/03 systems. The following paragraphs apply to both LSI-11 systems and PDP-11/03 systems, except where clearly stated otherwise.

**6.2 CONFIGURATION CHECKLIST**

LSI-11 and PDP-11/03 systems comprise user-selected module options as required for a particular application. Each module may require jumper alterations or switch settings to provide the correct addressing, operation, etc., for the user's application. A module configuration checklist for each module type is provided below. Detailed information for configuring the modules can be obtained by referring to the paragraphs listed in the checklist.

***KD11 Processor Jumpers***

- Power-up mode (Paragraph 5.2.2)
- Memory enable (Paragraph 5.2.2)
- Line time clock enable (Paragraph 5.2.2)
- Resident memory 4K address selection and reply (KD11-F only)

***MSV11-B 4K by 16 Random Access Memory Jumpers***

- Memory address (Paragraph 5.3.2)
- Reply to refresh (Paragraph 5.3.3)

***MMV11-A Core Memory 4K Address Selection***

- 4K address select switches (Paragraph 5.4.2)

***MRV11-A PROM/ROM Memory Jumpers***

- Memory address (Paragraph 5.5.3)
- Reply signal (Paragraph 5.5.4)
- 512 by 4-bit or 256 by 4-bit PROMs (Paragraph 5.5.2)

***DLV11 Serial Line Unit Jumpers***

- Device address (Paragraph 5.6.2.2)
- Vector address (Paragraph 5.6.2.3)
- Universal asynchronous receiver transmitter operation (Paragraph 5.6.2.4)
- Baud rate selection (Paragraph 5.6.2.5)
- EIA interface (Paragraph 5.6.2.6)
- 20mA current loop interface (Paragraph 5.6.2.7)
- Framing error halt (Paragraph 5.6.2.8)

***DRV11 Parallel Line Unit Jumpers and Pulse Width Modification***

- Device address (Paragraph 5.7.2.2)
- Vector address (Paragraph 5.7.2.3)
- NEW DATA READY and DATA TRANSMITTED pulse width modification (Paragraph 5.7.4.6)

### **DRV11-B DMA Interface**

Device address (Paragraph 5.8.3.2)

Vector address (Paragraph 5.8.3.3)

### **DRV11-P LSI-11 Bus Foundation Module**

Device address (Paragraph 5.9.2.2)

Vector address (Paragraph 5.9.2.3)

The following checklist is for LSI-11 system configurations. It includes items that are not contained on particular modules but which must be checked to ensure that the system is properly installed.

1. BDCOK, BPOK, BEVNT, and BHALT signals connected as required to backplane assembly (Paragraph 11.7.6).
2. Modules inserted in backplane slots according to desired priority (Paragraph 6.3).
3. Jumpers added to backplane when core memory (MMV11-A) is located between processor and I/O device modules (Paragraph 6.3.3).
4. Correct cabling selected for I/O device modules (Paragraph 6.5).
5. Modules inserted in backplane slots with components facing in the correct direction (Paragraph 6.4).
6. Correct power and ground inputs to backplane connector block (Paragraphs 6.7.3 and 6.7.4).
  - a. Voltage and current requirements met
  - b. Correct terminal block power connections made
  - c. Proper ground connection
7. Environmental requirements met (Paragraph 6.7.5).

#### **NOTE**

Special cooling considerations might be required if more than one core or PROM module, or a combination of core and PROM modules, is implemented on one H9270 backplane assembly.

## **6.3 DEVICE PRIORITY**

### **6.3.1 General**

Device priority is established by the relative position of the device interface module along the I/O bus in which the devices are installed. The H9270 and DRV11-B backplanes are structured to allow the user to configure device priority by installing modules in appropriate positions. The PDP-11/03 includes one factory-installed H9270 backplane.

### **6.3.2 Priority Selection Using the H9270 Backplane**

Figure 6-1 is a front view of the H9270 backplane, showing typical module locations. The processor module should be installed in backplane slots A1-D1.

The LSI-11 bus structure includes two daisy-chained signals: BIAKO L/BIAKI L (for interrupts) and BDMGO L/BDMGI L (for DMA grant). These signals normally propagate through option modules until they reach the requesting device. Option 1, as shown in Figure 6-1, is the first device location to receive the daisy-chained signals when the processor module is installed in slots A1-D1. Hence, six options can be in-

stalled in the backplane. The PDP-11/03 is shipped with the processor module installed in the backplane as shown in the figure. Do not relocate the processor module to another location; a separate non-bused (jumper) connection is provided on the backplane to this location for proper RUN indicator operation.

### CAUTION

Do not configure the system with unused option locations in the backplane between the processor module and I/O devices that require either of the two daisy-chained signals; an unused location will break the daisy-chain signal continuity, and devices in higher numbered locations will not receive interrupt or DMA grant signals. Unused locations should occur only in the highest numbered option locations.

Note that the daisy-chained BIAK and BDMG signals always follow in increasing numbered option locations, as shown in the figure.

### 6.3.3 H9270 Backplane/MMV11-A Configuration

The MMV11-A position on the backplane should be carefully considered when configuring the system. The MMV11-A's physical size is four times greater than other LSI-11 memory options, and it will require either two or four device (or option) locations on the backplane, depending on where it is located on the backplane. It is actually comprised of two 8.5 by 10 in. modules that are mated in a single assembly. However, only one module has fingers that plug into the backplane. Hence, if the MMV11-A is installed in backplane row 4, the MMV11-A module not having backplane fingers will be located below the backplane (where "row 5" should be located) and rows 2 and 3 will be available for other options. Thus, row 4 is the recommended location for the MMV11-A.

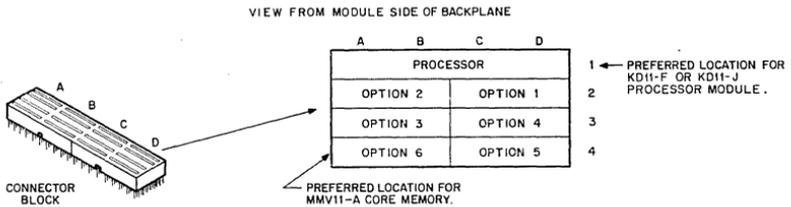


Figure 6-1 Typical H9270 Backplane Configuration  
—Processor and Option Locations

If the MMV11-A is installed in row 2, as shown in Figure 6-2, row 3 will also be occupied by the MMV11-A; however, the portion of the assembly in row 3 does not have backplane fingers. If any device modules are to be installed in row 4, it is necessary to install jumpers on the backplane in order to complete the DMA and interrupt grant signal chain. These jumpers (two required) should be wire wrapped between the backplane pins listed below:

### H9270 Backplane/MMV11-A Jumpers

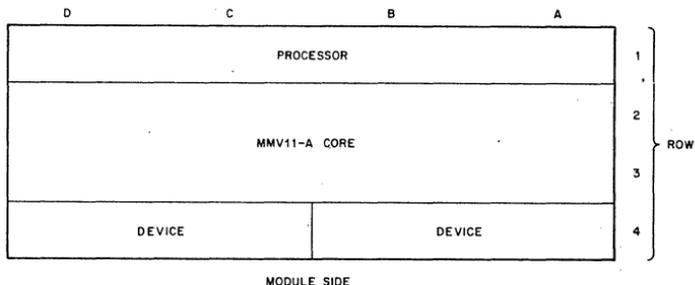
| From  | To    | Signal   |
|-------|-------|----------|
| C01N2 | C04M2 | BIAKI/LO |
| C01S2 | C04R2 | BDMGI/LO |

**NOTE**

These jumpers are required only if the MMV11-A is installed in row 2.

**6.3.4 DDV11-B Expanded Backplane Configuration**

Device priority on the DDV11-B backplane is established in the same manner as described for the H9270 backplane. However, larger physical size allows up to 16 options (including a bus terminator module) to be installed on the backplane. Device (option) locations are shown in Figure 6-3. The highest priority location is Option 1; the lowest priority location is option 16.



NOTE:  
This is not a preferred configuration, for the preferred configuration, refer to figure 11-1.

CP-1760

Figure 6-2 H9270 Backplane/MMV11-A Core

**VIEW FROM MODULE SIDE OF BACKPLANE**

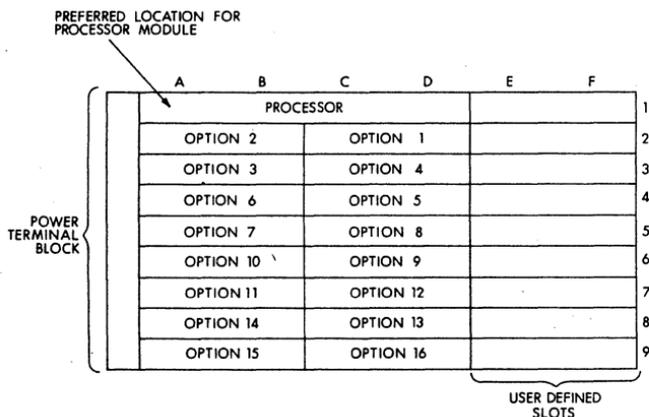
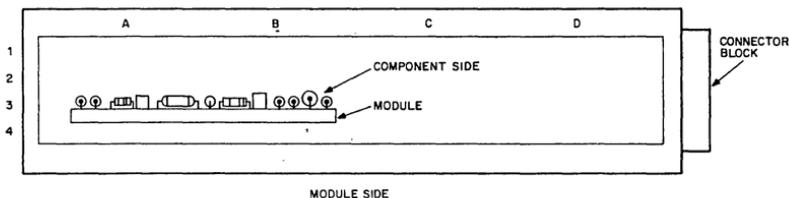


Figure 6-3 Typical DDV11-B Backplane Configuration—Processor and Option Locations



CP 1761

Figure 6-4 Module Installation in the H9270 Backplane

When using the MMV11-A core memory on the DDV11-B backplane, use the lowest priority option locations for the MMV11-A. Memory options, including core and semiconductor, are not priority dependent. If this procedure is followed, it will not be necessary to install BIAK1/O and BDMG1/O jumpers as described in paragraph 6.3.3. However, if the MMV11-A is installed between the processor module and any device requiring interrupt or DMA service, the jumpers must be installed.

When more than six options are installed on the backplane, a bus termination module is required. The TEV11 bus terminator module option is normally used for this purpose. Install the TEV11 in the last location (option 16 in Figure 6-3).

#### NOTE

This configuration is similar to the "intermediate configuration" shown in Figure 3-11. The bus must be terminated as specified in paragraph 3.13 for minimum and intermediate configurations.

The REV11-A terminator, DMA refresh, bootstrap ROM option can be used as the bus termination instead of the TEV11. However, this option requires DMA service and must be installed as directed in paragraphs 6.8. Improper REV11-A installation can result in improper memory refresh operation; loss of semiconductor memory contents may result.

#### 6.4 MODULE INSERTION AND REMOVAL

Modules must be installed or removed only when dc power is removed from the backplane. The PDP-11/03 contains a control/indicator panel on the front of the power supply; the DC ON/OFF switch allows the user to turn off dc power for safe module insertion and removal.

Modules must be installed in the backplane with components facing row 1, as shown in Figure 6-4.

Certain modules are equipped with metal handles that facilitate module installation and removal. These modules include the KD11-F and KD11-J processor modules, the MMV11-A core memory unit, the DRV11-B DMA interface, and the DRV11-P LSI-11 bus foundation module. When installing a module equipped with the metal handles into the H9270 backplane (or the DDV11-B backplane equipped with the H0341 card cage assembly), carefully start the module fingers into the backplane connector block while inserting the metal handle fingers into the card

cage as shown in Figure 6-5. Once the module has been started into the backplane in this manner, insertion can be completed by pressing downward on the handles; both handles must be pressed simultaneously. Module removal can be accomplished by simultaneously raising both handles until the handle fingers clear the card cage. The module can then be easily removed.

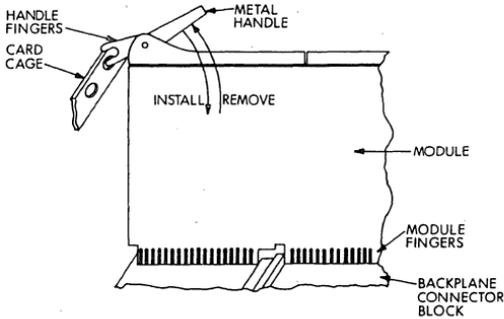


Figure 6-5 Module Insertion Using Metal Handles

**CAUTION**

The LSI-11 modules and the backplane assembly mounting blocks may be damaged if the modules are plugged in backward.

DC power must be removed from the backplane during module insertion or removal.

**6.5 I/O CABLING**

Recommended I/O cable options for use with the DLV11 serial line unit and DRV11 parallel line unit are listed below:

**DLV11 Serial Line Unit**  
 20 mA Current Loop  
 EIA Interface

**Cable\***  
 BC05M-X  
 BC05C-X

**DRV11 Parallel Line Unit**

**Data Cable—**  
 Two data cables (input data and output data) are required. Available in lengths of 10, 15, and 25 ft. Can be cut and terminated at the user's device.

**Cable\***  
 BC07D-X

**Maintenance Cable—**  
 One maintenance cable is required for certain diagnostic tests.

BC08R-X

**DRV11-B Parallel Line Unit**  
 Two data cables (input and output data) are required.

**Cable\***  
 BC04Z-X

\*The -X in the cable number denotes length in feet, as follows: -1, -6, -10, -20, -25. For example, a 10ft EIA interface cable would be ordered as BC05C-10.

## **6.6 PDP-11/03 INSTALLATION PROCEDURE**

### **6.6.1 Packaging and Mounting**

The PDP-11/03 is packaged as shown in Figure 2-5. It is designed with a removable front panel. Removing the front panel exposes the LSI modules and cables. This enables replacement or installation of a module from the front of the PDP-11/03. The 11/03 power supply is located on the right-hand side of the PDP-11/03 when viewed from the front. The power supply contains three front panel switches and indicators that are accessible through a cutout in the front panel. Therefore, when the front panel is removed, the lights and switches are still attached and functional.

The PDP-11/03 is designed to mount in a standard 19 in. cabinet (Figure 2-6). A standard 19 in. cabinet has two rows of mounting holes in the front, spaced  $18\frac{5}{16}$  in. apart. The holes are located  $\frac{1}{2}$  in. or  $\frac{5}{8}$  in. apart. Standard front panel increments are  $1\frac{3}{4}$  in.

### **6.6.2 Power Requirements**

Input (primary) power requirements are listed in Paragraph 2.3.2.

An appropriate power cable and plug is supplied with all PDP-11/03 models. Note that a ground wire (and ground pin on the plug) must be connected to the normal service ground to ensure safe operation. Do not cut or remove the ground pin.

The H780 power supply provides the required dc power for the backplane in the PDP-11/03 enclosure. Typical dc power requirements will range from 33 to 120 W (max). In addition, the power supply generates the necessary BPOK H and BDCOK H power supply status signals, displays the RUN and DC status, and contains the ENABLE/HALT, DC ON/OFF, and LTC ON/OFF control switches.

Before attempting to operate the system, ensure that the system is configured as previously described in this chapter, and that environmental requirements are met.

### **6.6.3 Environmental Requirements**

The PDP-11/03 will operate at temperatures of 41° to 104° F (5° to 40° C) with a relative humidity of 10 to 90 percent (no condensation), with adequate air flow across the modules. The fans in the H780 power supply will provide adequate air flow within the specified temperature range.

## **6.7 LSI-11 SYSTEM INSTALLATION**

### **6.7.1 General**

When installing the LSI-11 system, the user must mount the backplane; provide dc operating power, ground, and externally generated bus signals; and observe system environmental requirements. The following paragraphs describe the above items in detail.

### **6.7.2 Mounting the Backplane**

The H9270 backplane (Figure 2-1) is designed to accept the KD11-F or KD11-J microcomputer and up to six I/O interface or memory modules. Mounting of the H9270 backplane can be accomplished in any one of three planes, as shown in Figure 2-3.

The DDV11-B backplane (Figure 2-4) is designed to accept the KD11-F or KD11-J microcomputer and up to 15 I/O interface or memory modules, and one bus terminator module. Mounting dimensions for the DDV11-B are provided in the figure. The optional H0341 card cage assembly can be installed on the backplane; special mounting holes (not shown) are provided for that purpose.

### 6.7.3 DC Power Connections

**6.7.3.1 Voltage and Current Requirements**—A power supply for a single H9270 backplane LSI-11 system should have the following capacity:

- +5 V  $\pm$ 5% load; 0—18 A static/dynamic
- +12 V  $\pm$ 3% load; 0—2.5 A static/dynamic
- +5 ripple: less than 1% of nominal voltage
- +12 ripple: less than 150 mV pp (frequency 5 kHz)

#### NOTE

Regulation at the H9270 backplane must be maintained to the specifications listed above.

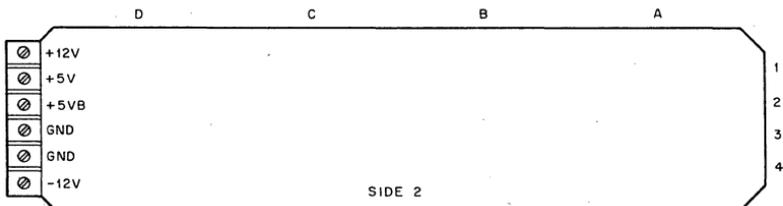
A power supply for a DDV11-B, or a multiple-backplane system using H9270 backplanes should have the same voltage regulation and ripple specifications as listed for the single H9270 backplane. However, it will be necessary to calculate the actual power requirements, based on individual power requirements for modules used in the system. Refer to Table 2-1 for this information.

**6.7.3.2 Backplane Power Connections**—Perform the following steps to connect power to the H9270 backplane (Figure 6-6):

1. Select wire size. (14 gauge is recommended.) Consider load current and distance between the power supply and backplane.
2. For a standard system, connect the applicable wires to the H9270 connector block per Table 11-1.

For battery backup, remove the jumper between +5V and +5VB and connect the applicable wires to the H9270 connector block per Table 11-2.

3. Connect the ground terminals at the power sources.
4. It is recommended that the backplane frame/casting be electrically connected to system/power supply ground.



CP-1762

Figure 6-6 H9270 Backplane Terminal Block

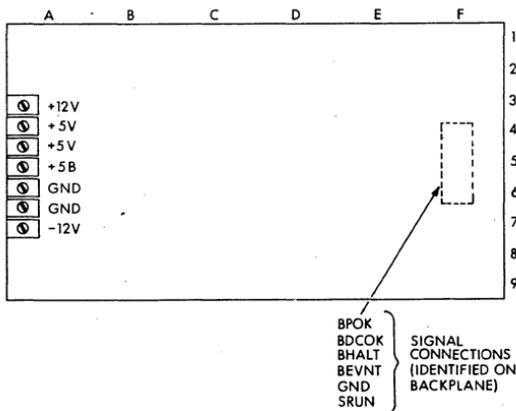
**Table 6-1 H9270 Backplane Standard Power Connections**

| Power Source (From) | H9270 Connector Block (To) |  |
|---------------------|----------------------------|--|
| +12V                | +12V                       | Factory Connected  |
| +5V                 | +5V                        |  |
|                     | +5B                        |  |
| GND                 | GND                        | Factory Connected  |
| GND                 | GND                        |  |
| -12V                | -12V                       | (This voltage is not required. The connection is available for custom interfaces.) |

**Table 6-2 H9270 Backplane Battery Backup Power Connections**

| Power Source (From)  | H9270 Connector Block (To) |  |
|----------------------|----------------------------|--|
| +12V                 | +12V                       | Remove Factory Connection  |
| +5V (System Power)   | +5V                        |  |
| +5B (Battery Backup) | +5B                        |  |
| GND                  | GND                        | Factory Connected  |
| GND                  | GND                        |  |
| -12V                 | -12V                       | (This voltage is not required. The connection is available for custom interfaces.) |

Power connections to the DDV11-B backplane are accomplished in the same manner as described for the H9270. The DDV11-B backplane terminal block power connections are located as shown in Figure 6-7.



**Figure 6-7 DDV11-B Backplane Terminal Block**

#### **6.7.4 Backplane Ground Connection**

Connect the backplane ground wire to system (or frame) ground in which backplane is installed. The ground terminal is located as shown in Figures 6-6 and 6-7.

#### **6.7.5 Environmental Requirements**

All LSI-11 modules will operate at temperatures of 41° to 122° F (5° to 50° C) with a relative humidity of 10 to 90 percent (no condensation), with adequate air flow across the modules. When operating at the maximum temperature (122° F or 50° C), air flow must maintain the inlet to outlet air temperature rise to 12.5° F (7° C) maximum. Air flow should be directed across the modules as shown in Figure 6-8.

#### **6.7.6 Externally Generated Bus Signals**

**6.7.6.1 General**—Externally generated bus signals include BDCOK H and BPOK H power status, BEVNT L (line time clock) (if required), and BHALT L (if desired). The signals are applied to the H9270 backplane via a connector and an optional mating connector as shown in Figure 6-9. The signals must conform to LSI-11 bus specifications described in Chapter 3. Connections made to the backplane via the ribbon cable shown in the figure must not exceed 12 inches in length. The DDV11-B includes signal connection pins in the same connector configuration. They are located as shown in Figure 6-7. Each signal is discussed in the following paragraphs.

**6.7.6.2 BDCOK H and BPOK H**—Correct sequencing of the BDCOK H and BPOK H signals is most necessary when the LSI-11 system contains core memory. (Core memory is supplied with the KD11-J LSI-11 processor and MMV11-A core memory option.) Proper sequencing of the power signals will allow power-up and power-down sequencing without loss of memory data. In addition, a power fail routine can be programmed that will save the contents of CPU registers during power down, and automatically restore CPU registers and restart the interrupted program (if desired) during the power-up sequence.

Since the 4K semiconductor memory used on the KD11-F processor and MSV11-B option is volatile (data is lost during absence of power), proper sequencing of power signals during power-down is not required; however, all systems can benefit through the use of proper power signal sequencing to bring the processor to an orderly Halt during power off/fail. This is important when the LSI-11 system is used in control applications that require an orderly Halt. Proper sequencing of the signals is not required during power-up if a manually operated INITIALIZE switch is installed (Figure 6-10). If automatic initialization, or the orderly Halt is required, use the power signal generation circuit shown for core memory applications (Figure 6-11). Otherwise, use the simple initialization circuit described for non-core (MOS memory) systems.

#### **NOTE**

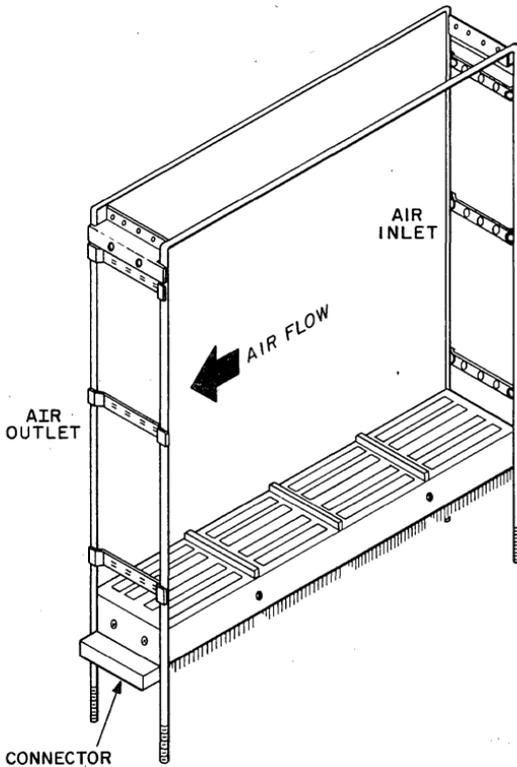
A switch bounce eliminator must be used with the manual BDCOK switch shown in Figure 6-10.

If the manual method of applying BDCOK H is selected, contents of semiconductor read/write

memory may be lost when the BDCOK switch is depressed.

It is not necessary to negate the BPOK H signal when manually initializing the processor. BPOK H may be left unconnected.

The switch may be placed in the BDCOK ON position only when +5 V and +12 V supply voltages are applied to the backplane. Place the switch in the INITIALIZE position during power down or power up.



CP-1764

Figure 6-8 H9270 Backplane Air Flow

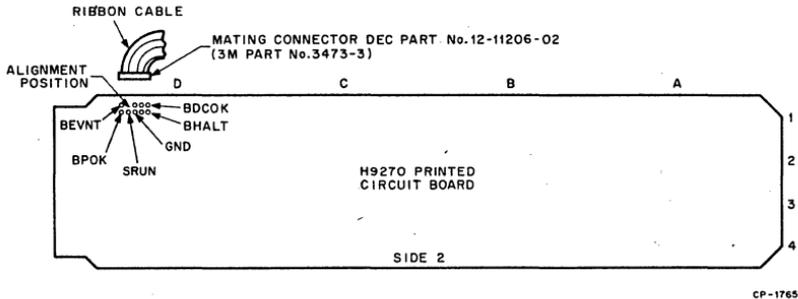
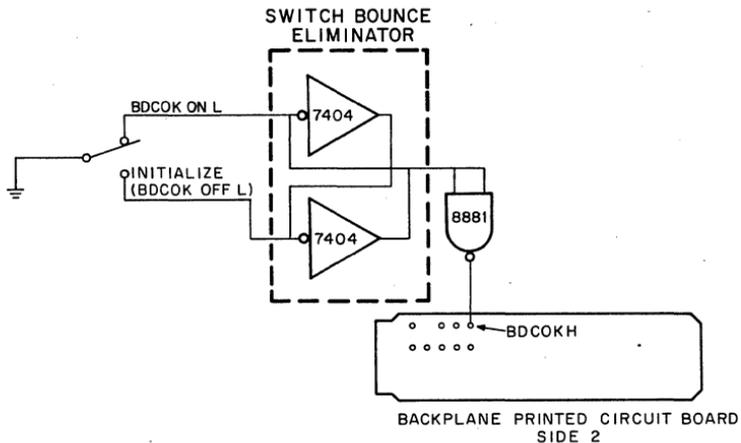


Figure 6-9 H9270 Backplane Signal Connections



NOTE:  
 BDCOKH SIGNAL (ASSERTED HIGH)  
 LOW: 1.3V MAX  
 HIGH: 1.7V MIN

CP-1768

Figure 6-10 INITIALIZE Switch Circuit

Figure 6-11 shows the circuit for the generation of the power sequence signals for the LSI-11 system. The circuit can be constructed by the user from standard, off-the-shelf components and incorporated into the LSI-11 system. Operating power for the power signal generation circuit is derived from the ac power line by means of a 28 Vac step-down transformer. The rectified secondary is applied to two three-terminal regulators. The 7805 regulator produces regulated +5-V, while the 320 regulator produces regulated -5 V for the power signal generation circuits. Circuit operation is described in the following paragraphs.

#### NOTES

1. R-C values shown are nominal for the delay times shown. Trim valves to produce the delay times shown.

2. Adjust resistance to just maintain the retrig-gered state with normal ac input. Sufficient range is provided for 50Hz or 60Hz operation.
3. Use a transformer with an appropriate pri-mary circuit for 115 V or 230 V operation. Secondary voltage/current should be 28 V, 0.250 A (nominal).
4. All resistors are  $\frac{1}{4}$  W.

**Power Up**—During the power-up sequence, ac voltage from the trans-former secondary is applied to a Schmitt trigger circuit (Q1 and Q2). The Schmitt trigger squares the ac sine wave and drives level converter Q3. Q3's output is a TTL-compatible signal. This square wave signal is applied directly to one input of an exclusive-OR gate, and to the second input of the exclusive-OR gate via a  $5 \mu\text{s}$  delay circuit. This gating of the square wave produces a  $5 \mu$  pulse on each transition of the square wave at a rate of 120 or 100 pps (two times the line frequency). The pulse triggers the 10 ms one-shot and its output goes high. Successive pulses normally retrigger the one-shot and its output remains high.

During the power-up sequence, +5 V SEN and +12 V SEN (voltage sense) signals rise to voltage levels that cause voltage comparators A and B to produce high outputs. The comparator outputs are connected together and applied to one input of gate A. The remaining input of gate A is enabled by the high one-shot output signal that is applied (but not delayed) via driver B and the 3 ms delay circuit. Gate A's output goes high. This signal is then delayed 4 ms and inverted, producing a low signal that is applied to the non-inverting input of comparator C and driver C. Comparator C's output goes low, turning off Q4 and producing an active BDCOK H signal 4 ms (minimum) after ac power is applied.

Driver C's output goes high enabling gate B. The remaining gate B input is enabled by the high one-shot output signal. Gate B's high output signal is delayed 70 ms and inverted, producing a low signal that is applied to the non-inverting input of comparator D. Comparator D's output signal goes low, turning off Q5, and producing the active BPOK H signal 70 ms after the active BDCOK H signal. With both signals in the active (high) state, normal system operation can proceed.

Note that the one-shot circuit includes a 10K potentiometer. The poten-tiometer allows for adjustment of the nominal 10 ms delay over an appropriate range for 50-Hz or 60-Hz line operation. It is adjusted to a point where the one-shot normally remains retrigged for the 10 ms (50-Hz line) or 8.34 ms (60-Hz line) period between ac line transitions.

**Power Down**—When an ac power failure occurs, the trigger pulses to the one-shot cease, and the one-shot times out. Its output goes low, in-hibiting gate B and gate A (via driver B). Gate B's output goes low; this low signal is inverted, but not delayed, by the 70 ms delay circuit, and the resulting high signal is applied to the non-inverting input of com-parator D. Comparator D's output goes high, turning on Q5, and negating BPOK H. Meanwhile, the high driver B output signal is delayed 3 ms



and inverted. The resulting low signal inhibits gate A causing its output signal to go low. The low signal is inverted (but not delayed) by the 4 ms delay circuit and applied to the non-inverting input of comparator C. Comparator C's output goes high, turning on Q4 and negating the BDCOK H signal 3 ms after BPOK H becomes negated.

A feature of the power signal generation circuit shown in Figure 6-11 is the line time clock (LTC) output BEVNT L signal. This signal is produced by Q3's square wave output signal at the 60-Hz or 50-Hz line frequency. Its use in the LSI-11 system is optional; however, if it is used, the circuit described in Paragraph 6.7.6.3 should not be used.

The BEVNT L, BDCOK H, and BPOK H signals produced by the power signal generation circuit are connected to the LSI-11 backplane by means of the nine male pins on the backplane. A mating female nine-pin connector (DIGITAL part no. 12-11206-02, or 3M part no. 3473-3) should be used to connect the power signals to the backplane.

**6.7.6.3 BEVNT L Signal**—The BEVNT L signal input to the backplane is the external event interrupt. Asserting the BEVNT L signal initiates the LTC (line time clock) interrupt on the processor. The processor will trap through location 100<sub>h</sub> if PS bit 7 = 0. A typical circuit for generating BEVNT L is shown in Figure 6-12.

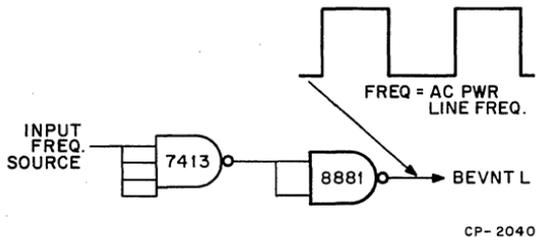


Figure 6-12 BEVNT Signal

**6.7.6.4 BHALT L Signal**—Manual control of the Halt mode can be obtained by connecting a BHALT L signal line to the backplane printed circuit board. The BHALT L signal level should meet bus specifications described in Paragraph 3.12.

When in the Halt mode, user program execution is not performed and the processor executes ODT console microcode. However, the processor will execute memory refresh in a normal manner and respond to DMA requests, even when BHALT is asserted; all device and LTC interrupt requests are ignored.

## 6.8 USING LSI-11 BUS ACCESSORY OPTIONS

### 6.8.1 General

Several LSI-11 bus accessory options are available for bus expansion, bus termination, DMA refresh, bootstrap ROM, and combinations of the preceding. The options can be used in both LSI-11 and PDP-11/03 applications. A summary of the options is provided in Table 6-3.

**Table 6-3 LSI-11 Bus Options**

| <b>Option No.</b> | <b>Includes</b>   | <b>System Functions</b>   |
|-------------------|---|---|
| REV11-A           | M9400-YA Module   | 120 $\Omega$ bus terminator, DMA refresh, bootstrap ROM.  |
| REV11-C           | M9400-YC Module   | DMA refresh, bootstrap ROM.   |
| TEV11             | M9400-YB Module   | 120 $\Omega$ bus terminator.  |
| BCV1B-XX          | Two BC05L-XX cables, one M9400-YE module, and one M9401 module. | Bus expansion: 250 $\Omega$ terminator (M9400-YE), two expansion cables, backplane connector (M9401). Normally used for expansion from first to second backplane in 2 or 3 backplane systems. |

**NOTE**

The -XX in BCV1A-XX and BCV1B-XX options denotes cable lengths. Options are available with cable lengths of 2, 4, 6, and 10 ft. For example, a BCV1A-06 includes two 6-ft cables.

|          |   |  |
|----------|---|--|
| BCV1A-XX | Two BC05L-XX cables, one M9400-YD module, and one M9401 module. | Bus expansion: two expansion cables and two backplane connector modules (M9400-YD and M9401). Normally used for expansion from second to third backplane in three-backplane systems. (A TEV11 120 $\Omega$ terminator must be installed in the last device slot in backplane 3.) |
|----------|---|--|

**NOTE**

BCV1A-XX and BCV1B-XX options, when used in a three backplane system, should be ordered with a 4 ft (minimum) cable length difference. For example, a BCV1A-02 and BCV1B-06 comprise a three-backplane set. The difference in length will locate signal transients (if present) to occur on the cables, rather than on one of the backplanes.

**6.8.2 Using the REV11-A**

**6.8.2.1 Installation**—The REV11-A is normally factory-installed in a system backplane as part of a complete system. However, if the module is removed for service or system modifications, it must be properly installed to ensure proper system operation. Items to be considered when installing the REV11-A are:

1. Jumpers (M9400-YA and processor module)
2. Module location on backplane

**CAUTION**

The use of other DMA devices that are installed on the I/O bus between the REV11-A and the processor module will likely result in ineffective DMA refresh and loss of memory data. When DMA devices are to be used in addition to the

REV11 option, use the REV11-C (no termination resistors) and the TEV11; install the REV11-C as the highest priority DMA device and the TEV11 as the 120  $\Omega$  bus termination in the last option location.

**Jumpers**—Three jumpers are normally installed on the M9400-YA module, as shown in Figure 6-13. Jumpers W2 and/or W4 can be removed to alter REV11-A operation as follows:

| Jumper | Normal<br>(Installed) Function | Altered<br>(Removed) Function |
|--------|--------------------------------|-------------------------------|
| W2     | DMA refresh enabled            | DMA refresh disabled          |
| W4     | Bootstrap ROM enabled          | Bootstrap ROM disabled        |

**Module Location on Backplane**—The REV11-A is used in two or three backplane configurations that require a 120  $\Omega$  bus termination in the last option location on the I/O bus. The REV11-A must not be used in system applications that use other DMA devices on the I/O bus, since those devices would have higher priority. Do not allow any option locations between the M9400-YA and the processor module to remain unoccupied; option locations must be occupied in order to pass the processor's daisy-chained BDMG1/O L signal to the M9400-YA module. Hence, the module should be located in the last *available* option location on the last (second or third) backplane.

**6.8.2.2 Operation**—Bootstrap ROM programs included in this option are used as described in Section II, Chapter 3.

### 6.8.3 Using the REV11-C

The REV11-C is identical to the REV11-A except the bus termination function is not included. This option should be installed as the first (highest priority) DMA device in the system. As supplied from the factory, the bootstrap ROM and DMA refresh functions are implemented. If desired, one of the functions can be disabled by removing a jumper, as directed for the REV11-A option (paragraph 6.8.2.1).

### 6.8.4 Using the TEV11

The TEV11 is a 120  $\Omega$  terminator module that is used in LSI-11 multiple backplane assemblies. Install the TEV11 in the last available location in the last (second or third) backplane.

### 6.8.5 Using the BCV1B

The BCV1B option includes two BC05L cables, one M9400-YE module, and one M9401 module. This option is always used to connect the first backplane to a second backplane in multiple backplane systems.

Install the M9400-YE module in the last location in the first backplane, slots A4, B4. Ensure that all option slots in the first backplane are occupied. This is necessary to ensure that daisy-chained BIAK and BDMG signals will be applied to the last option in the backplane (M9400-YE module). Install the M9401 module in the first option slot (A1, B1) of the second backplane. Install the two BC05L cables between the M9400-YE and M9401 modules. Note that J1 on each module are connected by the same cable; similarly, J2 on each module are connected by the second cable.

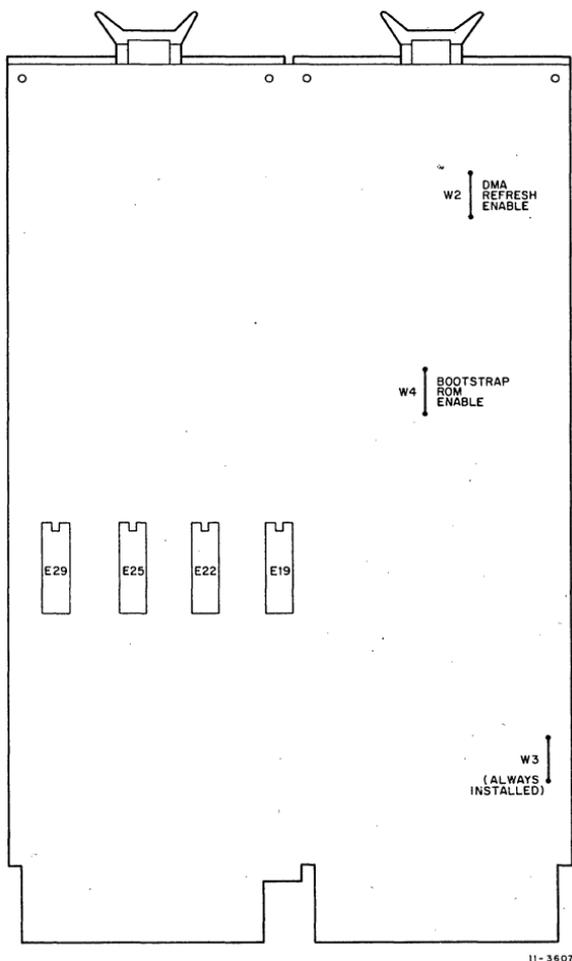
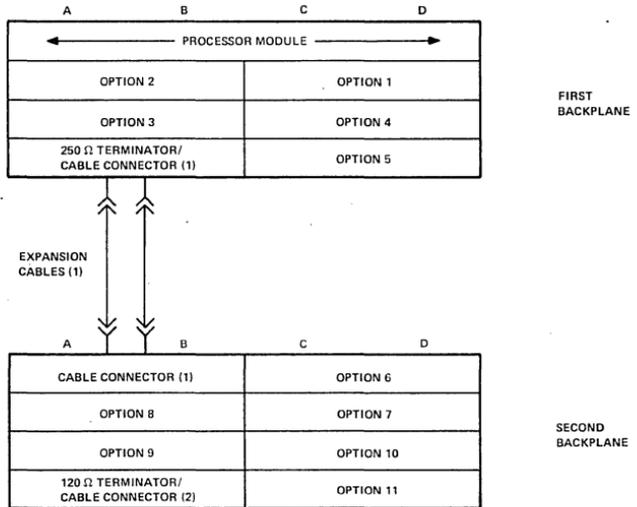


Figure 6-13 REV11-A, -C, Jumpers

The completed installation for a two backplane system using the BCV1B option is shown in Figure 6-14. A 120  $\Omega$  bus termination is required in the last option slot in the second backplane in a two backplane configuration. This function is normally provided by the TEV11 option, previously described.

### 6.8.6 Using the BCV1A

The BCV1A option includes two BC05L cables, one M9400-YD module, and one M9401 module. This option is always used to connect the second backplane to the third backplane in a three backplane system.



- Notes:
1. Included in BCV1B bus expansion option. (Cables are available in 2, 4, 6, or 12 ft. lengths.)
  2. Included in TEV11 bus terminator option.

CP-2047

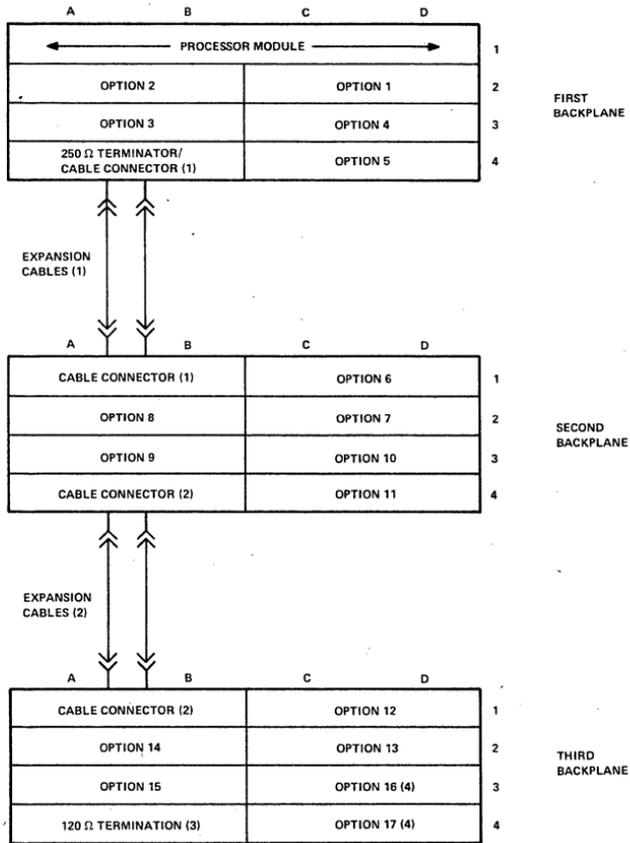
Figure 6-14 BCV1B Installation

Install the M9400-YD module in the last option location in the second backplane, slots A4, B4. Observe that all option slots in the first and second backplanes are occupied. This is necessary to ensure that daisy-chained BIAK and BDMG signals will be applied to the last option slot in the second backplane, in which the M9400-YD is installed. Install the M9401 module in the first option slot (A1, B1) in the third backplane. Install the two BC05L cables between the M9400-YD and M9401 modules. J1 on each module are connected by the same cable. Similarly, J2 on each module are connected by the second cable.

The completed installation for a three backplane system using the BCV1A option is shown in Figure 6-15. In addition to this option, the BCV1B option is required to connect the first backplane to the second backplane, a 120 Ω bus termination is required in the last option slot in the third backplane. The 120 Ω bus termination function is normally provided by the TEV11 option, as previously described.

### 6.9 USING BA11-ME AND BA11-MF EXPANSION BOXES

Install the BA11-ME (115 v, 60 Hz) or BA11-MF (230 v, 50 Hz) expansion box into a rack using the procedure described for the PDP-11/03 (Paragraph 6.6.1). When using an expansion box to expand from a single to a dual backplane system, the BCV1B bus expansion option and TEV11 bus terminator options must be used. Install the BCV1B modules and cables as shown in Figure 6-16. The TEV11 can be installed in option location "8" in the expander box. Carefully fold excess cable as shown in the figure. Refer to Paragraphs 6.8.4 and 6.8.5 for proper installation of the BCV1B and TEV11 options.



**Notes:**

1. Included in BCV1B bus expansion option. (Cables are available in 2, 4, 6, or 12 ft. lengths.)
2. Included in BCV1A bus expansion option. (Cables are available in 2, 4, 6, or 12 ft. lengths.)
3. Included in TEV11 bus terminator option.
4. The LSI11 Bus is restricted to 15 options, maximum. These option slots would only be used when previous option(s) occupy more than 1 option location.
5. BCV1A and BCV1B expansion cables must differ in length by four feet (minimum).

CP - 2048

**Figure 6-15 BCV1A Installation**

When expanding from a second to a third backplane, the BCV1B bus expansion option is required, in addition to the items required for expansion to the second backplane. The BCV1A installation is described in Paragraph 6.8.6. Note that the TEV11 is installed in the third backplane (second expansion box).

**NOTE**

BCV1A and BCV1B cables must differ in length by four feet (minimum).

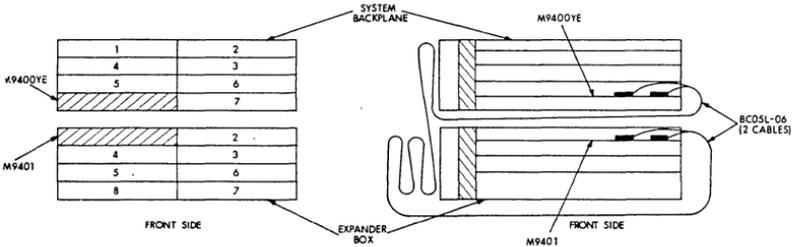


Figure 6-16 BA11 Expansion Box interconnections (two-backplane system)



## USING PROMs

**7.1 GENERAL**

This chapter contains specific instructions for programming, loading, and installing PROMs for use in the MRV11-AA module. MRV11-AC PROMs and user-supplied  $512 \times 4$  or  $256 \times 4$  PROMs are covered in the following paragraphs.

**7.2 PROM TYPES**

Basically, two general types of PROMs can be used in the MRV11-AA module:  $512 \times 4$  bit, and  $256 \times 4$  bit. The MRV11-AA module contains sockets for installation of up to 32 PROMs. Only the types listed in this chapter are recommended; the particular pinning and I/O levels for the devices listed are fully compatible with MRV11-AA addressing and data interface. Note that PROMs are always used in multiples of four, comprising the 16-bit LSI-11 word format. Hence, a minimum configuration of four PROM chips will comprise either a  $256 \times 16$  or  $512 \times 16$  read-only memory function. Recommended types are listed in Table 7-1.

**Table 7-1 MRV11-AA PROM Types**

| Manufacturer<br>or Source | 512 4-Bit<br>Chips | 256 4-Bit<br>Chips |
|---------------------------|--------------------|--------------------|
| Digital Equipment Corp.   | MRV11-AC           | —                  |
| Intersil                  | IM5624             | IM5623             |
| Signetics                 | 82S131             | 82S129             |
| MMI                       | 6306               | 6301               |

**NOTE**

Refer to PROM chip manufacturer's instructions for actual blasting procedure and recommended equipment.

**7.3 PROGRAMMING NOTES**

Generally, programs or data that can be read from read/write memory can also be read from PROMs. However, special care is required when using the MTPS-instruction and KEV11-option EIS instructions. These instructions are listed below:

| Mnemonic | Octal Code | Instruction               |
|----------|------------|---------------------------|
| MTPS     | 1064SS     | Move byte to PS           |
| MUL      | 070RSS     | Multiply                  |
| DIV      | 071RSS     | Divide                    |
| ASH      | 072RSS     | Shift arithmetically      |
| ASHC     | 073RSS     | Arithmetic shift combined |

These instructions, when executed on an LSI-11 processor (or PDP-11/03 system), fetch source operands via the DATIO bus cycle, rather than the DATI bus cycle. Hence, fetching a source operand from a PROM or ROM location will result in a bus error (time-out) because the processor will attempt to write into the addressed location after fetching the operand.

This potential problem can be avoided when writing the program by simply including a separate MOVE instruction. First, MOVE the source operand from the PROM or ROM location to a general register or a location in read/write memory. The MTPS or appropriate EIS installation is then executed using the general register and read/write memory location as the source operand.

Two examples are shown below using general register R4 and memory location TEMP as the source operand:

1. Using a general register:

MOV NEWPS, R4;    move source operand from PROM to temporary (general) register.

MTPS R4;            move NEWPS to PS.

2. Using a temporary read/write memory location:

MOV CONS, TEMP;    move source operand from PROM to temporary location in read/write memory.

MUL R1, TEMP;      multiply the contents of R1 by the CONSTANT in TEMP.

When programming PROMs for use as an RT-11 bootstrap, use  $256 \times 4$  PROMs instead of  $512 \times 4$  PROMs. This will allow the MRV11-AA address to be configured in the 173000-173776 range. Processor module power-up mode 2 can then be used for automatically bootstrapping RT-11 during system turn-on. Avoid using  $512 \times 4$  PROMs in this application. If  $512 \times 4$  PROMs are used, the MRV11-AA will respond in the 172000-173776 address range and the RT-11 Editor (EDIT.SAV) cannot run properly. This problem exists because the Editor tests for a peripheral device (the VT11) in the 172000-172776 address range. The problem can be avoided by using  $256 \times 4$  PROMs, as described.

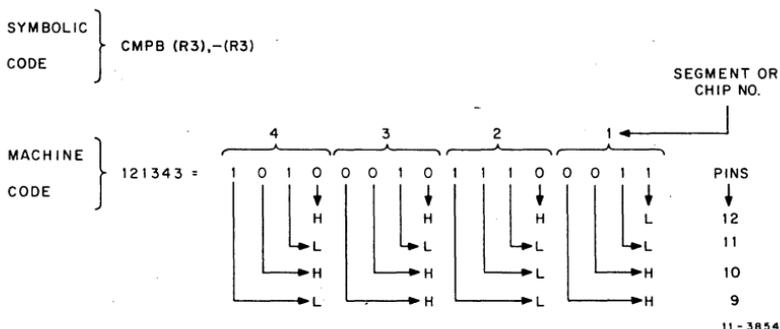


Figure 7-1 Data Format

## 7.4 LOADING PROMs

### 7.4.1 General

Loading (blasting, burning, or programming) PROMs is the process where the binary information is permanently stored in the PROM chip locations. This is a destructive process that must be carefully executed as directed by the appropriate PROM chip manufacturer's instructions.

### 7.4.2 Word Format

Each PROM word, when read by the LSI-11 processor, is stored in four 4-bit slices in four separate PROM chips. Each word is simultaneously addressed and produces its respective 4-bit portion of the 16-bit word that is read. For example, consider the CMPB instruction shown in Figure 7-1. Its machine code, using the addressing modes shown, is  $121343_8$  or  $1010001011100011_2$ . The binary bits are stored in chips numbered from 1 to 4. Chip output pins, as indicated, will yield the read data bits for this instruction when addressed.

Since the word format is contained in four 4-bit slices (one slice in each PROM chip), the user must load each PROM chip with successive memory locations, but dedicated to one 4-bit slice. This information can be generated manually—an error-prone, time-consuming process—or it can be generated automatically using the optional QJV11 ROM program software, described later.

### 7.4.3 Addressing

PROM chips, when installed in the MRV11-AA module, are addressed by low-active address bits. When loading PROMs, the user must be careful that the correct addressing technique is used. An example of this addressing technique, relative to PROM chip pins, is provided in Table 7-2. Note that  $256 \times 4$  bit and  $512 \times 4$  bit PROMs are addressed in exactly the same manner, except for pin 14 which is A8 in the  $512 \times 4$  bit part, and CE in the  $256 \times 4$  bit part. Also note that LSI-11 bus address bit 0 (DAL0 L) is not used in this application since all read operations are 16-bit word bus transfers.

The optional QJV11 ROM programmer software addresses PROM chips in the manner described herein.

The MRV11-AA address word format for  $512 \times 4$  and  $256 \times 4$  PROM applications is shown in Figure 7-2. Note the BDAL0 is not used in the address word format; BDAL1 corresponds to PROM chip address bit A0. The 4K bank select bits and 2K segment select bit ( $256 \times 4$  PROM applications only) are jumper-configured on the MRV11-AA module.

## 7.5 PROM FORMATTING USING THE QJV11 PROGRAM

### 7.5.1 General

The QJV11 PROM formatter program is a software option that greatly reduces the work required for coding binary patterns for individual PROM chips. Input to the program is object tapes punched in absolute loader format. It will produce and verify PROM tapes and listings for  $256 \times 4$  bit and  $512 \times 4$  bit PROMs for use in the MRV11-AA, and PROM chips in other configurations for special user applications.

Table 7-2 PROM Chip Addressing

| Address* |           | 8  | 7 | 6 | 5 | 4 | 3 | 2 | 1 | ←Address- (DAL) Bits   |
|----------|-----------|----|---|---|---|---|---|---|---|--|
| Octal    | Binary    | 14 | 1 | 2 | 3 | 4 | 7 | 6 | 5 | ←PROM Chip Pins  |
| 0        | 00000000  | H  | H | H | H | H | H | H | H |  |
| 2        | 00000010  | H  | H | H | H | H | H | H | L |  |
| 4        | 00000100  | H  | H | H | H | H | H | L | H |  |
| 6        | 00000110  | H  | H | H | H | H | H | L | L |  |
| 10       | 000001000 | H  | H | H | H | H | L | H | H |  |
| 12       | 000001010 | H  | H | H | H | L | L | H | L | Actual<br>Logic<br>Levels<br>Required<br>(256 <sub>10</sub> Locations) |
| 14       | 000001100 | H  | H | H | H | L | L | L | H |  |
| 16       | 000001110 | H  | H | H | H | L | L | L | L |  |
| 20       | 000010000 | H  | H | H | H | L | H | H | H |  |
| ◦        |           |    |   |   |   |   |   |   |   |  |
| ◦        |           |    |   |   |   |   |   |   |   |  |
| 774      | 111111100 | L  | L | L | L | L | L | L | H |  |
| 776      | 111111110 | L  | L | L | L | L | L | L | L |  |

\*Address bit 0 is not used, hence, only even-numbered addresses are shown.

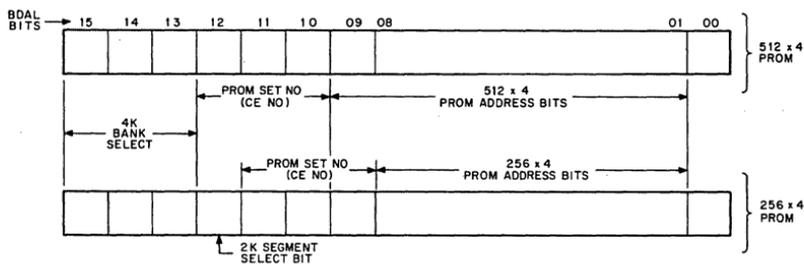


Figure 7-2 MRV11-AA Address Word Format

### 7.5.2 Loading QJV11

QJV11 is supplied on punched paper tape in absolute loader format. Load the program using the Absolute Loader program (DEC-11-UABLB-A-PO) or the REV11-A or REV11-C AL (absolute loader) command.

Hardware requirements include 8K read/write memory (minimum), and either a high-speed paper tape reader (CSR address = 177550) or a low-speed reader (Teletype®) used as the console terminal (CSR address = 177560). A procedure for loading QJV11 is shown in Figure 7-3. If problems are encountered, refer to the more detailed paper tape system operating instructions contained in Section II, Chapter 4. QJV11 is self-

® Teletype is a registered trademark of Teletype Corporation.

starting; when it has been correctly loaded the program automatically starts and the initial message shown in Figure 7-4 is displayed. QJV11 is now ready to receive specific input parameters.

### 7.5.3 Entering Parameters

QJV11 requires certain inputs that must be supplied for each PROM loading session. The dialogue between the QJV11 user and the program is as shown in Figure 7-4; note that this is an example for  $512 \times 4$  PROMs to be used in the MRV11-AA PROM module.

The first parameter to be entered is the number of words (locations) in a PROM. The parameter is requested in the form of a question at the end of the initial message. Operator response to QJV11 requests in Figure 7-4 are underlined. Refer to Table 7-3 for a list of valid parameter inputs for specific applications.

When reading source tapes for MRV11-AA programs that are not greater than 4K, only a single pass of the source tape is required; the QJV11's source buffer is 4K words ( $4096 \times 16$  bits). However, longer programs will require one additional pass for each 4K word buffer storage. The appropriate portion of the program is read into the buffer when reading the source tape as specified by the "starting address of the area to be output." Hence, the starting addresses shown in Table 7-3 are applicable for both multiple-pass programs to specify the starting address for that pass and programs that do not reside in the first 4K of system memory (addresses 0-17776).

The final input to QJV11 is the source program to be loaded into the PROMs. The program must be in absolute loader format. Place the source tape in the tape reader. Press the RETURN key on the console device to initiate tape reading.

Table 7-3 QJV11 Input Parameter

| Parameter   | MRV11-AA Applications                |                                     | Special Applications<br>(not for MRV11-AA use)                              |
|---|--------------------------------------|-------------------------------------|---|
|   | 512 × 4 PROMs                        | 256 × 4 PROMs                       |   |
| No. words in a PROM ( $N_8$ )                     | 1000                                 | 400                                 | Any integer power of two (2000 max.)  |
| No. bits in a PROM word ( $N_8$ )                 | 4                                    | 4                                   | 1, 2, 4, or 10 ( $8_{10}$ )   |
| No. PROMs used in parallel                        | 4                                    | 4                                   | Any number; however, No. bits × No. PROMs must not exceed 20 ( $16_{10}$ ). |
| Are data bits inverted                            | N                                    | N                                   | N or Y  |
| Are addr. lines inverted                          | Y                                    | Y                                   | N or Y  |
| How many bytes in the area to be output ( $N_8$ ) | 20000                                | 10000                               | Any integer power of two (20000 max.)                                       |
| Starting Address                                  | 0, 20000, 40000, 60000, 100000, etc. | 0, 10000, 20000, 30000, 40000, etc. | Any integer multiple of the no. of bytes in the area to be output           |
| I/O device on the H.S. reader/punch               | Y or N                               | Y or N                              | Y or N  |

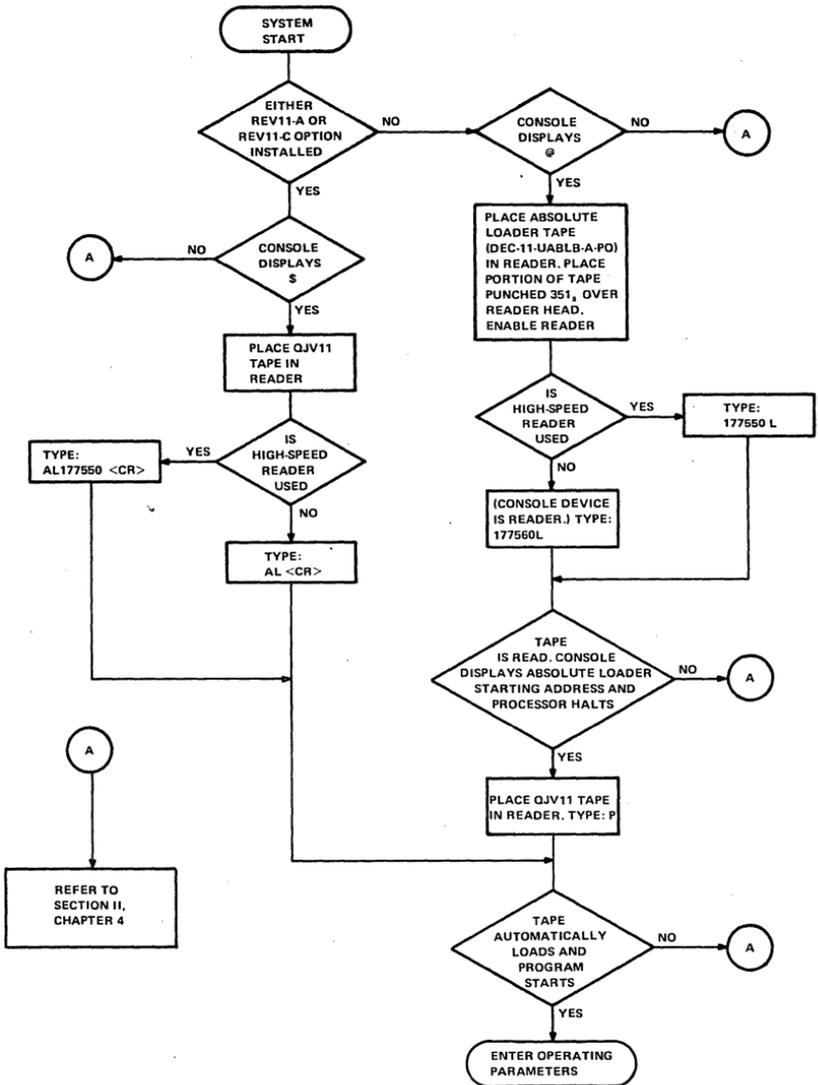


Figure 7-3 Loading QJV11 in LSI-11 and PDP-11/03 Systems

PROM VO1-00

|   |                          |
|---|--------------------------|
| ENTER AN OCTAL VALUE IN RESPONSE TO QUESTIONS WHICH REQUIRE A NUMERIC RESPONSE. TYPE 'Y' FOR YES AND 'N' OR NOTHING FOR NO. TERMINATE ALL RESPONSES WITH A <CR> (CARRIAGE RETURN). RUBOUT MAY BE USED TO DELETE ONE CHARACTER AT A TIME BEFORE <CR> IS TYPED. CTRL/U MAY BE USED TO DELETE THE ENTIRE RESPONSE. CTRL/O MAY BE TYPED TO TURN OFF OUTPUT TO THE TERMINAL.   | }<br>Initial<br>Message  |
| HOW MANY WORDS ARE IN A PROM? 1000<br>HOW MANY BITS ARE IN A PROM WORD? 4<br>HOW MANY PROMS ARE USED IN PARALLEL? 4<br>ARE THE DATA BITS INVERTED? N<br>ARE THE ADDRESS LINES INVERTED? Y<br>HOW MANY BYTES ARE IN THE AREA TO BE OUTPUT? 20000<br>WHAT IS THE STARTING ADDRESS OF THE AREA TO BE OUTPUT? 0<br>IS YOUR INPUT/OUTPUT DEVICE ON THE HIGH SPEED READER/PUNCH? Y<br>READY INPUT, TYPE <CR> WHEN READY. <CR> | }<br>Input<br>Parameters |
| DO YOU WISH TO PUNCH TAPES? Y<br>NO OUTPUT FOR PROM ADDRESS 010000<br>NO OUTPUT FOR PROM ADDRESS 012000<br>NO OUTPUT FOR PROM ADDRESS 014000<br>NO OUTPUT FOR PROM ADDRESS 016000<br>DO YOU WANT TO VERIFY A TAPE? N<br>DO YOU WANT A LIST OF THE PROM CONTENTS? Y<br>DO YOU WANT IT ON A LINE PRINTER? Y<br>DO YOU WISH TO MAKE ANOTHER TAPE? N  | }<br>QJV11<br>Operation  |

Figure 7-4 QJV11 Program Execution (512 × 4 PROMs)

## 7.5.4 QJV11 Operation

**7.5.4.1 General**—Once the input parameters and source program have been entered, the QJV11 is ready to output the desired tapes, listings, or to verify tapes. Operation is simple: respond to QJV11 questions by typing Y or N to indicate the operation(s) desired. The Y answers cause the appropriate QJV11 function to execute immediately. The example shown in Figure 7-4 does not contain the PROM program listing because the separate line printer was selected for the listing. (QJV11 assumes a line printer CSR address = 177514.) If the line printer was not selected, the listing would appear immediately below the listing request.

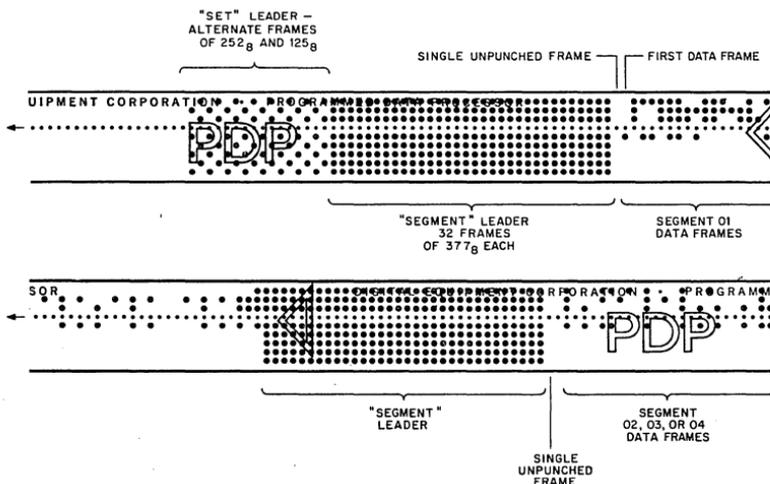
**7.5.4.2 PROM Paper Tape Format**—The QJV11 output tape is punched with as many segments as there are PROMs to be loaded for a particular application. A segment contains the information necessary for loading one 512 × 4 bit or one 256 × 4 bit PROM. Since PROMs are required for the 16-bit LSI-11 word format, four segments are required,

comprising a set. Therefore, the minimum-size QJV11 output would occur when programming a single set of four  $256 \times 4$  bit PROMs.

The actual tape is punched as shown in Figure 7-5. Special alternate punched frames (16 total) identify that a PROM set follows. This area is followed by 32 frames with all frames punched (377<sub>8</sub>), followed by an unpunched frame (0). The first data frame follows immediately after the unpunched frame.

This frame contains the low-order four bits of the 16-bit PROM word at the lowest address (0) in this PROM set; the bits are read over BDALO-3 bus lines. Successive frames contain 4-bit slices, each representing the 4-bit contents of a PROM location. A frame is punched for each of the 256 or 512 locations in the PROM's segment. Frames are punched in high-active PROM address sequence, rather than LSI-11 bus address sequence. (LSI-11 bus address bits are inverted; hence, PROMs are programmed starting at the highest bus address or lowest PROM location address.)

**7.5.4.3 Verifying Tapes**—Tapes punched by OJV11 can be verified by comparing the punched tape with the OJV11's source buffer contents. Respond to the "DO YOU WANT TO VERIFY A TAPE;" request by typing Y <CR>. The program responds with "READY INPUT, TYPE <CR> WHEN READY." Place the tape in the reader and press the RETURN key on the console device.



11-3852

Figure 7-5 OJV11 PROM Tape Format

If an error is found, the program responds with "ERROR VERIFYING TAPE;" when an error is found it is necessary to punch another tape. If errors are not found, the program responds with "DO YOU WANT A LIST OF THE PROM CONTENTS?"

**7.5.4.4 QJV11 PROM Listing Format**—A sample portion of a PROM listing is shown in Figure 7-6. The listing is organized by sets. Each set contains the successive PROM addresses, octal and binary codes for each of the four segments, the system memory address obtained from the absolute loader format source tape, and the octal content of the 16-bit PROM word.

**7.5.4.5 Using QJV11 PROM Tapes**—PROM tapes can be used with automatic PROM loaders, such as those manufactured by DATA I/O Corporation. Refer to the instructions supplied with that equipment for proper use of PROM tapes.

## 7.6 INSTALLING PROMs

After PROMs are properly programmed, loaded, and verified, they can be installed on the properly configured MRV11-AA module. Refer to chapter 5, paragraph 5, for instructions on installing and removing jumpers. Jumpers are used for configuring the module for  $512 \times 4$  or  $256 \times 4$  PROM use, module 4K bank address, and, for  $256 \times 4$  PROM applications, the upper or lower 2K segment selection within a memory bank. In addition, reply jumpers can be removed to disable the MRV11-AA's response to unpopulated PROM set(s) sockets. Refer to Figure 7-7 for proper location of PROM chips. Observe that PROMs are always installed in sets of four—one for each segment. Segment and set numbers correspond to the numbers listed in Figure 7-6.

An addressing summary for PROM sets as arranged by physical locations (CE numbers marked on the MRV11-AA module) is provided in Table 7-4.

**Table 7-4 MRV11-AA PROM Chip Addressing Summary**

| Set No. | Address Range |             |                   |               |           |                   |
|---------|---------------|-------------|-------------------|---------------|-----------|-------------------|
|         | 512 × 4 PROMs |             |                   | 256 × 4 PROMs |           |                   |
|         | Decimal       | Octal       | Physical Location | Decimal       | Octal     | Physical Location |
| 0       | 0-511         | 0-1777      | CE0               | 0-255         | 0-777     | CE0               |
| 1       | 512-1023      | 2000-3777   | CE1               | 256-511       | 1000-1777 | CE4               |
| 2       | 1024-1545     | 4000-5777   | CE2               | 512-767       | 2000-2777 | CE1               |
| 3       | 1546-2047     | 6000-7777   | CE3               | 768-1023      | 3000-3777 | CE5               |
| 4       | 2048-2557     | 10000-11777 | CE4               | 1024-1279     | 4000-4777 | CE2               |
| 5       | 2560-3071     | 12000-13777 | CE5               | 1280-1545     | 5000-5777 | CE6               |
| 6       | 3072-3583     | 14000-15777 | CE6               | 1546-1791     | 6000-6777 | CE3               |
| 7       | 3584-4095     | 16000-17777 | CE7               | 1792-2047     | 7000-7777 | CE7               |

| PROM<br>CHIP<br>ADDRESS  | PROM SET IDENTIFIER |      |    |      | PROM SEGMENT IDENTIFIER |      |    |      | MEMORY<br>ADDRESS | STORED<br>WORD |
|--|---------------------|------|----|------|-------------------------|------|----|------|-------------------|----------------|
|  | FROM #/             | 04   | 03 | 02   | 01                      | 00   | 01 | 02   |                   |                |
| 000  | 00                  | 0000 | 00 | 0000 | 00                      | 0000 | 10 | 1000 | 001776            | 000010         |
| 001  | 01                  | 0001 | 05 | 0101 | 17                      | 1111 | 07 | 0111 | 001774            | 012767         |
| 002  | 00                  | 0000 | 03 | 0011 | 00                      | 0000 | 11 | 1001 | 001772            | 001411         |
| 003  | 02                  | 0010 | 00 | 0000 | 14                      | 1100 | 11 | 1001 | 001770            | 020311         |
| 004  | 00                  | 0000 | 01 | 0001 | 00                      | 0000 | 07 | 0111 | 001766            | 000407         |
| 005  | 00                  | 0000 | 00 | 0000 | 13                      | 1011 | 17 | 1111 | 001764            | 000277         |
| 006  | 10                  | 1000 | 12 | 1010 | 11                      | 1001 | 17 | 1111 | 001762            | 105237         |
| 007  | 01                  | 0001 | 00 | 0000 | 02                      | 0010 | 06 | 0110 | 001760            | 010046         |
| 010  | 00                  | 0000 | 00 | 0000 | 01                      | 0001 | 12 | 1010 | 001756            | 000032         |
| 011  | 00                  | 0000 | 00 | 0000 | 00                      | 0000 | 07 | 0111 | 001754            | 000007         |
| 012  | 01                  | 0001 | 05 | 0101 | 17                      | 1111 | 07 | 0111 | 001752            | 012767         |
| <p style="text-align: center;">OCTAL VALUE OF PROM CONTENTS { OCTAL BINARY</p> |                     |      |    |      |                         |      |    |      |                   |                |
| 775  | 17                  | 1111 | 06 | 0110 | 15                      | 1101 | 16 | 1110 | 000004            | 173336         |
| 776  | 16                  | 1110 | 17 | 1111 | 16                      | 1110 | 06 | 0110 | 000002            | 167746         |
| 777  | 00                  | 0000 | 00 | 0000 | 05                      | 0101 | 17 | 1111 | 000000            | 000137         |
| <p style="text-align: center;">STARTING ADDRESS</p>                            |                     |      |    |      |                         |      |    |      |                   |                |
| <p style="text-align: center;">*** FROM SET 001 ***</p>                        |                     |      |    |      |                         |      |    |      |                   |                |
| FROM #/  | 04                  | 03   | 02 | 01   | 00                      | 01   | 02 | 03   | 04                | 05             |
| 000  | 00                  | 0000 | 00 | 0000 | 00                      | 0000 | 11 | 1001 | 003776            | 000011         |
| 001  | 12                  | 1010 | 05 | 0101 | 15                      | 1101 | 17 | 1111 | 003774            | 122737         |
| 002  | 00                  | 0000 | 01 | 0001 | 11                      | 1001 | 04 | 0100 | 003772            | 000624         |
| 003  | 00                  | 0000 | 01 | 0001 | 11                      | 1001 | 16 | 1110 | 003770            | 000636         |
| 004  | 00                  | 0000 | 03 | 0011 | 01                      | 0001 | 01 | 0001 | 003766            | 001421         |
| 005  | 00                  | 0000 | 01 | 0001 | 00                      | 0000 | 04 | 0100 | 003764            | 000404         |
| 006  | 00                  | 0000 | 00 | 0000 | 00                      | 0000 | 11 | 1001 | 003762            | 000011         |
| 007  | 12                  | 1010 | 05 | 0101 | 15                      | 1101 | 17 | 1111 | 003760            | 122737         |
| 010  | 00                  | 0000 | 02 | 0010 | 01                      | 0001 | 05 | 0101 | 003756            | 001025         |
| 011  | 01                  | 0001 | 17 | 1111 | 17                      | 1111 | 16 | 1110 | 003754            | 017776         |

11-3850

Figure 7-6 QJV11 PROM Listing Format

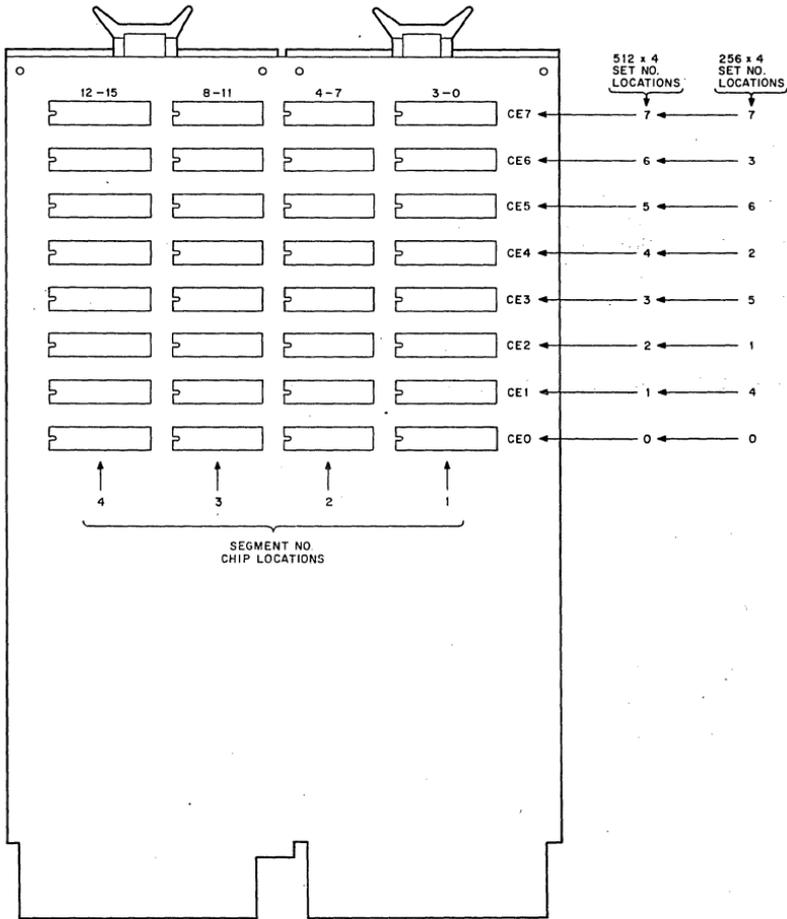


Figure 7-7 PROM Set and Segment Positions on the MRV11-AA Module

## USER-DESIGNED INTERFACES

## 8.1 GENERAL

This chapter contains sample circuits and information which can be utilized in user-designed hardware that is installed on the LSI-11 bus. The user must ensure that the circuit, as used in a particular application, conforms to the LSI-11 bus specifications included in Chapter 3. The various interface module and prewired backplane previously described in this handbook are designed to be easy to use in nearly any application. However, in those applications that require a custom interface, the user can construct the interface using the DRV11-P LSI-11 Bus Foundation module, on which all required bus control and interface logic is supplied, or a special module can be designed by the user. Refer to the *Logic Handbook and Hardware/Accessories Catalog* for a listing of backplanes, wire-wrap modules, blank modules, etc., that will enable system components to be rapidly developed.

## 8.2 BUS RECEIVER AND DRIVER CIRCUITS

The equivalent circuits of LSI-11 bus-compatible drivers and receivers are shown in Figure 8-1. Any device that meets these requirements is acceptable. To perform these functions, Digital Equipment Corporation uses two monolithic integrated circuits with the characteristics listed in Table 8-1. A typical bus driver circuit is shown in Figure 8-2. Note that DEC 8641 quad transceivers can be used, combining LSI-11 bus receiver and driver functions in a single package.

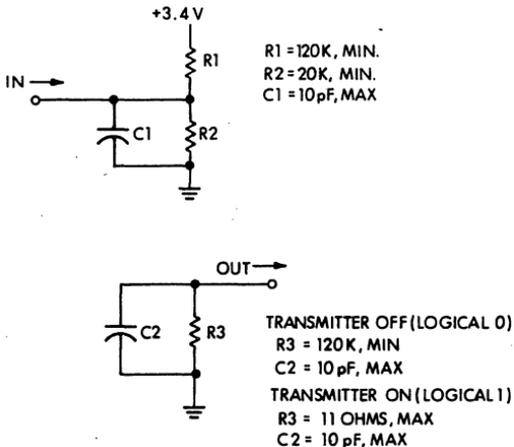
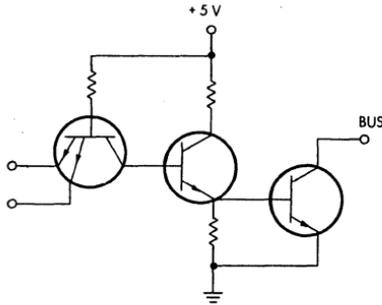


Figure 8-1 Bus Driver and Receiver Equivalent Circuits



TYPICAL BUS DRIVER

11-3307

Figure 8-2 Typical Bus Driver Circuit

Table 8-1 LSI-11 Bus Driver, Receiver, Transceiver Characteristics

|                                     |                                      | Characteristic | Specifications         | Notes |
|-------------------------------------|--------------------------------------|----------------|------------------------|-------|
| Receiver<br>(DEC 8640,<br>DEC 8641) | Input high threshold                 | VIH            | 1.7 V min              | 1     |
|                                     | Input low threshold                  | VIL            | 1.3 V max              | 1     |
|                                     | Input current at 2.5 V               | IIH            | 80 $\mu$ A max         | 1, 3  |
|                                     | Input current at 0 V                 | IIL            | 10 $\mu$ A max         | 1, 3  |
|                                     | Output high voltage                  | VOH            | 2.4 V min              | 2     |
|                                     | Output high current                  | IOH            | (16 TTL loads)         | 2,3   |
|                                     | Output low voltage                   | VOL            | 0.4 V max              | 2     |
|                                     | Output low current                   | IOL            | (16 TTL loads)         | 2,3   |
|                                     | Propagation delay to high state      | TPDH           | 10 ns min<br>35 ns max | 4, 5  |
|                                     | Propagation delay to low state       | TPDL           | 10 ns min<br>35 ns max | 4, 5  |
| Driver<br>(DEC 8881,<br>DEC 8641)   | Input high voltage                   | VIH            | 2.0 V min              | 6     |
|                                     | Input low voltage                    | VIL            | 0.8 V max              | 6     |
|                                     | Input high current                   | IIH            | 60 $\mu$ A max         | 6     |
|                                     | Input low current                    | IIL            | -2.0 mA max            | 6     |
|                                     | Output low voltage                   | VOL            | 0.8 V max              | 1     |
|                                     | Output high leakage current at 3.5 V | IOH            | 25 $\mu$ A max         | 1, 3  |
|                                     | Propagation delay to low state       | TPDL           | 25 ns max              | 5, 7  |
|                                     | Propagation delay to high state      | TPDH           | 35 ns max              | 5, 8  |

## NOTES

1. This is a critical parameter for use on the I/O bus. All other parameters are shown for reference only.
2. This is equivalent to being capable of driving 16 unit loads of standard 7400 series TTL integrated circuits.
3. Current flow is defined as positive if into the terminal.
4. Conditions of load are  $390\ \Omega$  to  $+5\text{ V}$  and  $1.6\text{ k}\Omega$  in parallel with  $15\text{ pF}$  to ground for  $10\text{ ns}$  min and  $50\text{ pF}$  for  $35\text{ ns}$  max.
5. Times are measured from  $1.5\text{ V}$  level on input to  $1.5\text{ V}$  level on output.
6. This is equivalent to 1.25 standard TTL unit loading of input.
7. Conditions of  $100\ \Omega$  to  $+V$ ,  $15\text{ pF}$  to ground on output.
8. Conditions of  $1\text{ k}\Omega$  to ground on output.

Bus receivers and drivers should be well grounded and bypassed with capacitors. They should be located within 4 in. (of etch) from the module fingers which plug into the backplane.

### 8.3 PROGRAMMED INTERFACE

A typical programmed I/O interface is shown in Figure 8-3. Note that only the control logic portion is shown in detail. This circuit is capable of input and output data transfers to and from four addressable data registers in the user's device. In addition, the reply gate will respond to programmed I/O and vector transfers.

Address/data bus interface is provided by DEC 8641 quad unified bus transceiver ICs, keeping component count to a minimum. Note that the DEC 8641 IC at the bottom of Figure 8-3 shows complete address/data I/O signal connections; the remaining DEC 8641s include only the interface signals required for device addressing. However, those ICs will normally be connected for the same type of data I/O interface as shown at the bottom of the figure for bits 0, 13, 14, and 15.

Addressing occurs in the 28-32K address range; BBS7 L is always asserted for this address range. Received data/address bits R3H—R12H and BS7 H are applied to 8136 (address) hex comparator/latch ICs where they are compared to a user-configured device address produced by switches or jumpers. The switches or jumpers must produce high logic levels for logical 1s and low logic levels for logical 0s. The result of the address comparison is latched in each 8136 on the leading edge of BSYNC L. The 8136 outputs will latch for the duration of BSYNC L, producing an active device selected (DEV SEL H) signal. The 74175 hex latch shown in Figure 10-3 latches address bits 0, 1, and 2 on the leading edge of BSYNC L. Bits 1 and 2 encode four unique bus addresses for user-supplied I/O functions. Address bit 0 is a byte pointer which is only used for DATOB or the write portion of DATIOB bus cycles.

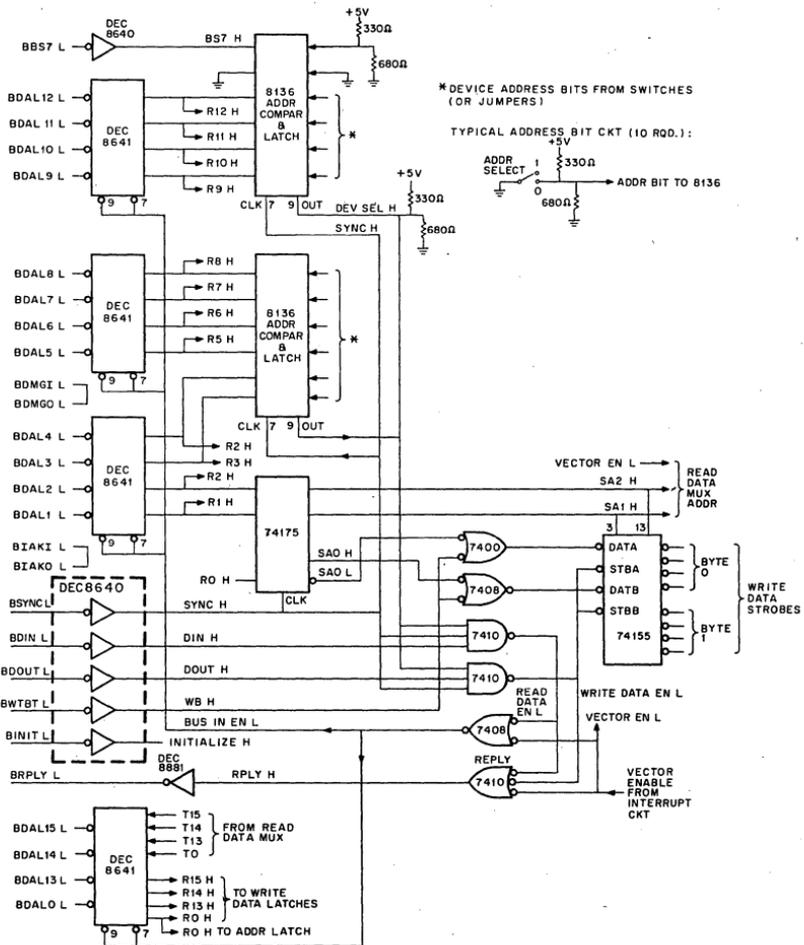


Figure 8-3 Programmed I/O Interface



Read data should be multiplexed using stored address bits SA1 and SA2 H. In addition, an interface circuit that also includes interrupt logic should use VECTOR EN L to inhibit register-read data and enable the interrupt vector transfer during the interrupt sequence.

Write data strobed for the four addressable device registers are produced by a 74155 dual 2:4 demultiplexer; however, other devices and circuits can be used. Both sections of the 74155 are simultaneously strobed by the WRITE DATA EN L signal. During word transfers, WB H is passive (low), enabling the DATA and DATB demultiplexer inputs. As a result of the logical state of stored address bits SA1 H and SA2 H, one Byte 0 and one Byte 1 write data strobe will go active, enabling writing into all 16 bits of the addressed device register. However, when outputting a byte to one of the registers, WB H goes active (high), enabling stored address bit 0 (SA0 H and SA0 L) to assert only one data input (DATA or DATB) on the 74155. Hence, only one of the eight write strobes will go to the active state; an 8-bit transfer to the appropriate high or low byte in the addressed register is thus completed.

#### NOTE

All devices, when addressed, should acknowledge with BRPLY to both BDIN and BDOU T strobes, even if the addressed location is read-only or write-only. This response is required because the LSI-11 processor may generate unnecessary BDIN or BDOU T strobes during execution of certain instructions. For example, the processor may execute a DATIO cycle when only the DATI cycle is required; if BRPLY L is not asserted in response to the BDOU T portion of the DATIO cycle, a bus error (time-out) would occur.

### 8.4 INTERRUPT LOGIC

The basic logic functions required in an interrupt circuit are shown in Figure 8-4. This is a dual interrupt circuit which will enable and control two interrupt request sources (A and B) supplied by the user. The four flip-flops, ENABLE A and B, and INT REQ A and B comprise bits of one or two control/status registers (CSR). The set/reset status of the Enable flip-flops is established by a programmed output transfer. EN A CLK H and EN B CLK H signals are the write data strobes shown in Figure 8-3; EN A DATA H and EN B DATA H would then be two of the received data bits (DEC 8641 "Rn" outputs). Similarly, INT REQ A and B flip-flop outputs INT REQ A and INT REQ B would be read as bits in the CSR via the read data multiplexer in the device's logic.

A typical interrupt sequence for "device A" is described below. An interrupt is enabled under program control by setting the ENABLE A flip-flop. When the user's device is ready for service, it produces an active RQST A H signal, which is ANDed with ENABLE A. The AND gate output clocks the IAK ENB A flip-flop to the set state and IRQA L is produced. Note that if the user's device terminates the RQST A H signal, the IRQA L signal will go low (false), causing BIRQ to go false. IRQA L is ORed with IRQB L and applied to a type DEC 8881 bus driver, asserting the

BIRQ bus signal line. The processor responds by asserting BDIN L, producing a high DIN H signal. This signal clocks the device states (A or B requesting or not requesting service) into the IAK flip-flops. At a later time, the processor asserts BIAKI L, producing a high IAK H signal. IAK H is gated with the IAK flip-flop signals, giving the highest priority to Request A, if both are requesting service. The 7400 gate associated with the IAK A flip-flop Q output goes low, clearing the IAK ENB A flip-flop, and producing VECTOR L and BRPLY L signals. VECTOR L is used for gating the vector address bits onto the I/O bus. With the device's IAK ENB flip-flop clear, it will not generate another interrupt until the device again requests service.

When not requesting service, both Interrupt Acknowledge (IAK) flip-flops remain cleared. The flip-flop Q outputs are both gated with IAK H, producing an active BIAKO L signal which is passed to the next (lower priority) device on the I/O bus. The INIT L signal, produced by a bus receiver and inverter, clears all Enable and IAK flip-flops, and presets (a don't care condition) all INT ENB flip-flops. When requesting service, the IAK flip-flops inhibit passing BIAKO L to the next lower priority device.

#### **CAUTION**

IAK flip-flops must function as synchronizers.  
(Data setup has no guaranteed minimum time.)  
Type 7474 and 74S74 are preferred.

### **8.5 DMA INTERFACE LOGIC**

A simple DMA request circuit is shown in Figure 8-5. In addition to this circuit, bus address, word count, control/status registers, and burst transfer control logic would normally be included. All registers would be accessible via programmed I/O operations.

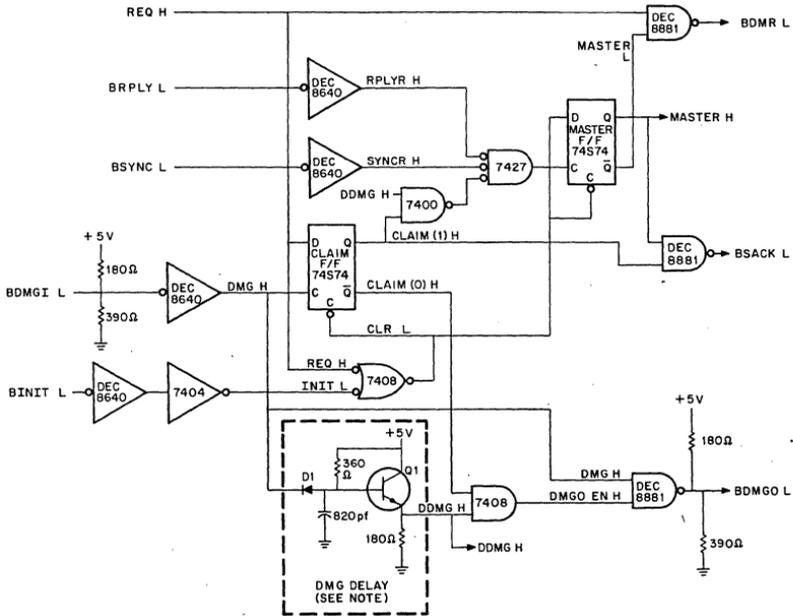
A DMA request is initiated by a device by producing an active REQ H signal. The RQST H signal must remain high until bus mastership is no longer required. The type DEC 8881 bus driver then asserts BDMRL.

The processor arbitrates the request by asserting BDMGI L, setting the Claim flip-flop in the first requesting device along the BDMG daisy chain. The state of the Claim flip-flop is sampled by two gates after the DMG delay. CLAIM (0) H is low (false) and it inhibits the DMGO EN H gate. Hence, when the Claim flip-flop is set, BDMGO L is not passed to lower priority devices. The active (high) CLAIM (1) H signal is gated with DDMG H producing a low signal which enables one of the three 7427 gate inputs. When BSYNC L and BRPLY L become negated, passive (low) SYNCR H and RPLYR H signals are gated with CLAIM (1) H and the 7427 output goes high. This transition clocks the Master flip-flop to the set state producing the active MASTER H signal, enabling BSACK L and negating BDMR L signals. MASTER H is used by the DMA device to enable its bus cycle. BSACK L informs the processor that the bus is in use. At the end of the bus cycle, the device negates REQ H, clearing the Claim and Master flip-flops. MASTER H and BSACK L signals then go passive.

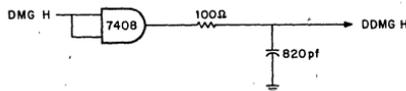
When not requesting DMA service, the device must pass BDHG signals to lower priority devices on the I/O bus. The active (high) CLAIM (0) H

signal is gated with DDMG H producing an active DMGO EN H signal. This signal enables the BDMGO L bus driver and DMG H is gated onto the bus.

The actual DMG delay is determined by the RC circuit shown on the figure, and should be 100 ns (min). BINIT L initializes the circuit by clearing the Claim and Master flip-flops.



NOTE:  
The DMG Delay Circuit shown above is preferred. However, the following DMG Delay Circuit can be used:



CP-1795

Figure 8-5 DMA Arbitration Logic

If dynamic MOS memory is used in the system (KD11-F processor and/or MSV11-B memory), a DMA device is restricted to one bus cycle for each BDMG signal from the processor. This must be done to allow the processor to execute memory refresh transactions. In systems which include dynamic MOS memory and use more than one DMA device, the DMA interface designer *must* ensure that sufficient time will be allowed for the processor to execute memory refresh transactions. As a general

rule, to ensure that the processor performs normal memory refresh, the DMA device should not occupy the I/O bus more than 200  $\mu$ s out of any 400  $\mu$ s period. If the REV11-A or REV11-C DMA refresh option is used in the system, it must be allowed to obtain the bus for one cycle every 27  $\mu$ s.

#### **CAUTION**

The Claim flip-flop must function as a synchronizer. (Data setup has no guaranteed minimum time.) Types 7474 and 74S74 are preferred.



# MAINTENANCE

### 9.1 GENERAL

This chapter contains general information that will aid the user in loading and running diagnostic software. No attempt is made in this chapter to include information that will allow the user to repair modules that fail diagnostic tests. The user can return modules to DIGITAL for repair as directed in Section V, Chapter 4.

In addition to using diagnostic programs for locating system hardware faults, the programs can be used as an assurance test to ascertain that modules, as received, function properly. Diagnostic programs are supplied on punched paper tape or on floppy disk. Both diagnostic software options include the required documentation (program listings and operating instructions).

Paper tape diagnostic programs are contained in option model No. ZJV01-RB. Minimum hardware requirements include the LSI-11 processor and 4K memory, a console terminal, cable and DLV11 interface, and a paper tape reader. The paper tape reader can be either a high-speed reader and interface (user-supplied), or a modified ASR-33 Teletype. DIGITAL can supply a modification kit for the ASR-33 (specify model No. LT33-MB); machines modified by DIGITAL are also available (contact your nearest DIGITAL sales office for more information and prices).

Diagnostic programs contained on floppy disk are available in option model No. ZJ215-RY. Minimum hardware requirements include the LSI-11 processor and 8K memory, a console terminal, cable and DLV11 interface, and the RXV11 floppy disk system.

### 9.2 OPERATIONAL CHECKLIST

#### 9.2.1 General

Many hardware problems encountered with complex systems are a result of operating procedures or improper system configuration, rather than an actual hardware failure. These problems are easily overlooked and sometimes are obvious when observed by a second person. Most of these problems become apparent when the system is first turned on. If a problem seems to occur, check the items listed in the following paragraphs.

#### 9.2.2 LSI-11 System

- Are all power supplies turned on? Check power at the backplane terminals to ensure that +5 V and +12 V power is applied. The system should produce an appropriate power-up response (Section II, Chapter 1) when the user-supplied Initialize switch (Paragraph 6.7) is pressed and released.

- Is the system properly configured? Check:
  1. All modules are correctly plugged into the backplane to ensure proper interrupt and DMA priority daisy chain. No empty slots should exist between modules. Refer to Figure 6-1 and 6-3 for correct installation sequence. If spurious processor halts occur, use the ODT "M" command to determine the cause of the halt (Section II, Chapter 2).
  2. Console DLV11 serial line unit jumpers for compatibility with the console terminal.
  3. Console terminal controls (baud rate, line/local, etc.).
  4. Processor module power-up mode:
    - Mode 0 is normally used with core or PROM memory.
    - Mode 1 can be used with any system hardware configuration. Power-up response is the @ symbol prompt character display on the console terminal.
    - Mode 2 is normally used with REV11-A, REV11-C, or other options containing programs in non-volatile memory, starting at location 173000.
    - Mode 3 is for special processor microcode (not implemented—do not use).
- Are signal cables properly installed? Check that cables are properly connected at each end.
- Can the console terminal be operated in the "local" mode? If the console can be operated independent of the LSI-11 system, it is likely that the terminal is operating properly.

### 9.2.3 PDP-11/03 System

The same items, in general, as listed for the LSI-11 system in Paragraph 9.2.2 also apply to the PDP-11/03 system. However, lights and switches on the PDP-11/03 front panel will aid in checking system operation.

- Is dc power on? The DC ON indicator will be lit only if +5 V and +12 V power supply circuits are operating and the DC ON/OFF switch is in the ON position. If the indicator is not lit, check:
  1. The DC ON/OFF switch is in the ON position.
  2. The AC ON/OFF switch on the rear of the power supply in the PDP-11/03 is turned on. This switch is normally left in the ON position and application of AC power is controlled by an external power control (user-supplied).
  3. AC power is available to the PDP-11/03.
  4. The line power fuse on the rear of the PDP-11/03's power supply. If it is blown, replace it. If it blows again, do not replace it; a power supply failure has probably occurred.
- Is the HALT/ENABLE switch in the HALT position? If it is, the normal console terminal display is the @ symbol. Other power-up displays are possible only when the switch is in the ENABLE position.
- Is the system properly configured? Check the items listed in Paragraph 9.2.2.

- Are signal cables properly installed? Check that cables are properly installed and connected at each end.
- Can the console terminal be operated in the "local" mode? If the console terminal can be operated independent of the PDP-11/03 system, it is likely that the terminal is operating properly.
- Is the LTC switch in the ON position? If it is, software supporting the event interrupt via vector location 100 must be provided. Otherwise, the switch must remain in the OFF position.

### 9.3 USING PAPER TAPE DIAGNOSTICS

#### 9.3.1 General

LSI-11 system diagnostic programs are supplied on punched paper tape in the ZJV01-RB software option. The diagnostic programs are capable of providing a rigorous test of processor memory and interface modules. They verify normal system operation, or identify specific fault symptoms. Minimum system hardware requirements for running the diagnostic programs include the LSI-11 processor, a console device (including interface module at address 177560), 4K read-write memory (semiconductor or core), a paper tape reader (LT33 teletype-writer or a high-speed reader), and diagnostic software. The diagnostic software includes an absolute loader and several diagnostic program tapes and listings. The following paragraphs briefly describe each program and provide a general guide for loading and using.

Diagnostic program abstracts are provided below. Each diagnostic program listing includes a detailed description and information for loading and using the program.

#### **MAINDEC-11-DVKAA-A-D**

LSI-11 Basic Instruction Tests—This program tests the LSI-11 basic instruction set in all addressing modes.

#### **MAINDEC-11-DVKAB-A-D**

LSI-11 Extended Instruction Tests—This program tests the extended arithmetic instructions ASH, ASHC, MUL, and DIV using general registers R0-R5 at least once with each instruction. This diagnostic program can only be used with LSI-11 processors in which the KEV11 Extended Arithmetic Chip option is installed.

#### **MAINDEC-11-DVKAC-A-D**

LSI-11 FIS Instruction Tests—This program tests the floating instructions FADD, FSUB, FMUL, and FDIV. It uses fixed number patterns and each general register at least once as the stack pointer. It also checks stack overflow and that floating instructions can be interrupted by the console device (if enabled by an operator option in the program). This diagnostic program can only be used with LSI-11 processors in which the KEV11 Extended Arithmetic Chip option is installed.

#### **MAINDEC-11-DVKAD-A-D**

LSI-11 Trap Tests—This program tests all operations and instructions that cause traps, oddities of the SP (R6), interrupts, RESET, and WAIT instructions.

### MAINDEC-11-DVKAE-A-D

DLV11 Test—This program tests the logic functions of the DLV11 Serial Line Unit. The program is supplied with device addresses and vectors for DLV11 use as the console device. However, the operator can easily alter device address and vector assignments for the particular DLV11 module being tested.

### MAINDEC-11-DVKAF-A-D

DRV11 Test—This program tests the logic functions of the DRV11 Parallel Line Unit. The program is supplied with device address and vector assignments of 167770 and 300, respectively. The operator can easily alter these values for the particular DRV11 module being tested. When using this program, a special maintenance cable (BC08R) must be connected to the module. NEW DATA READY and DATA TRANSMITTED signals generated by the DRV11 are not tested by this program; they can be checked by external hardware only.

### MAINDEC-11-DVKAG-A-D

LSI-11 4K Read/Write Memory Tests—This program tests 4K semiconductor and core read/write memory. Worst case tests (long galloping for semiconductor memory and worst case noise for core memory) are included in the program. This program will run in (and test) read/write memory systems of 4K or larger.

#### 9.3.2 Loading Diagnostic Program Tapes

Program tapes are loaded into memory by the absolute loader tape supplied with the diagnostic tapes, or by using the REV11-A or REV11-C option AL command. Load the tapes as directed in Section II, Chapter 4.

#### 9.3.3 Program Modification Procedure

Prior to executing diagnostic program tapes, the user must observe the operating procedure stated in the diagnostic program listing. This procedure generally includes a procedure for setting software switch register bits, simulating switch register bit positions which are available in other PDP-11 systems that include a hardware switch register. In addition, entering a device address may be required, especially when testing a device interface module that is configured to a device address other than the factory-supplied configuration.

When setting software switch register bits, determine the octal value required for the switch register word. This can be determined by adding the listed values for the desired program options as directed in the program listing. For example, the listing may state that the software switch register is located at address 122, and you may conclude that the following bits should be set:

| Bit | Octal Value |
|-----|-------------|
| 15  | 100000      |
| 10  | 002000      |
| 9   | 001000      |

The octal sum to be entered is 103000. Enter the value from the above example as follows:

```
@122/000000 103000 CR LF  
@
```

where @ is the prompt character typed out by the console ODT micro-code; 122 is the address of the software switch register to be opened; / is the "open location" command; 000000 is the preset contents of location 122; 103000 is the new software switch register value to be entered; CR (carriage RETURN key) enters the new value and closes the location; LF (line feed) @ is printed and a new console ODT command can be entered.

Other program values may be altered, such as device addresses or vector addresses, by using similar ODT commands shown for altering the software switch register. Refer to the diagnostic listing to obtain the procedure for these program changes.

### 9.3.4 Program Starting and Execution

Once a program has been loaded correctly, the Run mode can be enabled and program execution started. Place the HALT/ENABLE switch (PDP-11/03 panel, or equivalent user-supplied LSI-11 switch) in the ENABLE position. Start normal program execution as follows:

```
@200G
```

The 200 in the above example is a typical starting address. Each program listing specifies the correct starting address. G is the Go command, and program execution will immediately commence starting at the specified location.

Single instruction execution, when desired, is obtained by operating the processor in the Halt mode. Place the HALT/ENABLE switch in the HALT position. Enter the starting address (or a desired address for the first instruction to be executed) as described for normal program execution. The G command causes the processor to execute the first instruction and then Halt with the PC (R7) pointing to the next program instruction to be executed. This address is printed on the console device as shown in the following example:

```
@200G CR LF  
000202 CR LF  
@
```

Successive instruction executions will occur each time the Proceed (P) command is entered via the console device, as shown below:

```
@200G  
000202  
@P  
000204  
@P  
000206  
@
```

Note that after executing each instruction, the processor halts and prints the address of the next instruction. Thus, Branch, Jump, and Skip instruction will alter the PC, as required in normal program execution, allowing the operator to observe program and hardware operation.

### 9.3.5 Diagnostic Program Results

Basic results from running diagnostic programs range from pass results to error conditions. Error conditions may result in the program halting and printing out an error message or simply a PC value (the error Halt PC+2). The use of error information thus obtained is completely described in the program listing of the diagnostic program in use. As the operator becomes more familiar with the use of the diagnostic program, it may be desirable to invoke certain program options, as described in the listing, such as loop on error. This allows the operator to run (or single instruction execute) the program in an area where a hardware error is detected.

The key to effective use of diagnostic programs is in the ability of the operator to read the diagnostic listing (especially the comments associated with program instructions that do not execute properly), and relate the error conditions to hardware functions. Particular locations in the program can be examined to obtain much useful information about the fail conditions. Tags (symbolic labels) are associated with these locations, such as \$FATAL, \$MAIL, \$TESTN, etc. Tags and their individual functions vary somewhat in different diagnostic programs; refer to the program listings for a complete listing of tags.

Pass conditions generally result in a printout of the first pass and printouts at certain multiples of passes that follow. A pass counter is located in each program that can be examined by the operator. A typical tag for the pass counter is \$PASS. Intermittent errors can be detected by observing the contents of that location when errors are detected.

Program operation can be halted and restarted by the operator at any time. Restart at the beginning of the program, as previously described, or continue from the point at which the program halted by using the P command.

## 9.4 USING RXDP DIAGNOSTICS

RXDP diagnostics are contained on floppy diskette and provide the required diagnostics for testing processor, memory, and interface modules, the RXV11 floppy disk system, and certain peripherals. Documentation supplied with the RXDP software includes program listings and an *XXDP User's Manual*. A listing of RXDP diagnostic programs applicable to LSI-11, PDP-11/03, and PDP-11V03 systems is provided.

### 9.4.1 RXDP Diagnostics

#### MAINDEC-11-DVKAH

Basic System Exerciser—Tests serial line unit, memory, processor, EIS/FIS, clock, and both floppy disk drives under dynamic interactive conditions. Provides simple and rapid preliminary confidence check of overall system operation.

#### MAINDEC-11-DVKAA

LSI-11 Basic Instruction Tests—Exercises and tests the LSI-11 basic instruction set in all addressing modes.

#### MAINDEC-11-DVKAB

LSI-11 Extended Instruction Set (EIS) Tests—Exercises and tests the extended arithmetic instructions ASH, ASHC, MUL, and DIV using gen-

eral registers R0-R5 at least once with each instruction. This diagnostic program can only be used with LSI-11 processors in which the KEV11 Extended Arithmetic Chip option is installed.

#### **MAINDEC-11-DVKAC**

LSI-11 Floating Instruction Set (FIS) Tests—Exercises and tests the floating instructions FADD, FSUB, FMUL, and FDIV. Uses fixed number patterns and each general register at least once as the stack pointer. Also checks stack overflow and confirms that floating instructions can be interrupted by the console device (if enabled by an operator option in the program). This diagnostic program can only be used with LSI-11 processors in which the KEV11 Extended Arithmetic Chip option is installed.

#### **MAINDEC-11-DVKAD**

LSI-11 Trap Tests—Exercises and tests all operations and instructions which cause traps, oddities of the SP (R6), interrupts, RESET, and WAIT instructions.

#### **MAINDEC-11-DVKAE**

DLV11 Test—Exercises and tests the logic functions of the DLV11 Serial Line Unit. The program is supplied with device addresses and vectors for DLV11 use as the console device. However, the operator can easily alter address and vector assignments for the particular DLV11 module being tested.

#### **MAINDEC-11-DVKAF**

DRV11 Test—Exercises and tests the logic functions of the DRV11 Parallel Line Unit. The program is supplied with device address and vector assignments of 167770 and 300, respectively. The operator can easily alter these values for the particular DRV11 module being tested. When using this program, a special maintenance cable (BC08R) must be connected to the module. NEW DATA READY and DATA TRANSMITTED signals generated by the DRV11 are not tested by this program; they can be checked by external hardware only.

#### **MAINDEC-11-DZKMA**

Memory Exerciser—Tests system memory from the basic 8K (required for floppy disk system) to the maximum 28K.

#### **MAINDEC-11-DZLAC**

LA36 Diagnostic—Under software register control, executes tests on four basic categories of LA36 DECwriter functions:

1. Printing. Tests LA36 printing mechanism and associated control logic.
2. Echo. Tests keyboard as an aid in isolating faults within the terminal. Results are visually verified.
3. Options. Permits exercising LA36 options in whatever combination they occur. Results are visually verified.
4. Standard I/O function. Tests system/terminal interface logic. Processor halts in response to errors.

#### **MAINDEC-11-DZRXA**

RX11 System Reliability Check—Composed of selectable tests that write, read, and verify various data patterns under various (selectable)

head movements. Data transfer and error check can be performed over entire diskette or between separately selectable track and sector limits.

#### **MAINDEC-11-DZRXB**

Runs series of selectable RX11 Interface Diagnostics—Tests basic functions of RX11 interface. Errors are reported by the program, and it is possible to loop on the error or a particular test for scope testing.

#### **MAINDEC-11-DZVTC**

VT52 Diagnostic—Tests all characters and commands except for BREAK, REPEAT, AUTO-PRINT, and SCROLL. Program is divided into four parts:

1. Presents a series of test patterns on VT52 screen. Operator visually scans each pattern for errors.
2. Tests keyboard characters input by operator (from displayed instructions) to determine if terminal is generating valid ASCII codes.
3. Displays octal values and printable symbols corresponding to keys depressed by operator.
4. Echoes on the screen characters whose keys the operator has pressed.

#### **MAINDEC-11-DZM9A**

Bootstrap Terminator—Exercises and tests to verify the ROM contents of REV11-A and REV11-C modules. The program computes and checks a cyclic redundancy character and a longitudinal parity character for the contents of the ROM storage.

Error information consists of the following:

1. Location at which error was detected.
2. Bus address of VT52 under test.
3. Test pattern number of failing test.
4. Expected input character.
5. Received input character.

In addition to the diagnostic programs, the RXDP diskette typically contains the following programs:

|       |   |
|-------|---|
| RXDP  | Monitor   |
| UPD1  |   |
| UPD2  | Update programs that allow patching of diagnostics                  |
| COPY  | Copies and verifies entire diskette                                 |
| XTECO | An ASCII editor that provides for creating and updating ASCII files |

See *XXPD User's Manual* for detailed information.

Minimum hardware requirements include an LSI-11 processor, 8K (total) minimum read/write memory, RXV11 floppy disk system, and a console device.

#### **9.4.2 Program Modification and Execution**

The RXDP diskette can be run in either RXV11 disk drive (DX0 or DX1). Insert the diskette in the appropriate drive and bootstrap the system; instructions for bootstrapping the RXV11 are included in Section II,

Chapter 5. Bootstrapping the system causes the RXDP monitor to identify itself on the console device, followed by a restart address and the monitor prompt character (.). A typical display is shown below:

```
RXDP-XXDP RX11/RX01 Monitor M-11-DZQUJ-B 21-FEB-76 8 K
RESTART ADDR: 032260
```

A help message describing RXDP commands is also output. Any or all of the above messages can be aborted by CTRL/C (depressing CTRL and C keys simultaneously on the console device).

#### NOTE

The operator can abort program execution and return to the RXDP monitor at any time by simultaneously pressing CTRL and C keys on the terminal.

Once the disk monitor has been called successfully, the operator can proceed to run pertinent diagnostic programs. To precisely confirm the programs available on the diskette, the operator can issue the command

```
_ D <CR>
```

The period is the disk monitor prompt character and is underlined here, as elsewhere in this text, to indicate that it is generated by the system rather than the user. "D" requests that the disk directory be listed; "<CR>" (carriage return) is the RXDP Monitor execute operator. The system responds by outputting a list of all entries in the disk directory. This list should correspond with that packed with the diskette.

The normal procedure for running individual diagnostic programs once the RXDP Monitor prompt character (.) has been issued by the system is

```
_ R PROGRAM <CR>
```

where PROGRAM is the name of the RXDP diagnostic (VKAAAO, VKABAO, etc.) to be run. This will result in the program's being transferred from disk to memory and run. If there are no faults, the system will output the "END PASS" message at various intervals of from 3 seconds and longer, depending on the particular diagnostic program in use, indicating that the program has completed a test cycle and another has begun. This operation will continue until the operator halts the processor. To restart the RXDP monitor, enter the following command on the console device:

```
@032260G
```

where "@" is the ODT prompt character issued by the system, "032260" is the restart address output by the RXDP monitor when the system was bootstrapped, and "G" is the ODT GO command.

Occasionally, the user may wish to set initial conditions differently from those assumed by a given program—to accommodate new device register addresses, for example. This is accomplished as follows:

1. After receiving RXDP Monitor prompt character (.), type "L PRO-

## Sample Directory Printout:

.D

| ENTRY# | FILNAM,EXT | DATE      | LENGTH | START  |
|--------|------------|-----------|--------|--------|
| 000001 | RXDP .BIN  | 17-MAR-76 | 17     | 000050 |
| 000002 | UPD1 .BIN  | 17-MAR-76 | 17     | 000071 |
| 000003 | UPD2 .BIN  | 17-MAR-76 | 29     | 000112 |
| 000004 | COPY .BIN  | 17-MAR-76 | 24     | 000147 |
| 000005 | XTECO .BIN | 17-MAR-76 | 27     | 000177 |
| 000006 | VKAHA0.BIC | 17-MAR-76 | 17     | 000232 |
| 000007 | VKAAA0.BIC | 17-MAR-76 | 17     | 000253 |
| 000010 | VKAFA0.BIN | 17-MAR-76 | 6      | 000274 |
| 000011 | VKAEA0.BIN | 17-MAR-76 | 5      | 000302 |
| 000012 | VKABA0.BIC | 17-MAR-76 | 17     | 000307 |
| 000013 | VKACA0.BIC | 17-MAR-76 | 16     | 000330 |
| 000014 | VKADA0.BIC | 17-MAR-76 | 12     | 000350 |
| 000015 | ZKMA0.BIC  | 17-MAR-76 | 9      | 000364 |
| 000016 | ZM2A0.BIC  | 17-MAR-76 | 6      | 000375 |
| 000017 | ZRXA0.BIC  | 17-MAR-76 | 19     | 000403 |
| 000020 | ZRXB0.BIC  | 17-MAR-76 | 15     | 000426 |
| 000021 | ZLACC0.BIN | 17-MAR-76 | 16     | 000445 |
| 000022 | ZVTCC1.BIN | 17-MAR-76 | 22     | 000465 |

FREE FILES: 94  
FREE BLOCKS: 163

GRAM," where "PROGRAM" is the name of the diagnostic to be modified.

2. Halt the processor. The console device will display the ODT @ prompt character.
3. Determine from XXDP manual and program listing which memory address(es) must be modified and perform modifications using ODT commands. Use the general procedure described in Paragraph 9.3.3 for RXDP diagnostic program modifications.
4. Execute modified program by typing "200G" in response to last ODT "@" prompt character.

Changes can be permanently made by using UPD1, as described in Paragraph 9.4.7.

### 9.4.3 Single Instruction Execution

Single instruction execution, when desired, is obtained by operating the processor in the Halt mode. After receiving the RXDP monitor prompt character (.), load program with the "L" command. Place the HALT/

ENABLE switch in the HALT position. Enter the starting address (or a desired address for the first instruction to be executed) as described for normal program execution. The "G" command initializes the system bus; thereafter, the program can be executed one instruction at a time by repeatedly depressing "P" (Proceed) on the console device. An example of single word instruction execution is shown below:

```
@200G  
000202  
@P  
000204  
@P  
000206  
@P
```

Note that after executing each instruction, the processor halts and prints the address of the next instruction. Thus, Branch, Jump, and Skip instructions will alter the Program Counter (PC) as required in normal program execution, allowing the operator time to observe program and hardware operation.

#### 9.4.4 Diagnostic Program Results

Basic results from running diagnostic programs range from "end of pass" indications to error conditions. Error conditions may result in the program displaying an error message or simply entering Halt mode. This causes ODT to display the contents of the PC which will define the address of the Halt (+2) and thereby permit ascertaining what the program was testing when it issued the Halt. The use of error information thus obtained is completely described in the listing of the diagnostic program in use. As the operator becomes more familiar with the use of the diagnostic program, he may wish to invoke certain program options, described in the listing, such as loop on error. This allows the operator to run (or single instruction execute) the program in an area where the hardware error is detected or anticipated.

Effective use of diagnostic programs rests on the ability of the operator to read the diagnostic listing (especially the comments associated with program instructions that do not execute properly), and to relate the error conditions to hardware functions. Particular locations in the program can be examined to obtain much useful information about the fail conditions. Program operation can be halted and restarted by the operator at any time. Restart at the beginning of the program, as previously described, or continue from the point at which the program halted by using the P command.

#### 9.4.5 Running a Chain of Diagnostics

Running a chain of diagnostics is also called running a script of diagnostics. Several RXDP diagnostics may be strung together and run in automatic sequence as described in the following procedure:

#### NOTE

Only those diagnostics with a ".BIC" extension are chainable and may be included as part of a chain file.

1. Before a chain can be run, an ASCII chain file must be created.
  - a. In response to the RXDP monitor's "." prompt character, type "R XTECO<CR>".
  - b. XTECO will be brought into memory, a heading will be displayed, and the date will be requested; after the date has been input and echoed,\* XTECO will type the "\*" prompt character.
  - c. To create an ASCII test file, type "TEXT DX0: program name CCC<CR>".

\*Format for RXDP date is DD-MON-YY, e.g., 06-MAR-76 for March 6, 1976.

#### NOTE

The "n" "DXn" will be either 0 or 1, depending on whether the file is to be output on drive 0 or drive 1.

- d. XTECO will ask if the output file is ready. If Disk Drive 1 has been specified, insert disk in that drive and type <CR>; otherwise, just type <CR>.
- e. To enter data in the file, type "I" (no carriage return), and the desired ASCII data. Data on a line that is preceeded by a semi-colon (;) is interpreted as a comment. Legal commands that are input as part of the script are RXDP monitor level commands such as Run (R), Load (L), Start (S), or Chain (C). When done inputting, press the ESC key (or ALT MODE) (echoed as "\$") twice for execution. XTECO will respond with an "\*" prompt.

|            |  |
|------------|--|
| Example:   | Assume that the RXDP diskette includes:                      |
| TEST1 .BIC | (self-starting program)                                      |
| TEST2 .BIC | (not self-starting)  |
| TEST3 .BIN | (not chainable—shown by ".BIN" extension instead of ".BIC".) |
| TEST4 .BIC | (self-starting program)                                      |

In addition, assume that the test objective is to run TEST1 three times, followed by TEST2 seven times, followed by TEST4 once,

```
* I; this is an example
R TEST1/3
L TEST2
S TEST2/7
R TEST4
C CHAIN
$ $
*EX$ $
*BOOT DX0:
```

- f. To put the actual file onto the diskette, answer the "\*" with an "EX\$\$". (Note: \$ means ALT MODE.) XTECO will respond (after a writing/verifying interval) with "\*". The chain file is now complete.
2. To run the chain file, exit XTECO and re-enter RXDP monitor by typing "BOOT DX0:<CR>". Since the RXDP monitor was rebooted, the initial heading and help message will be printed and then the

“.” prompt. To run the chain file type “C filename<CR>” (in the example, filename would be CHAIN). To do a quick verify mode (i.e., run through each diagnostic listed in the file just once regardless of what the file says) type “C filename/QV<CR>”.

#### 9.4.6 Making a Duplicate of the Diskette

Making a duplicate of the RXDP diskette allows safe storage of the “master” and using a copy for normal RXDP use. This will protect the user from accidental loss of the diagnostic. Produce a copy as directed below:

1. Install a diskette in the remaining disk drive unit. This diskette will become the RXDP copy.
2. Respond to RXDP Monitor “.” prompt with “R COPY<CR>”.
3. The COPY program will be brought into memory and then will type out a heading and issue an “\*” prompt character.
4. Type “COPY DXd:=DXs:<CR>” (d = destination disk drive number; s = source disk drive number). The copy program will copy the entire diskette and verify the transfer.

#### NOTE

This process takes about 20 minutes.

**Example:** Disk Drive 0 contains the RXDP diskette and disk drive 1 contains a blank diskette.

\*COPY DX1: = DX0:<CR>

5. If return to the RXDP monitor is desired, type “BOOT DX0:<CR>”.

#### 9.4.7 Diagnostic Program Changes

Permanent changes to an RXDP diagnostic program can be made using the UPD1 program. Proceed as directed below:

1. Start the RXDP program as directed in Paragraph 9.4.2. In response to the “.” prompt character, run the UPD1 program by entering the following command:

\_R UPD1<CR>

2. UPD1 becomes loaded into system memory and starts automatically. UPD1 then requests the date. Enter the date in the following format:

DD-MON-YY

where DD is a two-digit day, MON is a three letter month, and YY is a two-digit year. The program responds by displaying the date and the “\*” prompt character.

3. Bring the program to be changed into memory by entering the following command:

\*LOAD DXN:program name.extension

Specify the disk drive (DXn) by entering DX0 for drive 0 and DX1 for drive 1.

#### NOTE

In the remainder of this procedure it is assumed that DX0 is in use. If DX1 is used, specify the drive as directed above.

4. After the program has been loaded into memory, UPD1 displays the "\*" prompt character. Specify the memory location to be modified (nnnnn) by entering the following command:

```
*MOD nnnnn<CR>
```

5. UPD1 responds by displaying

```
nnnnn xxxxx
```

where nnnnn is the address of the opened location and xxxxx is the present contents of that location. Enter the new (changed) octal value (yyyyy) for the location and terminate the command with CR or LF. LF closes the changed location and opens the next location in memory and displays its address and contents on the next line; that location can then be modified, as above. CR closes the changed location and returns the program to the command mode; UPD1 displays the "\*" prompt character on the next line. The complete command for changing one location is shown below:

```
nnnnn xxxxx yyyyy<LF>
```

6. Make additional changes, as necessary, using the procedure contained in steps 4 and 5. Use the CR to close the last changed location.
7. To store the modified program on diskette using its original name, first delete the unmodified version from the diskette by entering the following command (DX0 is assumed, see note in step 3):

```
*DEL DX0:program name.extension<CR>
```

8. Store the modified program by entering the following command:

```
*DUMP DX0:program name.extension<CR>
```

9. After storing the modified program, UPD1 displays the "\*" prompt character. Return control to the RXDP monitor by entering the following command:

```
*BOOT DX0:<CR>
```

#### 9.4.8 Creating a Unique RXDP Diskette

A unique RXDP diskette can be generated containing only selected portions of the RXDP programs as directed in the following procedure.

1. Place the RXDP diskette in disk drive 0. Place the blank diskette in drive 1.
2. Start the RXDP program as directed in Paragraph 9.4.2. In response to the "." prompt character, run the UPD1 program by entering the following command:

```
_R PUD1 <CR>
```

3. UPD1 becomes loaded into system memory and starts automatically. UPD1 then requests the date. Enter the date in the following format:

```
DD-MON-YY
```

where DD is a two-digit day, MON is a three letter month, and YY is a two-digit year. The program responds by displaying the date and the "\*" prompt character.

4. Load the RXDP Monitor into memory by entering the following command:

\*LOAD DX0:RXDP.BIN <CR>

5. Copy (save) the Monitor onto the new diskette by entering the following command:

\*SAVM DX1: <CR>

6. Additional RXDP programs can be copied by first loading the desired program into memory and then copying the program onto the new diskette. This must be done for each program desired. An example is shown below for the COPY.BIN program. Wait for UPD1 to respond with the "\*" prompt character before entering new commands:

\*LOAD DX0:COPY.BIN <CR>

\*DUMP DX1:COPY.BIN <CR>





section**2**  
operation

**CHAPTER 1 INTRODUCTION**

**CHAPTER 2 USING CONSOLE ODT COMMANDS**

**CHAPTER 3 USING REV11-A AND REV11-C  
COMMANDS**

**CHAPTER 4 PAPER TAPE SYSTEM OPERATION**

**CHAPTER 5 RXV11 FLOPPY DISK-BASED SYSTEM  
OPERATION**

## INTRODUCTION

### 1.1 GENERAL

All LSI-11 and PDP-11/03 systems use the same basic hardware components. Hence, the operating procedure for both systems is identical. The operator inputs commands, data, and instructions via a console terminal, which also serves as an output device. All console functions previously provided by a control panel containing switches and lights are provided by this terminal, including starting programs, depositing and examining memory and register locations, and, with most terminals, halting program execution. Console device hardware is described later in this chapter.

Bootstrap, loader, and diagnostic programs are included in PROMs contained on the REV11-A and REV11-C options. The bootstrap program allows bootstrapping either drive (DX0 or DX1) in the RXV11 floppy disk system by entering a command on the console device. Loader programs include the Absolute Loader for both absolute addressed and relocatable programs; the loaders can be used with the console device, when equipped with a paper tape reader capability, and a high-speed reader or other reader device at a specified device address. The diagnostic programs perform simple go-no-go tests on the processor and system memory. Diagnostic programs can be accessed by entering appropriate commands, or executed automatically whenever a loader or bootstrap program is accessed. In addition, a non-memory modifying processor test is executed whenever the REV11 option's starting address (173000) is accessed during power up. The remaining chapters in this section describe console device ODT commands (Chapter 2), REV11-A and REV11-C commands (Chapter 3), paper tape system operation (Chapter 4), and RXV11 floppy disk-based system operation (Chapter 5). The remainder of this chapter includes a description of the console device and normal power-up system response for various LSI-11 and PDP-11/03 system configurations.

### 1.2 THE CONSOLE DEVICE

The console device can be nearly any terminal that is capable of transmitting and receiving (for display) ASCII characters. The terminal normally interfaces with the system via a DLV11 serial line unit. Factory-installed jumpers encode the console device addresses and interrupt vectors. When in the console state (Halt mode), all communication between the processor and the serial line unit is controlled by the processor's microcode in all LSI-11 and PDP-11/03 systems. Terminals equipped with a BREAK key are capable of placing the processor in the console state at any time. Note that this use of the console terminal does not interfere with its use for programmed I/O since the console and run states of the processor are mutually exclusive. A factory-installed jumper (FEH) is included on the DLV11 serial line unit to enable

this function. Refer to Section I, Chapter 5, paragraph 5.6, for complete information on configuring the DLV11 for use with specific terminal I/O rates, serial word formats (number of data bits, number of stop bits, parity/no parity, odd, or even parity), and 20 mA current loop or EIA interface. Peripherals that are suitable for use as the console device terminal are listed in Table 1-1. Although the RT02-A is not capable of generating all ASCII characters required for full console operation, it can be used as the console device, but it is limited to the following ODT commands:

| <b>ODT Command</b> | <b>RT02-A Keys</b> |
|--------------------|--------------------|
| CR                 | SEND               |
| LF                 | SHIFT and CLEAR    |
| /                  | SHIFT and ÷        |
| @                  | SHIFT and @        |
| G                  | SHIFT and GO       |
| RO                 | SHIFT and ERROR    |

The console device can either be directly interfaced to the DLV11 or it can be operated in a remote location and interfaced via data sets or DF01-A acoustic couplers and telephone lines. Only the LA36, LT33, VT50, and VT52 are capable of remotely placing the LSI-11 system in the Halt state by asserting a line break (continuous "space" transmission). (This feature is jumper-enabled on the DLV11 through the use of framing error detection.)

### **1.3 POWER-UP RESPONSE**

#### **1.3.1 PDP-11/03 Power-On**

Proceed as follows:

1. Ensure that the system is properly configured as previously described in Section I, Chapter 5.
2. Place the DC ON/OFF switch in the down position (DC OFF).
3. Place the AC ON/OFF switch on the rear of H780 power supply in the AC ON position.
4. Place the HALT/ENABLE switch in the desired power-up position.

#### **NOTE**

The dc power can be applied with the HALT/ENABLE switch in either position. However, processor power-up response is affected by this switch and jumper-selected power-up modes, as listed in Table 1-2.

5. Place the LTC ON/OFF switch in the OFF position.
6. Place the DC ON/OFF switch in the up (DC ON) position. The console device should respond with a printout (or display) as shown in Table 1-2.
7. Proceed with initial power-on checkout by entering and executing the program listed in Paragraph 1.3.3.

**Table 1-1 Console Terminal Options**

| Terminal Type                                | Model  | Name                                  | Use as Console Device         | Display Capacity                      | I/O Speed (baud rate)             | BREAK Key | Serial Interface Type      | Required Interface Options   |
|--|--------|---------------------------------------|-------------------------------|---------------------------------------|-----------------------------------|-----------|----------------------------|------------------------------|
| Keyboard/Printer                             | LA36   | DECwriter 11                          | Yes                           | 132 characters/line                   | 300                               | Yes       | 20 mA loop optional EIA    | DLV11, BC05M<br>DLV11, BC05C |
| Keyboard/Printer and Paper Tape Reader/Punch | LT33   | Teletypewriter                        | Yes                           | 72 characters/line                    | 110                               | Yes       | 20 mA loop                 | DLV11, BC05M                 |
| Keyboard/CRT Display                         | VT05B  | Alphanumeric Terminal                 | Yes                           | 1440 characters (72 char. X 20 lines) | 110-2400                          | No        | 20 mA loop or EIA          | DLV11, BC05M<br>DLV11, BC05C |
| Keyboard/CRT Display                         | VT50   | DECscope                              | Yes                           | 960 characters (80 char. X 12 lines)  | 75-9600                           | Yes       | 20 mA loop or optional EIA | DLV11, BC05M<br>DLV11, BC05C |
| Keyboard/CRT Display                         | VT52   | DECscope                              | Yes                           | 1920 characters (80 char. X 24 lines) | 75-9600                           | Yes       | 20 mA loop or optional EIA | DLV11, BC05M<br>DLV11, BC05C |
| Alphanumeric Data Entry Terminal             | RT02-A | 30 Character Keyboard Remote Terminal | Yes, with limited Command Set | 32 characters                         | 110-300 (20 mA)<br>110-1200 (EIA) | No        | 20 mA loop or EIA          | DLV11, BC05M<br>DLV11, BC05C |
| Full Alphanumeric Data Entry Terminal        | RT02-B | Alphanumeric Terminal                 | Yes                           | 32 characters                         | 110-300 (20 mA)<br>110-1200 (EIA) | No        | 20 mA loop or EIA          | DLV11, BC05M<br>DLV11, BC05C |

### 1.3.2 LSI-11 Initial Power-On

Proceed as follows:

1. Ensure that there is no dc power applied to the backplane.
2. Remove all modules from the backplane.
3. It is recommended that a single switch be used to apply +5 V and +12 V to the backplane. Simultaneous application of +5 V and +12 V is recommended.
4. Turn power on.
5. At the backplane, check for the following voltages with respect to GND (pin C2 in any backplane slot):
  - Row 1, Slot A, Pin A2: + 5 V
  - Row 1, Slot A, Pin D2: +12 V
  - Row 1, Slot A, Pin V1: +5 V

#### CAUTION

Do not plug in modules with power applied to backplane.

6. Turn power off.
7. Ensure that the system is properly configured and installed as previously described in Section 1, Chapters 5 and 6. Install modules.
8. Turn on system power. Initialize the system by momentarily grounding the BDCOKH signal. Observe that the console device responds as described in Table 1-2.
9. Proceed with initial power-on checkout by entering and executing the program listed in Paragraph 6.3.3.

### 1.3.3 ASCII Character Console Printout Program

The following is a program that can be used to printout ASCII characters. The successful completion of this program can be used as a guide in determining the correct operation of the following:

KD11-F or -J Processor  
DLV11 Serial Line Unit  
LSI-11 data transfer and data control bus signals  
Power input connections for +12 V and +5 V

This program does not explicitly check the following bus signals.

|         |          |          |
|---------|----------|----------|
| BPOK H  | BHALT L  | BDMRL    |
| BEVNT L | BIRQ L   | BDMGI/OL |
| BDCOK H | BIAKI/OL | BSACK L  |

This program outputs all ASCII characters and may include control codes for specific devices.

Enter and execute the program via the console device as directed below:

**Table 1-2 Console Power-Up Printout (or Display)**

| Conditions                               | Mode 0 (Jumpers W5, W6 removed)   | Mode 1 (Jumper W6 removed, W5 installed)   | Mode 2 (Jumper W6 installed W5 removed)                                | Mode 3 (Jumper W5 and W6 installed)   |
|--|---|--|--|---------------------------------------|
| BHALT L (unasserted), Dynamic RAM Memory | Processor will execute program using contents of location 24 as the PC value.         | Terminal will print out a random 6-digit number, which is the contents of the program counter. | Processor will execute program at location 173000. (See Notes 2 and 3) | No printout at terminal. (See Note 1) |
| BHALT L (unasserted), Core Memory        | Processor will execute program in core using contents of location 24 as the PC value. | Terminal will print out a random 6-digit number, which is the contents of the program counter. | Processor will execute program at location 173000. (See Notes 2 and 3) | No printout at terminal. (See Note 1) |
| BHALT L (asserted), Dynamic RAM Memory   | Terminal will print out contents of memory location 024 (normally "000 000").         | Terminal will print out a random 6-digit number, which is the contents of the program counter. | Terminal will print out "173000."                                      | No printout at terminal. (See Note 1) |
| BHALT L (asserted), Core Memory          | Terminal will print out contents of memory location 024 (normally "000 000").         | Terminal will print out a random 6-digit number, which is the contents of the program counter. | Terminal will print out "173000."                                      | No printout at terminal. (See Note 1) |

## NOTES

1. If mode 3 is selected, and microaddress (3000—3777) is not implemented, the processor will trap to memory location 010 and start program execution using the contents of location 10 as the PC value and location 12 as the PSW value.
2. Whenever the PDP-11/03 is executing a program, the RUN indicator should be lit. If no program is provided or if a HALT instruction is executed, the RUN indicator will be extinguished.
3. Normal mode for use with the REV11-A or REV11-C options. Normal display is a \$ symbol, prompting the operator to input a command.

| Operator/System Response               | Symbolic Program (Do not enter) |
|--|---------------------------------|
| <u>@1000</u> /000000 105737<LF>        | LOOP: TSTB @ # 177564           |
| <u>001002</u> <u>000000</u> 177564<LF> |                                 |
| <u>001004</u> <u>000000</u> 100375<LF> | BPY LOOP                        |
| <u>001006</u> <u>000000</u> 110037<LF> | MOVB R0, @ # 177566             |
| <u>001010</u> <u>000000</u> 177566<LF> |                                 |
| <u>001012</u> <u>000000</u> 005200<LF> | INC R0                          |
| <u>001014</u> <u>000000</u> 000137<LF> | JMP @ # 1000                    |
| <u>001016</u> <u>000000</u> 001000<CR> |                                 |
| <u>@1000G</u>                          |                                 |

## NOTE

Underlined characters are those typed by the system. Characters not underlined are those typed by the operator.

1. Immediately after the "@" prompt character enter the starting address (1000), followed by a slash (/). The system responds by displaying the contents of location 1000 (000000). Enter the first instruction (105737) and close that location by pressing the LINE FEED key. The system responds by closing location 1000 and displaying the address and contents of the next location (1002) on the next line.
2. Enter the required contents for that location (177564), followed by the LINE FEED, and so on until the last location (001016) has been opened. After the required contents for that location have been entered, close it by pressing the RETURN key; the system responds by displaying the "@" prompt character on the next line.
3. Enable the Run mode by placing the ENABLE/HALT switch (PDP-11-03 panel or an equivalent LSI-11 switch) in the ENABLE position. Start the program by entering the starting address (1000) and the

Go (G) command. The system will continuously output ASCII characters to the console device for display until the program is halted. The processor can be halted at any time by pressing the BREAK key on the console device (if the FEH jumper is left installed, see Section I, chapter 5, paragraph 5, 6) or by placing the ENABLE/HALT switch (PDP-11/03 panel) in the HALT position. A sample console device (Teletype) printout is shown in Figure 1-1. When this program is executed using the LA36 as the console device, use 132 column paper in order to avoid typing off the end of the paper.

```

1000G !"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRS
!"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUvwxyz[\]^_`aBCDEF
!"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUvwxyz[\]^_`aBCDEF
!"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUvwxyz[\]^_`aBCDEF
!"#%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN0PQRSTUvwxyz[\]^_`aBCDEF

```

Figure 1-1 Sample Console Printout



## USING CONSOLE ODT COMMANDS

### 2.1 THE HALT MODE

Console ODT commands are executed by the LSI-11 processor only when the processor is in the Halt mode. When in this mode, the processor responds to commands and information entered via the console terminal, and all processor response is controlled by the processor's microcode.

#### NOTE

For console ODT communication, the DLV11 must be configured for console bus addresses 177560 through 177566. These addresses are included in the LSI-11 processor's microcode and cannot be changed. If no device responds to the above addresses, bus timeout errors will occur and the processor will go into an infinite microcode loop. The only way to get out of this loop is to initialize the system (momentarily assert the BDCOK H signal low, or cycle the power off and then on).

The Halt mode is entered by one of the following ways:

- Executing a HALT instruction
- Pressing the BREAK key on the console terminal (this feature can be disabled by removing the FEH jumper on the console device's DLV11 interface module)
- During power-up (power-up Mode 1 configured on the processor module)
- The BHALT L bus signal is asserted by any means
- A double Bus Error (Bus Error trap with SP (R6) pointing to non-existent memory)
- A Bus Error (timeout) during memory refresh
- A Bus Error (timeout) when the processor is attempting to input a vector from an interrupting device

Upon entering the Halt mode, the processor outputs the following ASCII non-printing and printing characters to the console terminal:

```
<CR><LF>
nnnnnn <CR><LF>
@
```

The nnnnnn is the location of the next instruction to be executed, and is always the contents of the PC (R7). The <CR> and <LF> are car-

riage return and line feed codes. The @ symbol is displayed as the prompt character for the operator; ODT will accept any of the commands described in this chapter at this point.

## 2.2 ODT COMMANDS

The following is a list of ODT commands and how they are used on the console terminal. Note that in the examples provided, characters output by the processor are shown underlined. Characters that are input by the operator are shown not underlined.

The commands described in this chapter are a subset of ODT-11. Only the commands necessary for implementing the required console functions are retained.

Note also that all commands and characters are echoed by the processor and that illegal commands will be echoed and followed by ? (ASCII 077) followed by CR (ASCII 015) followed by LF(ASCII 012) followed by @ (ASCII 100). If a valid command character is received when no location is open (e.g., when having just entered the halt state), the valid command character will be echoed and followed by a ?, CR, LF, @. Opening non-existent locations will have the same response. The console always prints six numeric characters; however, the user is not required to type leading zeros for either address or data. If a bus error (timeout) occurs during memory refresh while in the console ODT mode, a ?, CR, LF, @ will be typed.

### 1. "/" Slash (ASCII 057)

This command is used to open a memory location, general-purpose register, or the processor status word.

The / command is normally preceded by a location identifier. Before the contents is typed, the console will issue a space (ASCII 40) character.

**example:**

@ 001000/ 021525

**where:**

@ = KD11F prompt character (ASCII 100)  
001000 = octal location in address space to be opened  
/ = command to open and exhibit contents of location  
021525 = contents of octal location 1000

### NOTE

If / used without preceding location identifier, address of last opened location will be used. This feature can be used to verify the data just entered in a location.

### 2. "CR" carriage return (ASCII 015)

This command is used to close an open location. If contents of location are to be changed, CR should be preceded by the new value. If no change to location is necessary then CR will not alter contents.

**example:**

@ 001000/ 012525 CR LF

@ /012525

OR

**example:**

@ 001000/ 012525 15126421 CR LF

@ /126421

**where:**

CR = (ASCII 015) used to close location 1000 in both examples. Note that in second example contents of location 1000 was changed and that only the last 6 digits entered were actually placed in location 1000.

### 3. "LF" line feed (ASCII 021)

This command is also used to close an open location or GPR (general-purpose register). If entered after a location has been opened, it will close the open location or GPR and open location + 2 or GPR + 1. If the contents of the open location or GPR are to be modified, the new contents should precede the LF operator.

**example:**

@ 1000/ 012525 LF CR

001002/ 005252 CR LF

@

**where:**

LF = (ASCII 012) used to close location 1000 and open location 1002, if used on the PS, the LF will modify the PS if the data has been typed, and close it; then a CR, LF, @ is issued. If LF is used to advance beyond R7, the register name that is printed is meaningless but the contents printed is that of R0.

### 4. "↑" up arrow (ASCII 135)

The "↑" command is also the close an open location or GPR. If entered after a location or GPR has been opened, it will close the open location or GPR and open location 2, or GPR-1. If the contents of the open location or GPR are to be modified, the new contents should precede the "↑" operator.

**example:**

@ 1000/ 012525 ↑ CR LF

000776/ 010101 CR LF

@

**where:**

"↑" = (ASCII 135) used to close location 1000 and open location 776.

If used on the PS, the ↑ will modify the PS if the data has been typed and close it; then CR, LF, @ is issued. If ↑ is used to decrement below R0, the register name that is printed is meaningless but the contents is that of R7.

**5. "@" at sign (ASCII 100)**

The @ command is used once a location has been opened to open a location using the contents of the opened location as a pointer. Also the open location can be optionally modified similar to other commands and if done, the new contents will be used as the pointer.

**example:**

```
@ 1000/ 000200 @ CR LF  
000200/ 000137 CR LF  
@
```

**where:**

@ = (ASCII 100) used to close open location 1000 and open location 200.

Note that the @ command may be used with either GPRs or memory contents.

If used on the PS, the command will modify the PS if data has been typed and close it; however, the last GPR or memory location contents will be used as a pointer.

**6. "←" back arrow (ASCII 137)**

This command is used once a location has been opened to open the location that is the address of the contents of the open location plus the address of the open location plus 2. This is useful for relative instructions where it is desired to determine the effective address.

**example:**

```
@ 1000/ 00200 ← CR LF  
001202/ 002525 CR LF  
@
```

**where:**

"←" = (ASCII 137) used to close open location 1000 and open location 1202 (sum of contents of location 1000 which is 200, 1000 and 2). Note that this command cannot be used if a GPR or the PS is the open location and if attempted, the command will modify the GPR or PS if data has been typed, and close the GPR or PS; then a CR, LF, @ will be issued.

**7. \$ dollar sign (ASCII 044) or R (ASCII 122) internal register designator:**

Either command if followed by a register value 0—7 (ASCII 060—067) will allow that specific general-purpose register to be opened if followed by the / (ASCII 057) command.

**example:**

@ \$ n/ 012345 CR LF

@

**where:**

\$ = register designator. This could also be R.

n = octal register 0—7.

012345 = contents of GPR n.

Note that the GPRs once opened can be closed with either the CR, LF, “↑”, or @ commands. The “←” command will also close a GPR but will not perform the relative mode operation.

**8. “\$ s” (ASCII 123) processor status word**

By replacing “n” in the above example with the letter S (ASCII 123) the processor status word will be opened. Again either \$ or R (ASCII 122) is a legal command.

**example:**

@ \$ S/ 000200 CR LF

@

**where:**

\$ = GPR or processor status word designator

S = specifies processor status register; differentiates from GPRs.

000200 = eight bit contents of PSW; bit 7 = 1, all other bits = 0.

Note that the contents of the PSW can be changed using the CR command but bit 4 (the T bit) cannot be modified using any of the commands.

**9. “G” (ASCII 107)**

The “G” (GO) command is used to start execution of a program at the memory location typed immediately before the “G”.

**example:**

@ 100 G or 100;G

The LSI-11 PC(R7) will be loaded with 100, the PSW is cleared and execution will begin at that location. Immediately after echoing the “G” two null (000) characters are sent to the DLV11 to act as fill characters in case the bus BINIT L signal clears the DLV11. Before starting execution, a BUS INIT is issued for 10 μsec of idle time. Note that a semicolon character (ASCII 073) can be used to separate the address from the G and this is done for PDP-11 ODT compatibility. Since the console is a character by-character processor, as soon as the “G” is typed, the command is processed and a RUBOUT cannot be issued to cancel the command. If the B HALT L line is asserted, execution does not take place and only the BUS INIT sequence is done. The machine returns to console mode and prints the PC followed by CR, LF, @.

### NOTE

When program execution begins, the DLV11 serial line unit is still busy processing the two null characters. Thus, the program should not assume the done bit (bit 7) is set in the output status register at 177564.

#### 10. "P" (ASCII 120)

The "P" (Proceed) command is used to continue or resume execution at the location pointed to by the current contents of the PC(R7).

**example:**

@ P or ;P

If the B HALT L line is asserted, a single instruction will be executed, and the machine will return to console mode. It will print the contents of the PC followed by a CR, LF, @. In this fashion, it is possible to single instruction step through a user program. However, since the BHALT L line has higher priority than device interrupts, device interrupts will not be recognized in the single step mode.

The semicolon is accepted for PDP-11 ODT compatibility. If the semicolon character is received during any character sequence, the console ignores it.

#### 11. "M" (ASCII 115)

The "M" (Maintenance) command is used for maintenance purposes and prints the contents of an internal CPU register. This data reflects how the machine got to the console mode.

**example:**

@ M 00213 CR LF

@

The console prints six characters and then returns to command mode by printing CR,LF,@.

The last octal digit is the only number of significance and is encoded as follows. The value specifies how the machine got to the console mode.

#### Last Octal Digit Value

#### Function

0 or 4

Halt instruction or B Halt line

1 or 5

Bus Error occurred while getting device interrupt vector. This error probably indicates that the priority chain (BIAKI/O L signal) is broken in the system and that an open slot exists between modules. Modules must be inserted in a contiguous fashion according to the priority daisy chain.

| Last Octal Digit Value | Function  |
|------------------------|---|
| 2 or 6                 | Bus Error occurred while doing memory refresh   |
| 3                      | Double Bus Error occurred (stack contains non-existent address)                                 |
| 4                      | Reserved instruction trap occurred (non-existent Micro-PC address occurred on internal CPU bus) |
| 7                      | A combination of 1, 2, and 4, which implies that all three conditions occurred.                 |

In the above example, the last octal digit is a "3", which indicates a Double Bus Error occurred.

The codes listed above are valid only when the console mode is entered, and the code is immediately displayed. This information is lost when a "G" command is issued; the code reflects what happened in the program since the last "G" command was issued.

## 12. "RO" RUBOUT (ASCII 177)

While RUBOUT is not truly a command, the console does support this character. When typing in either address or data, the user can type RUBOUT to erase the previously typed character and the console will respond with a "\ " (Backslash—ASCII 134) for every typed RUBOUT.

example:

```
@ 000100/ 077777 123457 (RUBOUT) \ 6 CR LF
@ 000100/ 123456
```

In the above example, the user typed a "7" while entering new data and then typed RUBOUT. The console responded with a "\ " and then the user typed a "6" and CR. Then the user opens the same location and the new data reflects the RUBOUT. Note that if RUBOUT is issued repeatedly, only numerical characters are erased and it is not possible to terminate the present mode the console is in. If more than six RUBOUTS are consecutively typed, and then a valid location closing command is typed, the open location will be modified with all zeros.

The RUBOUT command cannot be used while entering a register number. R2 \ 4 / 012345 will not open register R4; however, the RUBOUT command will cause ODT to revert to memory mode and open location 4.

## 13. "L" (ASCII 114)

The "L" (Bootstrap Loader) command will cause the processor to self-size memory and then load a program that is in bootstrap loader format (e.g., the Absolute Loader program) from the specified device. The device is specified by typing the address of its input control and status register (RCSR) immediately before the "L". No bus initialize (BINIT L signal) is issued.

**example:**

**@ 177560L**

First, memory is sized, starting at 28K (157776), and the address is decremented by 2 until the highest read/write memory location is found. In small systems (e.g., 4K memory), a discernible pause of about 1 second will occur before tape motion is observed. Memory refresh continues in a normal manner during the sizing process. Then, the device RCSR address (177560 in the above example) is placed in the last location in memory (XXX776) for Absolute Loader compatibility. The program is then loaded by setting the "GO" bit (bit 0) in the device address and reading a byte of data from the device address plus 2 (177562); this address is the device's receiver data buffer. PDP-11 bootstrap loader format requires that the first data byte read from the tape is 351<sub>8</sub>. The Absolute Loader program tape, for example, has several inches of frames all punched with 351<sub>8</sub>. The first byte following the 351<sub>8</sub> bytes contains the low byte of the starting address minus 1. (For Absolute Loader, this byte is 075<sub>8</sub>.) All bytes which follow are data bytes. Loading continues until address XXX752 has been loaded. The data at that location is then treated as the low byte of a new load address. Loading continues until byte location XXX774 has been loaded. (Address detection is done via pointers contained in the LSI-11 processor's microcode.) The processor then loads a 1 into byte location XXX775 so that word location XXX774 contains a PDP-11 Branch instruction (000765). The processor does not modify the PSW nor issue a BINIT L signal; it starts program execution at location XXX774. The program being loaded must halt the processor, if that is desired. For example, when loading the Absolute Loader program, the processor will halt, and the console terminal will display XXX500 (the current PC contents), followed by CR, LF, @. When loading a program using the "L" command, the BHALT L signal line is ignored. If a timeout error occurs, such as would occur if a non-existent device was entered by the user preceding the "L" command, the console will terminate the load and print ?, CR, LF, @. Any device CSR address may be used that references an actual address configured on the reader device's bus interface controller module. For example, the console device address (RCSR=177560) can be configured on a DLV11 serial line unit which interfaces with an LT33 Teletype low-speed reader. If no address is entered for the reader device, address 0 will be used, and the system will likely "hang." The console ODT mode can be restored by momentarily asserting the BDCOK H signal low, or by cycling the power off and then on.

**14. "CONTROL SHIFT S" (ASCII 23)**

This command is used for manufacturing test purposes and is not a normal user command. It is briefly described here so that in case a user accidentally types this character, he will understand the machine response. If this character is typed, ODT expects two more characters, where the first character is treated as the high byte of an address, and the second character as the low byte of an address. It uses these two characters as a 16-bit binary address and start-

ing at that address, dumps five locations (or ten bytes) in binary format to the serial line.

It is recommended that if this mode is inadvertently entered, two characters such as a NULL (0) and @ (ASCII 100) be typed to specify an address in order to terminate this mode. Once completed, ODT will issue a CR, LF, @.



## USING REV11-A AND REV11-C COMMANDS

### 3.1 GENERAL

The REV11-A and REV11-C hardware options both contain the same programs, stored in read-only memory. The normal starting address for the programs is 173000. When started at this location, the program that is executed is a non-memory modifying processor test. If no errors are detected, the program outputs a dollar sign (\$) for display on the console device. This character is the prompt character for the operator to enter a command.

The starting address can be entered and program operation started either manually, using the console ODT Go command, or automatically during power-up. Automatic operation is accomplished by selecting power-up Mode 2 by appropriately configuring jumpers on the processor module. The normal power-up response for this mode results in the console device displaying the \$ prompt character instead of the @ console ODT prompt character.

Unsuccessful execution of the non-memory modifying processor test program results in the \$ prompt character not being displayed. Instead, the program hangs (branch to self) when a sequence of instruction do not execute properly, or the processor halts due to a double bus error; a halt normally results in the console terminal displaying the PC contents (the address of the Halt +2), followed by the console ODT prompt character @.

### 3.2 REV11-A AND REV11-C COMMAND SET

Once the \$ prompt character is displayed, the operator can enter one of the commands described in Table 3-1. Note that in the command examples, characters printed by the program are shown underlined; characters not underlined are entered by the operator. Command inputs to the program can either be upper or lower case characters. If an invalid command is entered following the \$ prompt character, the program responds by displaying ? after the invalid command and a new \$ prompt character on a new line. For example, program response to the invalid "XJ" command is shown below:

```
$ XJ ?
$
```

Table 3-1 REV11-A and REV11-C ROM Program Commands

| Command | Function  |
|---------|---|
| OD      | ODT (Halt). This allows the operator to examine and/or alter memory and register locations via the console device. Control can be returned to the |

**Table 3-1 REV11-A and REV11-C ROM Program Commands (Cont.)**

| Command | Function  |
|---------|---|
|         | <p>REV11 program by entering the ODT P (Proceed) command if the PC has not been altered, and the console device will display the \$ prompt character. If the PC has been altered, the operator can start program execution by entering the starting address 165006 and the G (Go) commands as follows:</p> <p style="padding-left: 40px;">@ 165006G</p> <p style="padding-left: 40px;">\$</p>   |
| XM<CR>  | <p>The processor responds by displaying the \$ prompt character on a new line and another REV11 command can be entered.</p> <p>Memory Diagnostic program. After successfully completing the diagnostic, the prompt character (\$) is displayed on the console device. Errors are indicated by the following displays on the console device:</p> <ol style="list-style-type: none"> <li data-bbox="335 671 888 865">1. <u>173732</u><br/> <p style="padding-left: 40px;">@</p> <p>This is an address test error. The expected (normal) data is in R3 and the invalid data is in the memory location pointed to by R2. If desired, continue diagnostic program execution by entering the ODT P command.</p> </li> <li data-bbox="335 874 888 1069">2. <u>173756</u><br/> <p style="padding-left: 40px;">@</p> <p>This is a data test error. The expected (normal) data is stored in R3 and the invalid data is in the memory location pointed to by R2. If desired, continue diagnostic program execution by entering the ODT P command.</p> </li> <li data-bbox="335 1078 888 1226">3. <u>000010</u><br/> <p style="padding-left: 40px;">@</p> <p>A timeout trap has occurred in testing memory locations outside of the first (lowest) 4K memory.</p> </li> <li data-bbox="335 1235 888 1433">4. <u>nnnnnn</u><br/> <p style="padding-left: 40px;">@</p> <p>A timeout trap has occurred in testing memory locations within the first 4K memory. The nnnnnn displayed is an indeterminate number.</p> <p>The actual memory test consists of an address test and a data test. The address test first writes all</p> </li> </ol> |

**Table 3-1 REV11-A and REV11-C ROM Program Commands (Cont.)**

| <b>Command</b>      | <b>Function</b>   |
|---------------------|---|
|                     | memory locations with addresses; it then reads and verifies the addresses. The data test consists of two parts. An "all 1's" word is first walked through all memory locations, which are initially 0. The second part consists of walking an "all 0's" word through all memory locations which are all 1's.  |
| <b>XC&lt;CR&gt;</b> | Processor Diagnostic program. This is a memory-modifying instruction test. Successful execution of the diagnostic program results in the prompt character (\$) being displayed on the console device. Errors are indicated by: <ol style="list-style-type: none"><li>1. The program halts when an instruction sequence is not correctly executed.</li><li>2. The program halts in the trap vector area for various traps.</li></ol> |

**NOTE**

When a halt occurs, the console ODT M command can be used to determine how the halt mode was invoked. When the system fails to successfully execute the above diagnostics, maintenance diagnostic programs should be used to thoroughly test processor (and memory) functions.

**AL<CR>** Absolute Loader program, normal (absolute address) loading operation. Entering AL<CR> specifies that a paper tape is to be loaded via the console device (CSR address = 177560). However, another device can be specified by entering the appropriate CSR address. For example, to load paper tapes in absolute loader format via a device whose CSR address is 177550, enter the following command:

\$ AL177550<CR>

The program responds by first executing the memory-modifying CPU instruction test and memory test (refer to the XC and XM commands). Successful test execution results in execution of the Absolute Loader program.

A successful program load is indicated when the console device displays:

165625

@

**Table 3-1 REV11-A and REV11-C ROM Program Commands (Cont.)**

| Command | Function  |
|---------|---|
|         | <p>the loaded program automatically starts execution, or<br/>                     Absolute loader errors are:</p> <ol style="list-style-type: none"> <li>1. Checksum error, with the program halting and producing the following display:<br/> <u>165534</u><br/> <u>@</u></li> <li>2. Program halts in the trap vector area for traps other than a timeout trap.</li> <li>3. Timeout trap occurs, causing the display of \$ on a new line on the console device.</li> </ol>  |
| AR<CR>  | <p>Absolute Loader program, relocated loading operation. When this command is entered, the memory-modifying CPU instruction test and memory test are automatically first executed (refer to the XC and XM commands), followed by the Absolute Loader program. Successful execution of the tests results in the program halting with the following console display:</p> <p><u>165412</u><br/> <u>@</u></p> <p>The operator must then enter the appropriate "software switch register" contents in R4. To select relocated loading, which uses an address (bias) contained in the software switch register, enter the following commands:</p> <p><u>@ R4/ <u>xxxxxx</u> nnnnnn &lt;CR&gt;</u><br/> <u>@ P</u></p> |
|         | <p>The value nnnnnn is a relocation value selected by the operator as directed in the <b>PDP-11 Paper Tape Software User's Handbook</b>. Observe that the least significant "n" value entered must be an odd number; this sets the software switch register (R4) bit 0 to a logical 1, selecting the relocated loading mode. Note that the program being loaded must be in Position Independent Code (PIC) format for relocated loading.</p>  |
|         | <p>When large programs are contained on more than one tape, the program halts at the end of the first tape. Install the second tape in the reader and enter a "1" in R4 using the ODT command shown below; resume loading by entering the P command.</p>  |

**Table 3-1 REV11-A and REV11-C ROM Program Commands (Cont.)**

| Command                    | Function  |
|----------------------------|---|
| @ R4/ <u>xxxxxx</u> 1 <CR> | The six octal digits (xxxxxx) are the present contents of R4. Entering a value of 1 selects relocated loading for the next program tape, starting at the address following the end of the previous load operation. The P command allows the absolute loader program execution to continue the loading process once the software switch register value has been entered. |
| @ P                        | A successful program load is indicated when the loaded program automatically starts execution, or the console device displays   |
|                            | <u>165626</u>   |
|                            | @   |
|                            | Absolute loader errors are as described for the AL command.   |
| DX <CR> or<br>DXn <CR>     | RXV11 floppy disk system bootstrap. Entering the DX <CR> command starts the memory-modifying CPU instruction test and memory test execution (see the XC and XM commands). Successful test execution results in execution of the bootstrap program for disk drive 0, the System disk. Otherwise, specify the drive number (n) as 0 (drive 0) or 1 (drive 1).             |
|                            | Floppy disk bootstrap errors are:   |
|                            | 1. The program halts and the console device displays:   |
|                            | <u>165316</u>   |
|                            | @   |
|                            | indicating that the device Done flag in the RXV11 interface was not set within the required time (approx. 1.3 seconds). The bootstrap can be restarted by entering the P command; the \$ is then displayed on the console device and the bootstrap command can be entered.  |
|                            | 2. The program halts and the console displays:  |
|                            | <u>165644</u>   |
|                            | @   |
|                            | indicating that a bootstrap error occurred. The RXV11's Error Register contents are stored in   |

**Table 3-1 REV11-A and REV11-C ROM Program Commands (Cont.)**

| Command | Function  |
|---------|---|
|         | <p>R2. By examining the contents of R2 and using the information contained in the <b>RXV11 User's Manual</b>, the exact nature of the error can be determined. Examine the contents of R2 (nnnnn) as follows:</p> <pre>@ R2/nnnnnn &lt;CR&gt;</pre> <pre>@ P</pre> <pre>\$</pre>  |
|         | <p>After examining R2, the bootstrap can be restarted by the P command; enter the desired bootstrap command immediately after the \$ prompt character.</p>  |
|         | <p>3. The program halts in the trap vector for traps; a timeout trap returns the program to the \$ prompt character. If a timeout trap occurs first check for proper system cable connections and device interface module installation. Then, attempt to successfully bootstrap the system by again entering the desired bootstrap command.</p> |

Bootstrap programs can be started from the Halt (console ODT) mode (refer to the OD command) without first executing the Diagnostic programs. Programs started in this manner include the Absolute Loaders (AL and AR) and the bootstrap for disk drive 0 (DX).

Start the DX bootstrap program by first loading R4 with the DX bootstrap starting address (165264). Start the REV11 program at 165242. This sequence of operations is shown below:

```
$ OD
@ R4/xxxxxx 165264 <CR>
@ 165242G
```

Start the Absolute Loader program (AL or AR) by loading the appropriate starting address in R4:

```
AL starting address = 165414
AR starting address = 165406
```

Load the highest available memory address in R5. For example, if the system contains 4K read/write memory, load R5 with 177776. Proceed loading by starting the REV11 program at 165242. The complete sequence for starting the AL program in a 4K system is shown below:

```
$ OD
@ R4/xxxxxx 165414 <CR>
@ R5/xxxxxx 177776 <CR>
@ 165242 G
```

## PAPER TAPE SYSTEM OPERATION

### 4.1 GENERAL

Paper tape systems include no mass storage devices and programs must be read into system memory prior to system operation. Programs are read from punched paper tapes using either an optional low-speed reader, such as the LT-33 Teletypewriter, or a high-speed reader (user-supplied). The normal sequence of operation is:

1. Load the Absolute Loader
2. Load program tapes
3. Execute the program

### 4.2 LOADING THE ABSOLUTE LOADER

The Absolute Loader program tape is loaded using the Bootstrap Loader, which is resident in the processor's microcode. The Bootstrap Loader is executed via the Halt mode and console ODT commands as directed below:

1. Enter the Halt mode
  - PDP-11/03**—Place the HALT/ENABLE switch in the HALT position. The console device prints the @ prompt character.
  - LSI-11**—Since LSI-11 systems are completely user-configured, the HALT mode can be entered via one or more of the following means:
    - a. Momentarily place the user-supplied HALT/ENABLE switch in the HALT position; return the switch to the ENABLE position.
    - b. Press the BREAK key on the console device (FEH jumper must be installed in the console SLU interface module).
    - c. Initialize the processor by momentarily negating BDCOK H (processor module jumper W5 must be installed and W6 removed).
2. Place the Absolute Loader tape (DEC-11-UABLB-A-PO) in the paper tape reader. Note that a long portion of the tape is punched with the octal value 351 (Channels 8, 7, 6, 4, and 1 are punched); position the tape so that any of those characters is located over the reader head.
3. Enable the tape reader as follows:
  - LT33 Teletypewriter**
    - a. Enable the low-speed reader by placing the LINE/OFF/LOCAL switch in the LINE position.
    - b. Place the START/STOP/FREE switch in the START position.
  - High-Speed Reader**—Turn the reader on and place "on line," as appropriate for the type of reader being used.
4. Enter the reader's CSR address by typing the value on the console device. For example, if the console device includes a paper tape reader (such as the LT33 low-speed reader), type:
  - @ 177560

### NOTE

In the above example, and all examples which follow, characters printed by ODT are shown underlined. Characters entered by the operator are shown not underlined.

The value 177560 is the console device's CSR.

5. Load the Absolute Loader tape by typing L immediately after the reader device's CSR. The tape will automatically be read followed by printing the Absolute Loader's starting address on a new line and the @ character on the following line. The complete command is shown below:

@ 177560L

037500

@

The starting address of the absolute loader depends upon the size of the system read/write memory (in any increments). Memory sizing is automatic and the Absolute Loader will be properly located for the particular system in which it is loaded. The above example is for a system containing 8K memory.

A listing of typical printouts for 4K memory increments is provided below:

| MEMORY SIZE | PRINTOUT      |
|-------------|---------------|
| 4K          | <u>017500</u> |
|             | @             |
| 8K          | <u>037500</u> |
|             | @             |
| 12K         | <u>057500</u> |
|             | @             |
| 16K         | <u>077500</u> |
|             | @             |
| 20K         | <u>117500</u> |
|             | @             |
| 24K         | <u>137500</u> |
|             | @             |
| 28K         | <u>157500</u> |
|             | @             |

If a proper printout of the absolute loader's starting address is not obtained, repeat steps 4 and 5. If no printout occurs, reenter the console ODT mode as directed in steps 1a through 1c, or cycle power off and then on.

## 4.3 LOADING PROGRAM TAPES

### 4.3.1 General

Program tapes are loaded into memory by the Absolute Loader program. The Absolute Loader can be used for normal loading and relocated loading operations. Normal loading causes the program being loaded to load at an absolute address punched in the program tape. Relocated loading allows loading certain program tapes into any specific area in memory, or to continue loading from where the loader left off on a previous load operation.

### 4.3.2 Normal Loading Procedure

1. Place the program tape in the reader with blank (leader) tape over the read head.
2. Enable the tape reader as described in Paragraph 4.2, Step 3.
3. Read the program tape:
  - a. If processor halted at Absolute Loader starting address (Paragraph 4.2, Step 5), type P (Proceed command).
  - b. Load additional programs by typing the starting address of the loader program (previously printed out as described in Paragraph 4.2, Step 5), followed by the G command, as follows:  
@037500G

### NOTE

1. The above example is applicable for an 8K memory system. Use an appropriate starting address as printed out on the console device after loading the Absolute Loader.
2. Leading 0's can be ignored. For example, 037500 can be entered as: @ 37500G.

After loading the program tape, the Absolute Loader program halts. The Halt PC +2 address is printed on a new line, followed by the @ character on the following line. The complete program loading sequence is shown below:

```
@ 177560L  
037500  
@ P  
037712 First Prog Tape Loaded
```

#### 4. Incorrect load:

If the Absolute Loader detects a checksum error while loading, it will halt at location XXX612, and the low byte of register R0 will contain the difference between the calculated checksum and the checksum that appears at the end of a data block. If this condition occurs, it usually indicates misalignment in the paper tape reader. When loading long tapes (i.e., 4K words or more), checksum errors are likely

to occur on some electromechanical readers. When checksum errors occur, repeat the loading procedure, starting at Step 1.

#### 4.3.3 Relocated Loading Procedure

Relocated loading can be specified by setting the software switch register (in the Absolute Loader program) to a particular value. This value is normally 0, and normal program loading is selected by default. Note that the software switch register's address is dependent upon the Absolute Loader starting address, as previously printed. Use the first three octal digits of the starting address as the most significant three digits and 516 as the three least significant digits. Hence, 037516 is the software switch register location for the Absolute Loader when loaded in an 8K system.

To continue loading from a previous load operation, type:

```
@ 037516/000000 1 <CR> <LF>  
@
```

#### NOTE

The above example is for an 8K memory system.

This type of relocated loading is particularly useful for large programs which are contained on more than one tape. To select relocated loading which uses an address (bias) contained in the software switch register, type:

```
@ 037516/000000 nnnnnn <CR> <LF>  
@
```

#### NOTES

1. The above example is for an 8K memory system.
2. Select a relocation value nnnnnn as specified in the **PDP-11 Paper Tape Software User's Handbook**. Observe that the least significant "n" value entered must be an odd number; this sets the software switch register bit 0 to a logical 1, selecting the relocated loading mode.
3. The program being loaded must be in a Position Independent Code (PIC) format to allow relocation.

#### 4.3.4 Self Starting Programs

Some programs are self starting (described in the **Paper Tape Software Programming Handbook**), and can automatically proceed into execution immediately after loading. Load the program tape as previously described. Instead of the Absolute Loader program halting after loading the program tape, program control transfers to the loaded program's starting address, and the processor proceeds with normal program execution.

#### 4.4 PROGRAM STARTING AND EXECUTION

Once a program has been correctly loaded, program execution can be

started by placing the HALT/ENABLE switch (PDP-11/03 panel, or equivalent user-supplied LSI-11 switch) in the ENABLE position. Start normal program execution as follows:

```
@ 200G
```

The 200 in the above example is a typical starting address. Each program listing specifies the correct starting address. G is the Go command, and program execution will immediately commence, starting at the specified location.

Single instruction execution, when desired, is obtained by operating the processor in the Halt mode. Place the HALT/ENABLE switch in the HALT position. Enter the starting address (or a desired address for the first instruction to be executed) as described for normal program execution. The G command causes the processor to initialize the system and then Halt with the PC (R7) pointing to the first instruction. This address is printed on the console device as shown in the following example:

```
@ 200G <CR LF>  
000200 <CR LF>  
@
```

Successive instruction executions will occur each time the Proceed (P) command is entered via the console device.

An example of single instruction execution using the P command is shown below:

```
@ 200G  
000200  
@ P  
000202  
@ P  
000204  
@ P  
000206  
@
```

Note that after executing each instruction, the processor halts and prints the address of the next instruction. Thus, Branch, Jump, and JSR instructions will alter the PC as required in normal program execution, allowing the operator time to observe program operation.

#### NOTE

Avoid single instruction execution of programs using interrupts. Those interrupts cannot be serviced by the processor when in the Halt mode because the Halt mode service has higher priority than device interrupts.

#### 4.5 PAPER TAPE SOFTWARE

Three paper tape software options are presently available for use on LSI-11 and PDP-11/03 systems:

##### 1. QJV10-CB

This option contains the paper tape software tools required for user-generation of programs for specific applications. Documentation included in the option provides complete instructions for using the various paper tape programs. Programs included in QJV10-CB are listed below:

- ED-11 Text Editor
- PAL11S Assembler
- LINK11 Linker
- DUMPB Memory Dump utility program
- ODT-11 On-Line Debugging Technique program for debugging assembled programs
- IOX Input/Output Executive
- Absolute Loader

##### 2. QJV11-CB

This program aids in user-generation of PROM listings and tapes. A complete description of this program is included in Section I, Chapter 7.

##### 3. ZJV01-RB LSI-11

LSI-11, PDP-11/03 diagnostic software. Paper tape diagnostic software and documentation contained in this software option are described in Section I, Chapter 9, Paragraph 9.3.

## RXV11 FLOPPY DISK-BASED SYSTEM OPERATION

### 5.1 GENERAL

A typical RXV11-based system includes the RXV11 Floppy Disk System, a console terminal and its serial line unit interface, the LSI-11 processor, and an 8K (minimum) read/write memory. PDP-11/03 systems include the LSI-11 processor, console terminal serial line unit interface, and 4K read/write memory; an optional 4K read/write memory (MSV11-B), console terminal, and the RXV11 Floppy Disk System are required.

RXV11 hardware includes the RX01 single or dual floppy disk drive, M7946 interface module, and BC05L-15 interface cable. Models are available for 115 V, 60 Hz and 230 V, 50 Hz operation.

The RX01 contains no operator controls or indicators other than the load door(s) on the front panel. The left drive dual drive models is named DX0 and the right drive is DX1. Load the system diskette in the left (or only) drive: this drive (DX0) is called the System device in most systems.

### 5.2 BOOTSTRAPPING THE RXV11

#### 5.2.1 General

The RXV11 bootstrap loader program loads the system monitor from disk into system memory. No operation can occur until the monitor is contained in system memory. Bootstrapping ("booting") the system can be accomplished via a hardware-implemented bootstrap in the REV11-A or REV11-C option, or it can be entered and executed via the console device.

#### 5.2.2. Bootstrapping The System Using The REV11-A or REV11-C

The REV11-A and REV11-C implement the RXV11 bootstrap (and other programs) in four pre-programmed ROM chips. When system power is applied, and LSI-11 processor Mode 2 power-up sequence is configured on the processor module, the system responds with a dollar sign (\$) on a new line. If Mode 2 is not configured, the REV11 program can be started from the console ODT mode by entering 173000G <CR>. The operator then responds by typing the device to be bootstrapped. DX (or DX0) is disk drive 0; DX1 is the second drive in dual drive RXV11 systems. A normal sequence of operations from power up through booting DX0 is shown below:

```
$DX <CR>
RT-11SJ V02C-XX
```

## NOTE

During the booting process the floppy disk drive will make audible clicking sounds. This is normal, indicating that the heads are moving over the diskette.

After executing the DX0 bootstrap, the system responds by displaying the monitor in use (RT-11SJ in the above examples) and the particular version in use (V02C-XX); the version is changed software changes are implemented. Finally, a dot is displayed on the next line, indicating that the RT-11 Keyboard Monitor is ready to accept a command. The system is correctly booted and RT-11 programs can be executed as desired. Other software systems can be booted on disk drive 0 or disk drive 1 in the same manner if they contain a software bootstrap program on track 1, sector 1.

If incorrect loading occurs (an unexpected halt during the booting process), refer to Paragraph 5.2.4.

### 5.2.3 Booting The System Via the Console Device

When the REV11-A or REV11-C option is not included in the system, the operator must enter a bootstrap program via the console device. Place the processor in the Halt mode and proceed as shown below; observe that underlined characters are printed by the processor and non-underlined characters are entered by the operator:

```
@1000/000000 12702 <LF>          @1000/000000 5000 <LF>
001002/000000 1002n7 <LF> *      001002/000000 12701 <LF>
001004/000000 12701 <LF>         001004/000000 177170 <LF>
001006/000000 177170 <LF>       001006/000000 105711 <LF>
001010/000000 130211 <LF>       001010/000000 1776 <LF>
001012/000000 1776 <LF>        001012/000000 12711 <LF>
001014/000000 112703 <LF>      001014/000000 3 <LF>
001016/000000 7 <LF>           001016/000000 5711 <LF>
001020/000000 10100 <LF>       001020/000000 1776 <LF>
001022/000000 10220 <LF>       001022/000000 100405 <LF>
001024/000000 402 <LF>        001024/000000 105711 <LF>
001026/000000 12710 <LF>     001026/000000 100004 <LF>
001030/000000 1 <LF>          001030/000000 116120 <LF>
001032/000000 6203 <LF>      001032/000000 2 <LF>
001034/000000 103402 <LF>     001034/000000 770 <LF>
001036/000000 112711 <LF>     001036/000000 0 <LF>
001040/000000 111023 (LF)     001040/000000 5007 <CR>
001042/000000 30211 <LF>
001044/000000 1776 <LF>
001046/000000 100756 <LF>
001050/000000 103766 <LF>
001052/000000 105711 <LF>
001054/000000 100771 <LF>
001056/000000 5000 <LF>
001060/000000 22710 <LF>
001062/000000 240 <LF>
001064/000000 1347 <LF>
001066/000000 122702 <LF>
001070/000000 247 <LF>
001072/000000 5500 <LF>
001074/000000 5007 <CR>
```

\*n = 4 for Unit 0  
n = 6 for Unit 1  
<LF> = Line Feed  
<CR> = Carriage Return  
Starting address = 1000

The bootstrap program can be started at location 1000. Enable the Run mode by placing the HALT/ENABLE switch (on the PDP-11/03 panel, or an equivalent LSI-11 switch) in the ENABLE position. Start the program using the Go command as follows:

@ 1000G

After a few seconds the monitor will be loaded in system memory. The monitor will identify itself on the console device by displaying a message, such as: RT-11SJ V02X-XX; this printout is followed by the Keyboard Monitor prompt character (.) displayed on the next line, for the RT-11 example shown above.

If incorrect loading occurs (an unexpected halt during the booting process), refer to Paragraph 5.2.4.

#### **5.2.4 Incorrect Loading**

Incorrect loading (booting) results in unexpected halts or a hung program. If a halt occurs, type "M" (maintenance command) to determine the cause of the halt. Six digits will be displayed, but only the last digit is used.

If the last digit is a 1 or 5, it indicates that the priority chain is broken. This is probably due to an open option slot between modules. Recheck all modules to ensure that they are inserted in contiguous locations; unused option locations must not occur between peripheral devices requiring interrupt (or DMA) service and the processor module.

If the last digit is a 3 or 7, the stack pointer (R6) has become invalid. This may be the result of a faulty memory module.

When booting the system, if the system does not respond with a display within 5 seconds, halt the processor and examine the part of the program that is presently being executed (at the time of the halt). The program may be in a loop. The actual cause of the problem may be discerned by decoding that portion of the program. A simple hardware problem may exist, such as the door of the floppy disk drive may not be closed properly.



# section**3**

# processor



**CHAPTER 1 INTRODUCTION**

**CHAPTER 2 ADDRESSING MODES**

**CHAPTER 3 INSTRUCTION SET**

**CHAPTER 4 EXTENDED ARITHMETIC OPTION**

**CHAPTER 5 PROGRAMMING TECHNIQUES**

# CHAPTER 1

## INTRODUCTION

In this section the processor hardware is examined from the programmer's point of view. The major portion of the section includes a description of the PDP-11 addressing modes and instruction set. In addition, the KEV11 EIS/FIS option instructions are described. Programming techniques that illustrate the use of the instruction set, addressing modes, and processor hardware features are included at the end of this section. The remainder of this chapter contains general information on processor organization, bus cycles, and addressing.

### 1.1 PROCESSOR HARDWARE

#### 1.1.1 General Registers

The LSI-11 central processor module contains eight 16-bit general-purpose registers that can perform a variety of functions. These registers can serve as accumulators, index registers, autoincrement registers, auto-decrement registers, or as stack pointers for temporary storage of data. Arithmetic operations can be from one general register to another, from one memory location or device register to another, or between memory locations or a device register and a general register. The following illustration identifies the eight 16-bit general registers R0 through R7.

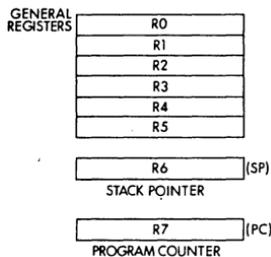


Figure 1-1 General Register Identification

Registers R6 and R7 in the LSI-11 are dedicated. R6 serves as the Stack Pointer (SP) and contains the location (address) of the last entry in the stack. Register R7 serves as the processor's Program Counter (PC) and contains the address of the next instruction to be executed. It is normally used for addressing purposes only and not as an accumulator. Register operations are internal to the processor and do not require bus cycles (except for instruction fetch); all memory and peripheral device data transfers do require bus cycles and longer execution time. Thus, general registers used for processor operations result in faster execution times. The bus cycles required for memory and device references are described below.

### Bus Cycles

The bus cycles (with respect to the processor) are:

|        |   |                                 |
|--------|---|---------------------------------|
| DATI   | Data word transfer input                                  | Equivalent to Read operation    |
| DATIO  | Data word transfer input followed by word transfer output | Equivalent to Read/Modify Write |
| DATIOB | Data word transfer input followed by byte transfer output | Equivalent to Read/Modify Write |
| DATO   | Data word transfer output                                 | Equivalent to write operation   |
| DATOB  | Data byte transfer output                                 | Equivalent to write operation   |

Every processor instruction requires one or more bus cycles. The first operation required is a DATI, which fetches an instruction from the location addressed by the Program Counter (R7). If no further operands are referenced in memory or in an I/O device, no additional bus cycles are required for instruction execution. If memory or a device is referenced, however, one or more additional bus cycles are required.

Note the distinction between interrupts and DMA operations: Interrupts, which may change the state of the processor, can occur only between processor instructions; DMA operations can occur between individual bus cycles since these operations do not change the state of the processor.

### Addressing Memory and Peripherals

The maximum direct address space of the LSI-11 is 32K 16-bit words. LSI-11's memory locations and peripheral device registers are addressed in precisely the same manner. The upper 4096 addresses (28K-32K) are reserved by convention for peripheral device addressing. However, the user does not need to dedicate the entire 4K space to I/O; he can implement only what he needs.

An LSI-11 word is divided into a high byte and a low byte as shown below.

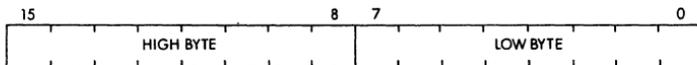


Figure 1-2 High and Low Byte

Word addresses are always even-numbered. Byte addresses can be either even- or odd-numbered. Low bytes are stored at even-numbered memory locations and high bytes at odd-numbered memory locations. Thus, it is convenient to view the memory as:



### Condition Codes

The condition codes contain information on the result of the last CPU operation. The bits are set as follows: (The bits are set after execution of all arithmetic or logical single operand or double operand instructions.)

Z = 1, if the result were zero

N = 1, if the result were negative

C = 1, if the operation resulted in a carry from the MSB (most significant bit) or a 1 were shifted from MSB or LSB (least significant bit)

V = 1, if the operation resulted in an arithmetic overflow

### Trap (T Bit)

The program can only set or clear the trap bit (T) by popping a new PSW off the stack. When set, a processor trap will occur through location 14 at completion of the current instruction execution, and a new processor status word will be loaded from location 16. This T bit is especially useful in debugging programs since it allows programs to be single instruction stepped.

### 1.1.3 Instruction Set

Implementing the PDP-11 instruction repertoire in the LSI chip set permits the user to take advantage of Digital Equipment Corporation's years of experience with the PDP-11 family—more than 17,000 units installed, with all associated application notes, software, documentation, training, reliability, customer references, and the DECUS library of application programs.

The instruction complement uses the flexibility of the general-purpose registers to provide more than 400 powerful hard-wired instructions—the most comprehensive and powerful instruction repertoire of any computer in the 16-bit class. Unlike conventional 16-bit computers, which usually have three classes of instructions (memory reference instructions, operate or accumulator control instructions, and I/O instructions), all data manipulation operations in the LSI-11 are accomplished with one set of instructions. Since peripheral device registers can be manipulated as flexibly as memory by the central processor, instructions that are used to manipulate data in memory can be used equally well for data in peripheral device registers. For example, data in an external device register can be tested or modified directly by the CPU without bringing it into memory or disturbing the general registers. One can add or compare data logically or arithmetically in a device register.

The basic order code of the LSI-11 uses both single and double operand address instructions for words or bytes. The LSI-11 therefore performs very efficiently in one step such operations as adding or subtracting two operands or moving an operand from one location to another.

### LSI-11 Approach

ADD A, B

Add contents of location A to location B;  
store results at location B

### Conventional Approach

|       |   |
|-------|---|
| LDA A | Load contents of memory location A into accumulator |
| ADD B | Add contents of memory location B to accumulator    |
| STA B | Store result at location B                          |

### Addressing

Much of the power of the LSI-11 is derived from its wide range of addressing capabilities. LSI-11 addressing modes include sequential forward or backward addressing, address indexing, indirect addressing, 16-bit word addressing, 8-bit byte addressing, and stack addressing. Variable-length instruction formatting allows a minimum number of words to be used for each addressing mode. The result is efficient use of program storage space.

## 1.2 LSI-11 MEMORY ORGANIZATION

The LSI-11 processor organization and addressing, register, memory, and device addresses are shown.

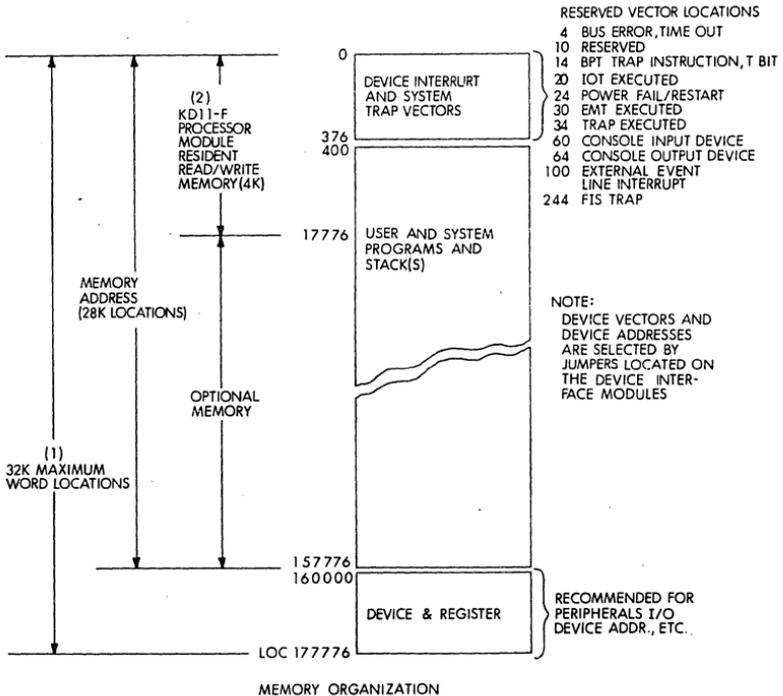


Figure 1-5 Memory Organization

#### **NOTE**

1. There is 32K of users memory space available; however, 0-28K is recommended for memory address locations, and 28K-32K for peripherals I/O device addresses, etc.
2. KD11-F resident memory can be assigned to the first 4K memory bank, as shown, or the second memory bank (address range 20000-37776). Core memory supplied with the KD11-J processor can reside in any 4K memory bank.

# ADDRESSING MODES

Data stored in memory must be accessed and manipulated. Data handling is specified by an LSI-11 instruction (MOV, ADD, etc.), which usually indicates:

- The function (operation code).
- A general-purpose register is to be used when locating the source operand and/or a general-purpose register to be used when locating the destination operand.
- An addressing mode (to specify how the selected register(s) is/are to be used).

A large portion of the data handled by a computer is usually structured (in character strings, arrays, lists, etc.). LSI-11's addressing modes provide for efficient and flexible handling of structured data.

The general registers may be used with an instruction in any of the following ways:

- As accumulators. The data to be manipulated resides within the register.
- As pointers. The contents of the register is the address of the operand, rather than the operand itself.
- As pointers which automatically step through memory locations. Automatically stepping forward through consecutive locations is known as autoincrement addressing; automatically stepping backwards is known as autodecrement addressing. These modes are particularly useful for processing tabular or array data.
- As index registers. In this instance, the contents of the register and the word following the instruction are summed to produce the address of the operand. This allows easy access to variable entries in a list.

An important LSI-11 feature, which should be considered in conjunction with the addressing modes, is the register arrangement:

- Six general-purpose registers (R0 – R5)
- A hardware Stack Pointer (SP) register (R6)
- A Program Counter (PC) register (R7)

Registers R0 through R5 are not dedicated to any specific function; their use is determined by the instruction that is decoded:

- They can be used for operand storage. For example, contents of two registers can be added and stored in another register.
- They can contain the address of an operand or serve as pointers to the address of an operand.
- They can be used for the autoincrement or autodecrement features.
- They can be used as index registers for convenient data and program access.

The LSI-11 also has instruction addressing mode combinations that facilitate temporary data storage structures. This can be used for convenient handling of data that must be accessed frequently. This is known as stack manipulation. The register used to keep track of stack manipulation is known as the stack pointer (SP). Any register can be used as a "stack pointer" under program control; however, certain instructions associated with subroutine linkage and interrupt service automatically use Register R6 as a "hardware stack pointer." For this reason, R6 is frequently referred to as the "SP:"

- The stack pointer (SP) keeps track of the latest entry on the stack.
- The stack pointer moves down as items are added to the stack and moves up as items are removed. Therefore, it always points to the top of the stack.
- The hardware stack is used during trap or interrupt handling to store information allowing the processor to return to the main program.

Register R7 is used by the processor as its program counter (PC). It is recommended that R7 not be used as a stack pointer or accumulator. Whenever an instruction is fetched from memory, the program counter is automatically incremented by two to point to the next instruction word.

The next section is divided into seven major categories:

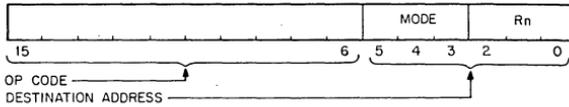
- **Single Operand Addressing**—One part of the instruction word specifies a register; the second part provides information for locating the operand.
- **Double Operand Addressing**—Part of the instruction word specifies the registers; the remaining parts provide information for locating two operands.
- **Direct Addressing**—The operand is the content of the selected register.
- **Deferred (Indirect) Addressing**—The contents of the selected register is the address of the operand.
- **Use of the PC as a General Register**—The PC is unique from other general-purpose registers in one important respect. Whenever the processor retrieves an instruction, it automatically advances the PC by 2. By combining this automatic advancement of the PC with four of the basic addressing modes, we produce the four special PC modes—immediate, absolute, relative, and relative deferred.
- **Use of Stack Pointer as General Register**—Can be used for stack operations.
- **Summary of Addressing Modes**

#### NOTE

Instruction mnemonics and address mode symbols are sufficient for writing assembly language programs. The programmer need not be concerned about conversion to binary digits; this is accomplished automatically by the assembler program.

## 2.1 SINGLE OPERAND ADDRESSING

The instruction format for all single operand instructions (such as clear, increment, test) is:



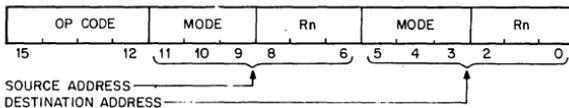
Bits 15 through 6 specify the operation code that defines the type of instruction to be executed.

Bits 5 through 0 form a six-bit field called the destination address field. This consists of two subfields:

- Bits 0 through 2 specify which of the eight general-purpose registers is to be referenced by this instruction word.
- Bits 3 through 5 specify how the selected register will be used (address mode). Bit 3 is set to indicate deferred (indirect) addressing.

## 2.2 DOUBLE OPERAND ADDRESSING

Operations which imply two operands (such as add, subtract, move, and compare) are handled by instructions that specify two addresses. The first operand is called the source operand, the second the destination operand. Bit assignments in the source and destination address fields may specify different modes and different registers. The instruction format for the double operand instruction is:



The source address field is used to select the source operand, the first operand. The destination is used similarly, and locates the second operand and the result. For example, the instruction ADD A, B adds the contents (source operand) of location A to the contents (destination operand) of location B. After execution B will contain the result of the addition and the contents of A will be unchanged.

Examples in this section and chapter use the following sample LSI-11 instructions. A complete listing of the LSI-11 instructions is located in the appendix.

| <b>Mnemonic</b> | <b>Description</b>   | <b>Octal Code</b> |
|-----------------|--|-------------------|
| CLR             | Clear (zero the specified destination)   | 0050DD            |
| CLRB            | Clear byte (zero the byte in the specified destination)  | 1050DD            |
| INC             | Increment (add 1 to contents of destination)   | 0052DD            |
| INCB            | Increment byte (add 1 to the contents of destination byte)   | 1052DD            |
| COM             | Complement (replace the contents of the destination by their logical complement; each 0 bit is set and each 1 bit is cleared)            | 0051DD            |
| COMB            | Complement byte (replace the contents of the destination byte by their logical complement; each 0 bit is set and each 1 bit is cleared). | 1051DD            |
| ADD             | Add (add source operand to destination operand and store the result at destination address)  | 06SSDD            |

DD = destination field (6 bits)

SS = source field (6 bits)

( ) = contents of

### 2.3 DIRECT ADDRESSING

The following table summarizes the four basic modes used with direct addressing.

#### DIRECT MODES

| Mode | Name          | Assembler Syntax | Function   |
|------|---------------|------------------|--|
| 0    | Register      | Rn               | Register contains operand  |
|      |               |                  |  |
| 2    | Autoincrement | (Rn) +           | Register is used as a pointer to sequential data then incremented                        |
|      |               |                  |  |
| 4    | Autodecrement | -(Rn)            | Register is decremented and then used as a pointer.                                      |
|      |               |                  |  |
| 6    | Index         | X(Rn)            | Value X is added to (Rn) to produce address of operand. Neither X nor (Rn) are modified. |
|      |               |                  |  |

#### 2.3.1 Register Mode

OPR Rn

With register mode any of the general registers may be used as simple accumulators and the operand is contained in the selected register. Since they are hardware registers, within the processor, the general registers operate at high-speeds and provide speed advantages when used for operating on frequently-accessed variables. The assembler interprets and assembles instructions of the form OPR Rn as register mode operations. Rn represents a general register name or number and OPR is used to represent a general instruction mnemonic. Assembler syntax requires that a general register be defined as follows:

R0 = %0 (% sign indicates register definition)

R1 = %1

R2 = %2, etc.

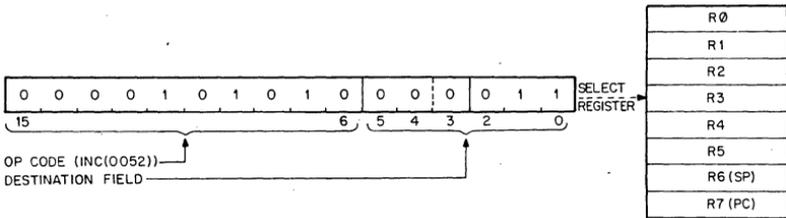
Registers are typically referred to by name as R0, R1, R2, R3, R4, R5, R6 and R7. However R6 and R7 are also referred to as SP and PC, respectively.

**Register Mode Examples**  
(all numbers in octal)

**Symbolic                      Octal Code    Instruction Name**

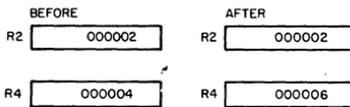
1.            **INC R3**                      005203            Increment

Operation:                      Add one to the contents of general register 3.



2.            **ADD R2,R4**                      060204            Add

Operation:                      Add the contents of R2 to the contents of R4.



3.            **COMB R4**                      105104            Complement Byte

Operation:                      One's complement bits 0-7 (byte) in R4. (When general registers are used, byte instructions only operate on bits 0-7; i.e., byte 0 of the register.)



### 2.3.2 Autoincrement Mode

OPR (Rn)+

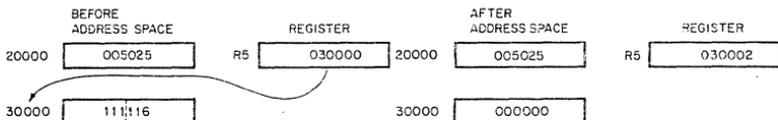
This mode provides for automatic stepping of a pointer through sequential elements of a table of operands. It assumes the contents of the selected general register to be the address of the operand. Contents of registers are stepped (by one for bytes, by two for words, always by two for R6 and R7) to address the next sequential location. The auto-increment mode is especially useful for array processing and stack processing. It will access an element of a table and then step the pointer to address the next operand in the table. Although most useful for table handling, this mode is completely general and may be used for a variety of purposes.

#### Autoincrement Mode Examples

|    | Symbolic   | Octal Code | Instruction Name |
|----|------------|------------|------------------|
| 1. | CLR (R5) + | 005025     | Clear            |

Operation:

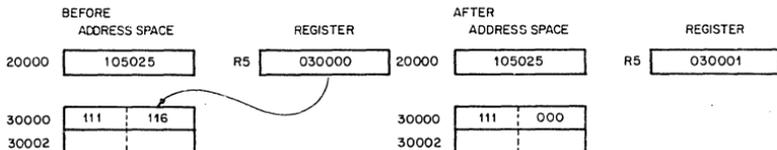
Use contents of R5 as the address of the operand. Clear selected operand and then increment the contents of R5 by two.



|    |             |        |            |
|----|-------------|--------|------------|
| 2. | CLRB (R5) + | 105025 | Clear Byte |
|----|-------------|--------|------------|

Operation:

Use contents of R5 as the address of the operand. Clear selected byte operand and then increment the contents of R5 by one.



3.        **ADD (R2) +,R4    062204    Add**

**Operation:**                    The contents of R2 are used as the address of the operand which is added to the contents of R4. R2 is then incremented by two.



**2.3.3 Autodecrement Mode (Mode 4)**

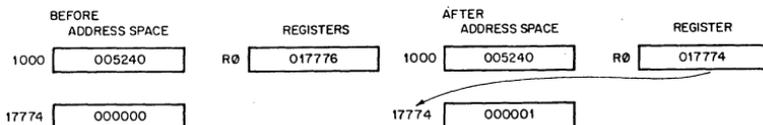
**OPR-(Rn)**

This mode is useful for processing data in a list in reverse direction. The contents of the selected general register are decremented (by two for word instructions, by one for byte instructions) and then used as the address of the operand. The choice of postincrement, predecrement features for the LSI-11 were not arbitrary decisions, but were intended to facilitate hardware/software stack operations.

**Autodecrement Mode Examples**

- |    | Symbolic        | Octal Code | Instruction Name |
|----|-----------------|------------|------------------|
| 1. | <b>INC-(R0)</b> | 005240     | Increment        |

**Operation:**                    The contents of R0 are decremented by two and used as the address of the operand. The operand is incremented by one.



- |    |                  |        |                |
|----|------------------|--------|----------------|
| 2. | <b>INCB-(R0)</b> | 105240 | Increment Byte |
|----|------------------|--------|----------------|

**Operation:**                    The contents of R0 are decremented by one then used as the address of the operand. The operand byte is increased by one.





## 2.4 DEFERRED (INDIRECT) ADDRESSING

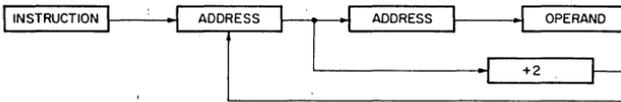
The four basic modes may also be used with deferred addressing. Whereas in the register mode the operand is the contents of the selected register, in the register deferred mode the contents of the selected register is the address of the operand.

In the three other deferred modes, the contents of the register select the address of the operand rather than the operand itself. These modes are therefore used when a table consists of addresses rather than operands. Assembler syntax for indicating deferred addressing is "@" (or "(" ) when this is not ambiguous). The following table summarizes the deferred versions of the basic modes:

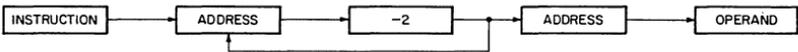
| Mode | Name              | Assembler Syntax | Function                                      |
|------|-------------------|------------------|---|
| 1    | Register Deferred | @Rn or (Rn)      | Register contains the address of the operand. |



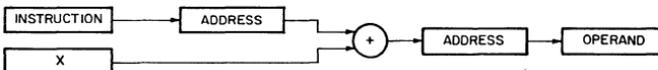
|   |                        |         |  |
|---|------------------------|---------|--|
| 3 | Autoincrement Deferred | @(Rn) + | Register is first used as a pointer to a word containing the address of the operand, then incremented (always by 2; even for byte instructions). |
|---|------------------------|---------|--|



|   |                        |        |   |
|---|------------------------|--------|---|
| 5 | Autodecrement Deferred | @-(Rn) | Register is decremented (always by two; even for byte instructions) and then used as a pointer to a word containing the address of the operand. |
|---|------------------------|--------|---|



|   |                |        |  |
|---|----------------|--------|--|
| 7 | Index Deferred | @X(Rn) | Value X (stored in a word following the instruction) and (Rn) are added and the sum is used as a pointer to a word containing the address of the operand. Neither X nor (Rn) are modified. |
|---|----------------|--------|--|

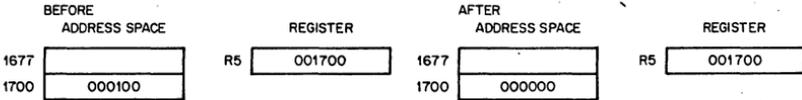


The following examples illustrate the deferred modes.

### Register Deferred Mode Example

| Symbolic | Octal Code | Instruction Name |
|----------|------------|------------------|
| CLR @R5  | 005015     | Clear            |

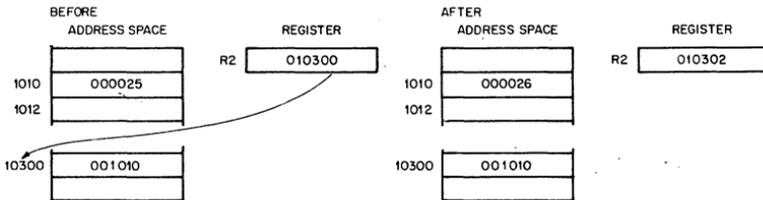
Operation: The contents of location specified in R5 are cleared.



### Autoincrement Deferred Mode Example (Mode 3)

| Symbolic   | Octal Code | Instruction Name |
|------------|------------|------------------|
| INC@(R2) + | 005232     | Increment        |

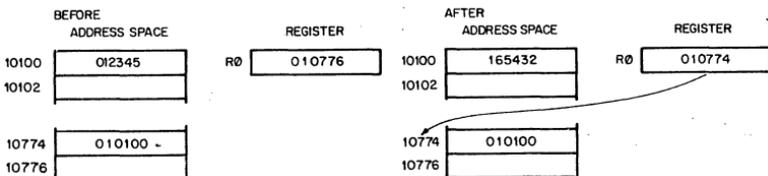
Operation: The contents of R2 are used as the address of the address of the operand. Operand is increased by one. Contents of R2 are incremented by 2.



### Autodecrement Deferred Mode Example (Mode 5)

| Symbolic  | Octal Code | Complement |
|-----------|------------|------------|
| COM @(R0) | 005150     |            |

Operation: The contents of R0 are decremented by two and then used as the address of the address of the operand. Operand is one's complemented. (i.e., logically complemented).

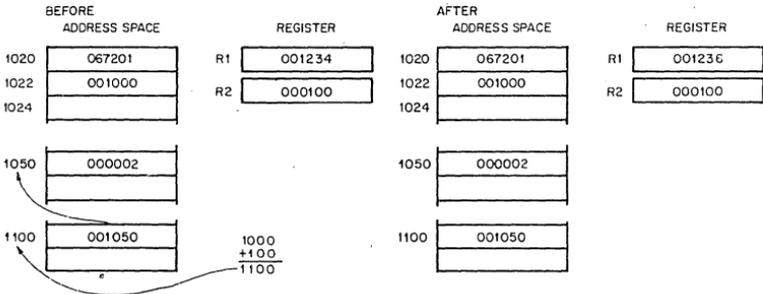


## Index Deferred Mode Example (Mode 7)

|                   |                  |                  |
|-------------------|------------------|------------------|
| Symbolic          | Octal Code       | Instruction Name |
| ADD @ 1000(R2),R1 | 067201<br>001000 | Add              |

Operation:

1000 and contents of R2 are summed to produce the address of the address of the source operand the contents of which are added to contents of R1; the result is stored in R1.



## 2.5 USE OF THE PC AS A GENERAL REGISTER

Although Register 7 is a general purpose register, it doubles in function as the Program Counter for the LSI-11. Whenever the processor uses the program counter to acquire a word from memory, the program counter is automatically incremented by two to contain the address of the next word of the instruction being executed or the address of the next instruction to be executed. (When the program uses the PC to locate byte data, the PC is still incremented by two.)

The PC responds to all the standard LSI-11 addressing modes. However, there are four of these modes with which the PC can provide advantages for handling position independent code and unstructured data. When utilizing the PC these modes are termed immediate, absolute (or immediate deferred), relative and relative deferred, and are summarized below:

| Mode | Name              | Assembler Syntax | Function   |
|------|-------------------|------------------|--|
| 2    | Immediate         | #n               | Operand follows instruction.   |
| 3    | Absolute          | @#A              | Absolute Address of operand follows instruction.   |
| 6    | Relative          | A                | Relative Address (index value) follows the instruction.  |
| 7    | Relative Deferred | @A               | Index value (stored in the word following the instruction) is the relative address for the address of the operand. |

The reader should remember that the special PC modes are the same as modes described in 2.3 and 2.4, but the general register selected is R7, the program counter.

When a standard program is available for different users, it often is helpful to be able to load it into different areas of memory and run it there. LSI-11's can accomplish the relocation of a program very efficiently through the use of position independent code (PIC) which is written by using the PC addressing modes. If an instruction and its operands are moved in such a way that the relative distance between them is not altered, the same offset relative to the PC can be used in all positions in memory. Thus, PIC usually references locations relative to the current location.

The PC also greatly facilitates the handling of unstructured data. This is particularly true of the immediate and relative modes.

### 2.5.1 Immediate Mode

OPR #n,DD

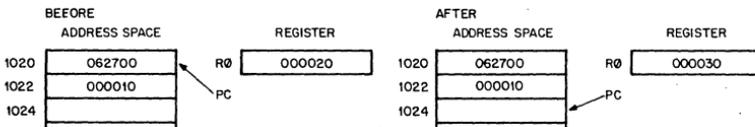
Immediate mode is equivalent to using the autoincrement mode with the PC. It provides time improvements for accessing constant operands by including the constant in the memory location immediately following the instruction word.

#### Immediate Mode Example

| Symbolic    | Octal Code       | Instruction Name |
|-------------|------------------|------------------|
| ADD # 10,R0 | 062700<br>000010 | Add              |

#### Operation:

The value 10 is located in the second word of the instruction and is added to the contents of R0. Just before this instruction is fetched and executed, the PC points to the first word of the instruction. The processor fetches the first word and increments the PC by two. The source operand mode is 27 (autoincrement the PC). Thus, the PC is used as a pointer to fetch the operand (the second word of the instruction) before being incremented by two to point to the next instruction.



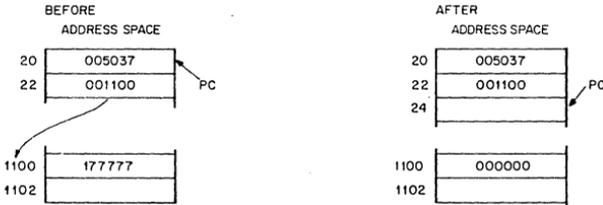
## 2.5.2 Absolute Addressing OPR @ #A

This mode is the equivalent of immediate deferred or autoincrement deferred using the PC. The contents of the location following the instruction are taken as the address of the operand. Immediate data is interpreted as an absolute address (i.e., an address that remains constant no matter where in memory the assembled instruction is executed).

### Absolute Mode Examples

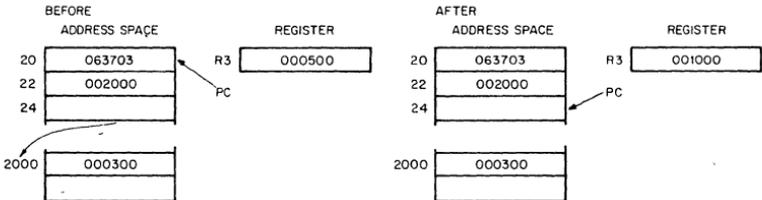
|    | Symbolic     | Octal Code       | Instruction Name |
|----|--------------|------------------|------------------|
| 1. | CLR @ # 1100 | 005037<br>001100 | Clear            |

Operation: Clear the contents of location 1100.



|    |                 |                  |
|----|-----------------|------------------|
| 2. | ADD @ # 2000,R3 | 063703<br>002000 |
|----|-----------------|------------------|

Operation: Add contents of location 2000 to R3.



### 2.5.3 Relative Addressing

OPR A or OPR X (PC),  
 where X is the location of A relative to the instruction.

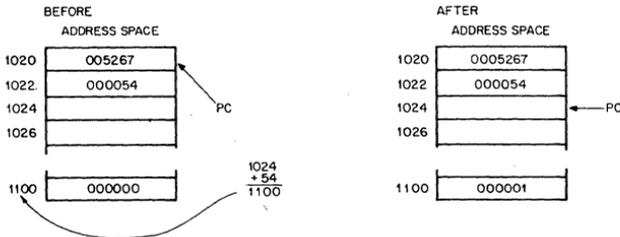
This mode is assembled as index mode using R7. The base of the address calculation, which is stored in the second word of the instruction, is not the address of the operand, but the number which, when added to the (PC), becomes the address of the operand. This mode is useful for writing position independent code (see Chapter 5) since the location referenced is always fixed relative to the PC. When instructions are to be relocated, the operand is moved by the same amount.

#### Relative Addressing Example

| Symbolic | Octal Code       | Instruction Name |
|----------|------------------|------------------|
| INC A    | 005267<br>000054 | Increment        |

#### Operation:

To increment location A, contents of memory location immediately following instruction word are added to (PC) to produce address A. Contents of A are increased by one.



### 2.5.4 Relative Deferred Addressing

OPR@A or  
 OPR@X(PC), where x is location containing address of A, relative to the instruction.

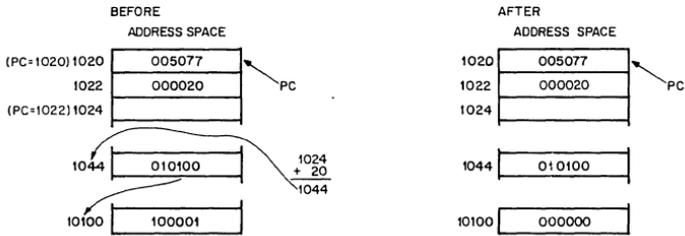
This mode is similar to the relative mode, except that the second word of the instruction, when added to the PC, contains the address of the address of the operand, rather than the address of the operand.

#### Relative Deferred Mode Example

| Symbolic | Octal Code       | Instruction Name |
|----------|------------------|------------------|
| CLR @A   | 005077<br>000020 | Clear            |

#### Operation:

Add second word of instruction to updated PC to produce address of address of operand. Clear operand.



## 2.6 USE OF STACK POINTER AS GENERAL REGISTER

The processor stack pointer (SP, Register 6) is in most cases the general register used for the stack operations related to program nesting. Auto-decrement with Register 6 "pushes" data on to the stack and autoincrement with Register 6 "pops" data off the stack. Index mode with SP permits random access of items on the stack. Since the SP is used by the processor for interrupt handling, it has a special attribute: autoincrements and autodecrements are always done in steps of two. Byte operations using the SP in this way leave odd addresses unmodified.

## 2.7 SUMMARY OF ADDRESSING MODES

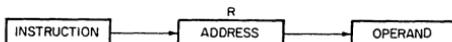
### 2.7.1 General Register Addressing

R is a general register, 0 to 7  
(R) is the contents of that register

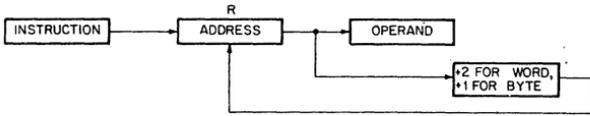
**Mode 0**                      **Register**                      OPR R                      R contains operand



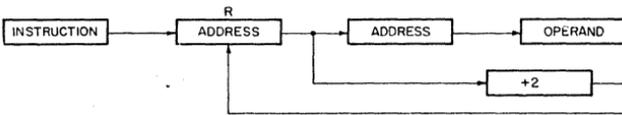
**Mode 1**                      **Register deferred**                      OPR (R)                      R contains address



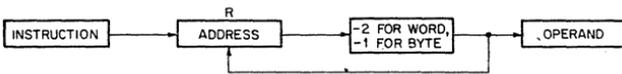
**Mode 2**                      **Autoincrement**                      **OPR (R)+**  
 R contains address, then increment (R)



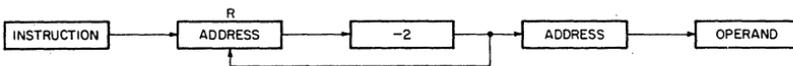
**Mode 3**    **Autoincrement deferred**    **OPR @(R)+**    R contains address of address, then increment (R) by 2



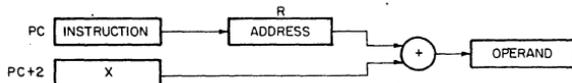
**Mode 4**                      **Autodecrement**                      **OPR -(R)**  
 Decrement (R), then R contains address



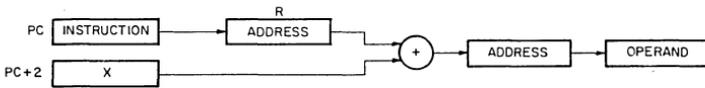
**Mode 5**    **Autodecrement deferred**                      **OPR @-(R)**                      Decrement (R) by 2, then R contains address of address



**Mode 6**                      **Index**                      **OPR X(R)**                      (R) + X is address



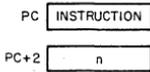
**Mode 7 Index deferred** OPR @X(R) (R) + X is address of address



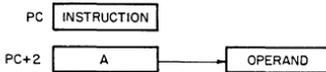
### 2.7.2 Program Counter Addressing

Register = 7

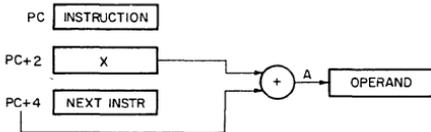
**Mode 2 Immediate** OPR #n Operand n follows instruction



**Mode 3 Absolute** OPR @#A Address A follows instruction

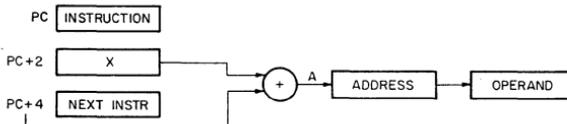


**Mode 6 Relative** OPR A  $\underbrace{PC + 4 + X}_{\text{updated PC}}$  is address



**Mode 7 Relative deferred** OPR @A

$\underbrace{PC + 4 + X}_{\text{updated PC}}$  is address of address





## INSTRUCTION SET

## 3.1 INTRODUCTION

The specification for each instruction includes the mnemonic, octal code, binary code, a diagram showing the format of the instruction, a symbolic notation describing its execution and the effect on the condition codes, a description, special comments, and examples.

**MNEMONIC:** This is indicated at the top corner of each page. When the word instruction has a byte equivalent, the byte mnemonic is also shown.

**INSTRUCTION FORMAT:** A diagram accompanying each instruction shows the octal op code, the binary op code, and bit assignments. (Note that in byte instructions the most significant bit (bit 15) is always a 1.)

**SYMBOLS:**

( ) = contents of

SS or src = source address

DD or dst = destination address

loc = location

← = becomes

↑ = "is popped from stack"

↓ = "is pushed onto stack"

∧ = boolean AND

∨ = boolean OR

⊕ = exclusive OR

~ = boolean not

Reg or R = register

B = Byte

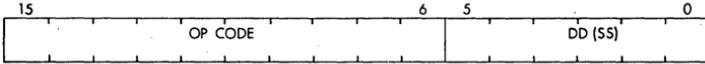
■ =  $\begin{cases} 0 & \text{for word} \\ 1 & \text{for byte} \end{cases}$

, = concatenated

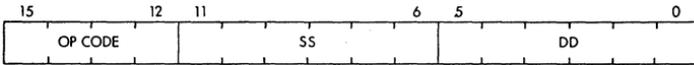
### 3.2 INSTRUCTION FORMATS

The following formats include all instructions used in the LSI-11. Refer to individual instructions for more detailed information.

- Single Operand Group (CLR, CLRB, COM, COMB, INC, INCB, DEC, DECB, NEG, NEGB, ADC, ADCB, SBC, SBCB, TST, TSTB, ROR, RORB, ROL, ROLB, ASR, ASRB, ASL, ASLB, JMP, SWAB, MFPS, MTPS, SXT, XOR)

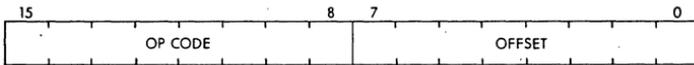


- Double Operand Group (BIT, BITB, BIC, BICB, BIS, BISB, ADD, SUB, MOV, MOVB, CMP, CMPB)

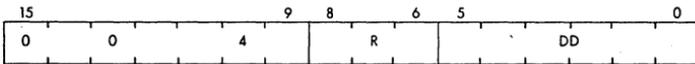


- Program Control Group

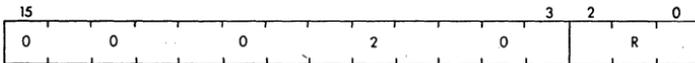
- Branch (all branch instructions)



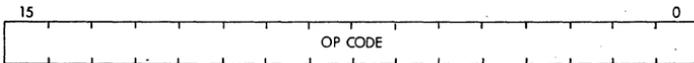
- Jump To Subroutine (JSR)



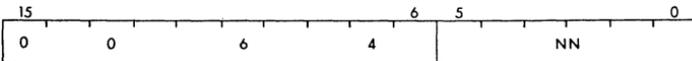
- Subroutine Return (RTS)



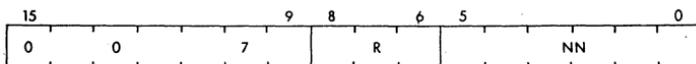
- Traps (break point, IOT, EMT, TRAP, BPT)



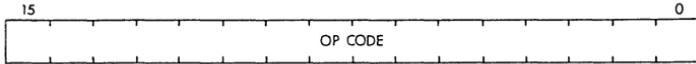
- Mark (MARK)



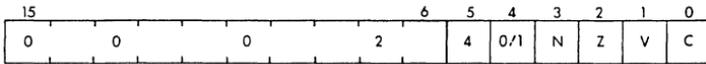
- Subtract 1 and branch (if = 0)(SOB)



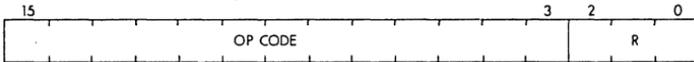
4. Operate Group (HALT, WAIT, RTI, RESET, RTT, NOP)



5. Condition Code Operators (all condition code instructions)

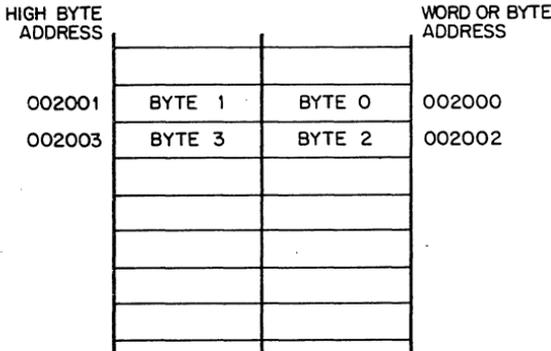


6. Fixed and Floating Point Arithmetic (optional EIS/FIS) (FADD, FSUB, FMUL, FDIV, MUL, DIV, ASH, ASHC)



**Byte Instructions**

The LSI-11 includes a full complement of instructions that manipulate byte operands. Since all LSI-11 addressing is byte-oriented, byte manipulation addressing is straightforward. Byte instructions with autoincrement or autodecrement direct addressing cause the specified register to be modified by one to point to the next byte of data. Byte operations in register mode access the low-order byte of the specified register. These provisions enable the LSI-11 to perform as either a word or byte processor. The numbering scheme for word and byte addresses in memory is:



The most significant bit (Bit 15) of the instruction word is set to indicate a byte instruction.

Example:

| Symbolic | Octal  |            |
|----------|--------|------------|
| CLR      | 0050DD | Clear Word |
| CLRB     | 1050DD | Clear Byte |

### 3.3 LIST OF INSTRUCTIONS

The LSI-11 instruction set is shown in the following sequence.

#### SINGLE OPERAND

| Mnemonic                  | Instruction                  | Op Code | Page |
|---------------------------|------------------------------|---------|------|
| <b>General</b>            |                              |         |      |
| CLR(B)                    | clear dst .....              | ■050DD  | 3-6  |
| COM(B)                    | complement dst .....         | ■051DD  | 3-7  |
| INC(B)                    | increment dst .....          | ■052DD  | 3-7  |
| DEC(B)                    | decrement dst .....          | ■053DD  | 3-8  |
| NEG(B)                    | negate dst .....             | ■054DD  | 3-8  |
| TST(B)                    | test dst .....               | ■057DD  | 3-9  |
| <b>Shift &amp; Rotate</b> |                              |         |      |
| ASR(B)                    | arithmetic shift right ..... | ■062DD  | 3-10 |
| ASL(B)                    | arithmetic shift left .....  | ■063DD  | 3-11 |
| ROR(B)                    | rotate right .....           | ■060DD  | 3-11 |
| ROL(B)                    | rotate left .....            | ■061DD  | 3-12 |
| SWAB                      | swap bytes .....             | 0003DD  | 3-13 |
| <b>Multiple Precision</b> |                              |         |      |
| ADC(B)                    | add carry .....              | ■055DD  | 3-15 |
| SBC(B)                    | subtract carry .....         | ■056DD  | 3-15 |
| SXT                       | sign extend .....            | 0067DD  | 3-16 |
| <b>PS WORD OPERATORS</b>  |                              |         |      |
| MFPS                      | move byte from PS .....      | 1067DD  | 3-17 |
| MTPS                      | move byte to PS .....        | 1064SS  | 3-17 |

#### DOUBLE OPERAND

|                |                                  |        |      |
|----------------|----------------------------------|--------|------|
| <b>General</b> |                                  |        |      |
| MOV(B)         | move source to destination ..... | ■1SSDD | 3-18 |
| CMP(B)         | compare src to dst .....         | ■2SSDD | 3-19 |
| ADD            | add src to dst .....             | 06SSDD | 3-20 |
| SUB            | subtract src from dst .....      | 16SSDD | 3-20 |
| <b>Logical</b> |                                  |        |      |
| BIT(B)         | bit test .....                   | ■3SSDD | 3-21 |
| BIC(B)         | bit clear .....                  | ■4SSDD | 3-22 |
| BIS(B)         | bit set .....                    | ■5SSDD | 3-23 |
| XOR            | exclusive or .....               | 074RDD | 3-24 |

## PROGRAM CONTROL

| Mnemonic                           | Instruction  | Op Code<br>or<br>Base Code | Page |
|------------------------------------|--|----------------------------|------|
| <b>Branch</b>                      |  |                            |      |
| BR                                 | branch (unconditional) .....                       | 000400                     | 3-25 |
| BNE                                | branch if not equal (to zero) .....                | 001000                     | 3-26 |
| BEQ                                | branch if equal (to zero) .....                    | 001400                     | 3-26 |
| BPL                                | branch if plus .....                               | 100000                     | 3-27 |
| BMI                                | branch if minus .....                              | 100400                     | 3-27 |
| BVC                                | branch if overflow is clear .....                  | 102000                     | 3-28 |
| BVS                                | branch if overflow is set .....                    | 102400                     | 3-28 |
| BCC                                | branch if carry is clear .....                     | 103000                     | 3-28 |
| BCS                                | branch if carry is set .....                       | 103400                     | 3-29 |
| <b>Signed Conditional Branch</b>   |  |                            |      |
| BGE                                | branch is greater than or equal<br>(to zero) ..... | 002000                     | 3-30 |
| BLT                                | branch if less than (zero) .....                   | 002400                     | 3-30 |
| BGT                                | branch if greater than (zero) .....                | 003000                     | 3-31 |
| BLE                                | branch if less than or equal (to zero) ..          | 003400                     | 3-31 |
| <b>Unsigned Conditional Branch</b> |  |                            |      |
| BHI                                | branch if higher .....                             | 101000                     | 3-32 |
| BLOS                               | branch if lower or same .....                      | 101400                     | 3-32 |
| BHIS                               | branch if higher or same .....                     | 103000                     | 3-33 |
| BLO                                | branch if lower .....                              | 103400                     | 3-33 |
| <b>Jump &amp; Subroutine</b>       |  |                            |      |
| JMP                                | jump .....   | 0001DD                     | 3-34 |
| JSR                                | jump to subroutine .....                           | 004RDD                     | 3-35 |
| RTS                                | return from subroutine .....                       | 00020R                     | 3-37 |
| MARK                               | mark .....   | 006400                     | 3-38 |
| SOB                                | subtract one and branch (if $\neq 0$ ) .....       | 077R00                     | 3-39 |
| <b>Trap &amp; Interrupt</b>        |  |                            |      |
| EMT                                | emulator trap .....                                | 104000—104377              | 3-40 |
| TRAP                               | trap .....   | 104400—104777              | 3-41 |
| BPT                                | breakpoint trap .....                              | 000003                     | 3-42 |
| IOT                                | input/output trap .....                            | 000004                     | 3-42 |
| RTI                                | return from interrupt .....                        | 000002                     | 3-43 |
| RTT                                | return from interrupt .....                        | 000006                     | 3-43 |
| <b>MISCELLANEOUS</b>               |  |                            |      |
| HALT                               | halt .....   | 000000                     | 3-46 |
| WAIT                               | wait for interrupt .....                           | 000001                     | 3-46 |
| RESET                              | reset external bus .....                           | 000005                     | 3-47 |
| <b>RESERVED INSTRUCTIONS</b>       |  |                            |      |
|                                    |  | 00021R                     | 3-47 |
|                                    |  | 00022                      | 3-48 |

## CONDITION CODE OPERATORS

|     |                         |        |      |
|-----|-------------------------|--------|------|
| CLC | clear C .....           | 000241 | 3-49 |
| CLV | clear V .....           | 000242 | 3-49 |
| CLZ | clear Z .....           | 000244 | 3-49 |
| CLN | clear N .....           | 000250 | 3-49 |
| CCC | clear all CC bits ..... | 000257 | 3-49 |
| SEC | set C .....             | 000261 | 3-49 |
| SEV | set V .....             | 000262 | 3-49 |
| SEZ | set Z .....             | 000264 | 3-49 |
| SEN | set N .....             | 000270 | 3-49 |
| SCC | set all CC bits .....   | 000277 | 3-49 |
| NOP | no operation .....      | 00240  | 3-49 |

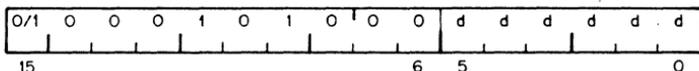
## 3.4 SINGLE OPERAND INSTRUCTIONS

### General

## CLR CLRB

clear destination

■050DD



**Operation:** (dst) ← 0

**Condition Codes:** N: cleared  
Z: set  
V: cleared  
C: cleared

**Description:** Word: Contents of specified destination are replaced with zeroes.  
Byte: Same

**Example:** CLR R1

|               |               |
|---------------|---------------|
| Before        | After         |
| (R1) = 177777 | (R1) = 000000 |
| N Z V C       | N Z V C       |
| 1 1 1 1       | 0 1 0 0       |

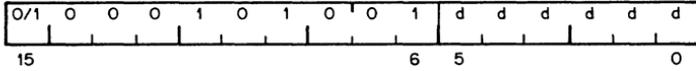
### NOTE

CLR and CLRB perform a DATIO bus cycle as the last bus cycle during the instruction execution. The DATI portion of the DATIO cycle is a "don't care" condition, but the addressed memory or device must be capable of responding to the DATI cycle to avoid a bus timeout error.

# COM COMB

complement dst

■051DD



**Operation:** (dst) ← ~(dst)

**Condition Codes:** N: set if most significant bit of result is set; cleared otherwise  
 Z: set if result is 0; cleared otherwise  
 V: cleared  
 C: set

**Description:** Replaces the contents of the destination address by their logical complement (each bit equal to 0 is set and each bit equal to 1 is cleared)  
 Byte: Same

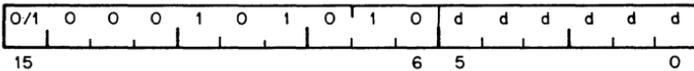
**Example:** COM R0

|               |               |
|---------------|---------------|
| Before        | After         |
| (R0) = 013333 | (R0) = 164444 |
| N Z V C       | N Z V C       |
| 0 1 1 0       | 1 0 0 1       |

# INC INCB

increment dst

■052DD



**Operation:** (dst) ← (dst) + 1

**Condition Codes:** N: set if result is < 0; cleared otherwise  
 Z: set if result is 0; cleared otherwise  
 V: set if (dst) held 077777; cleared otherwise  
 C: not affected

**Description:** Word: Add one to contents of destination  
 Byte: Same

**Example:**

**INC R2**

Before  
(R2) = 000333

After  
(R2) = 000334

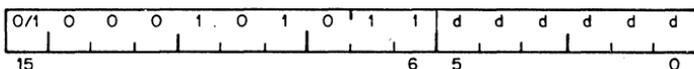
N Z V C  
0 0 0 0

N Z V C  
0 0 0 0

## DEC DECB

decrement dst

■053DD



**Operation:** (dst) ← (dst) - 1

**Condition Codes:** N: set if result is <0; cleared otherwise  
Z: set if result is 0; cleared otherwise  
V: set if (dst) was 100000; cleared otherwise  
C: not affected

**Description:** Word: Subtract 1 from the contents of the destination  
Byte: Same

**Example:**

**DEC R5**

Before  
(R5) = 000001

After  
(R5) = 000000

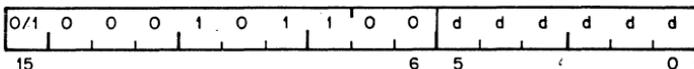
N Z V C  
1 0 0 0

N Z V C  
0 1 0 0

## NEG NEGB

negate dst

■054DD



**Operation:** (dst) ← -(dst)

**Condition Codes:** N: set if the result is <0; cleared otherwise  
Z: set if result is 0; cleared otherwise  
V: set if the result is 100000; cleared otherwise  
C: cleared if the result is 0; set otherwise

**Description:** Word: Replaces the contents of the destination address by its two's complement. Note that 100000 is replaced by itself -(in two's complement notation the most negative number has no positive counterpart).  
Byte: Same

**Example:**

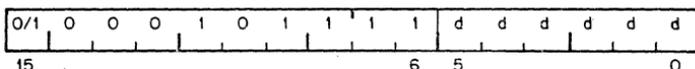
NEG R0

| Before        | After         |
|---------------|---------------|
| (R0) = 000010 | (R0) = 177770 |
| NZVC          | NZVC          |
| 0000          | 1001          |

TST  
TSTB

test dst

057DD



**Operation:** (dst) ← (dst)

**Condition Codes:** N: set if the result is <0; cleared otherwise  
Z: set if result is 0; cleared otherwise  
V: cleared  
C: cleared

**Description:** Word: Sets the condition codes N and Z according to the contents of the destination address, contents of dst remains unmodified  
Byte: Same

**Example:**

TST R1

| Before        | After         |
|---------------|---------------|
| (R1) = 012340 | (R1) = 012340 |
| NZVC          | NZVC          |
| 0011          | 0000          |

## Shifts

Scaling data by factors of two is accomplished by the shift instructions:

**ASR** - Arithmetic shift right

**ASL** - Arithmetic shift left

The sign bit (bit 15) of the operand is reproduced in shifts to the right. The low order bit is filled with 0 in shifts to the left. Bits shifted out of the C bit, as shown in the following examples, are lost.

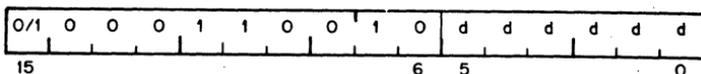
## Rotates

The rotate instructions operate on the destination word and the C bit as though they formed a 17-bit "circular buffer." These instructions facilitate sequential bit testing and detailed bit manipulation.

# ASR ASRB

arithmetic shift right

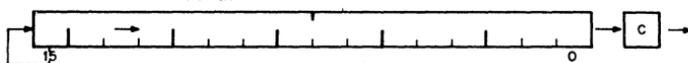
■062DD



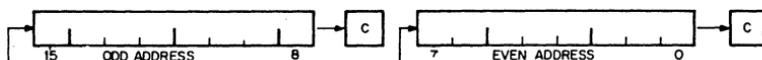
**Operation:** (dst) ← (dst) shifted one place to the right

**Condition Codes:** N: set if the high-order bit of the result is set (result < 0); cleared otherwise  
Z: set if the result = 0; cleared otherwise  
V: loaded from the Exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)  
C: loaded from low-order bit of the destination

**Description:** Word: Shifts all bits of the destination right one place. Bit 15 is reproduced. The C-bit is loaded from bit 0 of the destination. ASR performs signed division of the destination by two.  
Word:



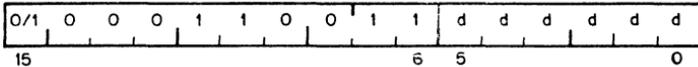
Byte:



# ASL ASLB

arithmetic shift left

■063DD

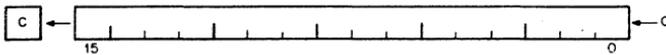


**Operation:** (dst) ← (dst) shifted one place to the left

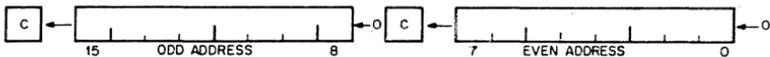
**Condition Codes:** N: set if high-order bit of the result is set (result < 0); cleared otherwise  
 Z: set if the result = 0; cleared otherwise  
 V: loaded with the exclusive OR of the N-bit and C-bit (as set by the completion of the shift operation)  
 C: loaded with the high-order bit of the destination

**Description:** Word: Shifts all bits of the destination left one place. Bit 0 is loaded with an 0. The C-bit of the status word is loaded from the most significant bit of the destination. ASL performs a signed multiplication of the destination by 2 with overflow indication.

Word:



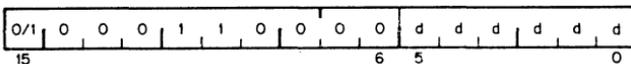
Byte:



# ROR RORB

rotate right

■060DD



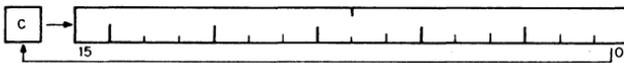
**Operation:** (dst) ← (dst) rotate right one place

**Condition Codes:** N: set if the high-order bit of the result is set (result < 0); cleared otherwise  
 Z: set if all bits of result = 0; cleared otherwise  
 V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)  
 C: loaded with the low-order bit of the destination

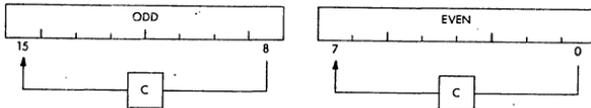
**Description:** Rotates all bits of the destination right one place. Bit 0 is loaded into the C-bit and the previous contents of the C-bit are loaded into bit 15 of the destination.  
 Byte: Same

**Example:**

Word:



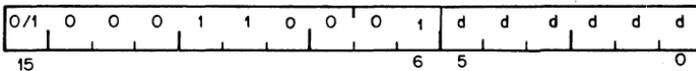
Byte:



## ROL ROLB

rotate left

■061DD



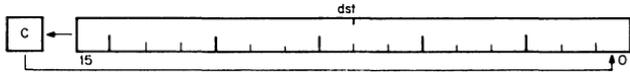
**Operation:** (dst) ← (dst)  
 rotate left one place

**Condition Codes:** N: set if the high-order bit of the result word is set (result < 0); cleared otherwise  
 Z: set if all bits of the result word = 0; cleared otherwise  
 V: loaded with the Exclusive OR of the N-bit and C-bit (as set by the completion of the rotate operation)  
 C: loaded with the high-order bit of the destination

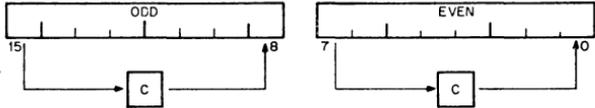
**Description:** Word: Rotate all bits of the destination left one place. Bit 15 is loaded into the C-bit of the status word and the previous contents of the C-bit are loaded into Bit 0 of the destination.  
 Byte: Same

**Example:**

Word:



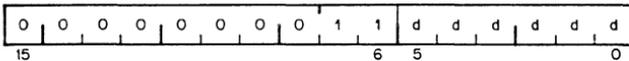
Bytes:



## SWAB

swap bytes

0003DD



**Operation:** Byte 1/Byte 0  $\leftrightarrow$  Byte 0/Byte 1

**Condition Codes:** N: set if high-order bit of low-order byte (bit 7) of result is set; cleared otherwise  
Z: set if low-order byte of result = 0; cleared otherwise  
V: cleared  
C: cleared

**Description:** Exchanges high-order byte and low-order byte of the destination word (destination must be a word address).

**Example:** SWAB R1

Before  
(R1) = 077777

After  
(R1) = 177577

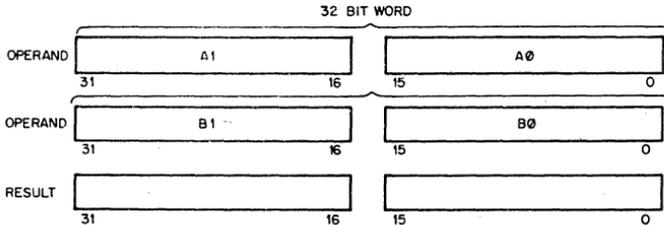
NZVC  
1111

NZVC  
0000

### Multiple Precision

It is sometimes necessary to do arithmetic on operands considered as multiple words or bytes. The LSI-11 makes special provision for such operations with the instructions ADC (Add Carry) and SBC (Subtract Carry) and their byte equivalents.

For example two 16-bit words may be combined into a 32-bit double precision word and added or subtracted as shown below:



### Example:

The addition of -1 and -1 could be performed as follows:

$$-1 = 3777777777$$

$$(R1) = 177777 \quad (R2) = 177777 \quad (R3) = 177777 \quad (R4) = 177777$$

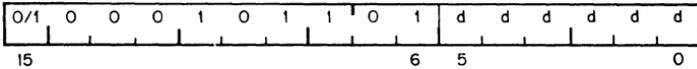
```
ADD R1,R2
ADC R3
ADD R4,R3
```

1. After (R1) and (R2) are added, 1 is loaded into the C bit
2. ADC instruction adds C bit to (R3); (R3) = 0
3. (R3) and (R4) are added
4. Result is 3777777776 or -2

# ADC ADCB

add carry

■055DD



**Operation:**  $(dst) \leftarrow (dst) + (C \text{ bit})$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: set if (dst) was 077777 and (C) was 1; cleared otherwise  
C: set if (dst) was 177777 and (C) was 1; cleared otherwise

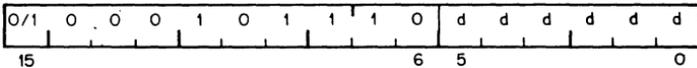
**Description:** Adds the contents of the C-bit into the destination. This permits the carry from the addition of the low-order words to be carried into the high-order result.  
Byte: Same

**Example:** Double precision addition may be done with the following instruction sequence:  
ADD A0,B0 ; add low-order parts  
ADC B1 ; add carry into high-order  
ADD A1,B1 ; add high order parts

# SBC SBCB

subtract carry

■056DD



**Operation:**  $(dst) \leftarrow (dst) - (C)$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
Z: set if result 0; cleared otherwise  
V: set if (dst) was 100000; cleared otherwise  
C: set if (dst) was 0 and C was 1; cleared otherwise

**Description:** Word: Subtracts the contents of the C-bit from the destination. This permits the carry from the subtraction of two low-order words to be subtracted from the high order part of the result.  
 Byte: Same

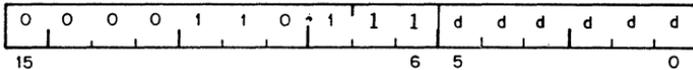
**Example:** Double precision subtraction is done by:

```
SUB  A0,B0
SBC  B1
SUB  A1,B1
```

## SXT

sign extend

0067DD



**Operation:** (dst) ← 0 if N-bit is clear  
 (dst) ← -1 N-bit is set

**Condition Codes:** N: unaffected  
 Z: set if N-bit clear  
 V: cleared  
 C: unaffected

**Description:** If the condition code bit N is set then a -1 is placed in the destination operand; if N bit is clear, then a 0 is placed in the destination operand. This instruction is particularly useful in multiple precision arithmetic because it permits the sign to be extended through multiple words.

**Example:** SXT A

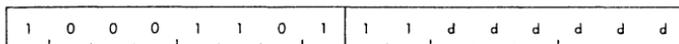
|         |        |         |        |
|---------|--------|---------|--------|
|         | Before |         | After  |
| ( A ) = | 012345 | ( A ) = | 177777 |
|         | NZVC   |         | NZVC   |
|         | 1 000  |         | 1 000  |

### 3.5 PS WORD OPERATORS

## MFPS

Move byte From Processor Status word

1067DD



**Operation:** (dst) ← PSW  
dst lower 8 bits

#### Condition Code

**Bits:** N = set if PSW bit 7 = 1; cleared otherwise  
Z = set if PS <0:7> = 0; cleared otherwise  
V = cleared  
C = not affected

**Description:** The 8 bit contents of the PS are moved to the effective destination. If destination is mode 0, PS bit 7 is sign extended through upper byte of the register. The destination operand address is treated as a byte address.

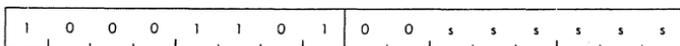
**Example:** MFPS R0

|    | before   | after    |
|----|----------|----------|
| R0 | [0]      | [000014] |
| PS | [000014] | [000000] |

## MTPS

Move byte To Processor Status word

1064SS



**Operation:** PSW ← (SRC)

**Condition Codes:** Set according to effective SRC operand bits 0-3

**Description:** The 8 bits of the effective operand replaces the current contents of the PSW. The source operand address is treated as a byte address.  
Note that the T bit (PSW bit 4) cannot be set with this instruction. The SRC operand remains unchanged. This instruction can be used to change the priority bit (PSW bit 7) in the PSW

### NOTE

When executing the MTPS instruction, the LSI-11 processor fetches the source operand via the DATIO bus cycle, rather than the DATI bus cycle. If the source operand is contained in a PROM or ROM location, a bus error (timeout) will occur because the processor will attempt to write into the addressed location after fetching the operand. When using the MTPS instruction in programs that will be stored in PROM or ROM, refer to Section I, Chapter 7, paragraph 7.3.

## 3.6 DOUBLE OPERAND INSTRUCTIONS

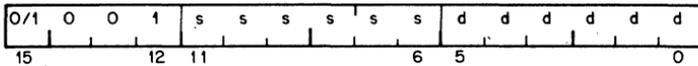
Double operand instructions provide an instruction (and time) saving facility since they eliminate the need for "load" and "save" sequences such as those used in accumulator-oriented machines.

### General

## MOV MOVB

move source to destination

■1SSDD



**Operation:** (dst) ← (src)

**Condition Codes:** N: set if (src) < 0; cleared otherwise  
Z: set if (src) = 0; cleared otherwise  
V: cleared  
C: not affected

**Description:** Word: Moves the source operand to the destination location. The previous contents of the destination are lost. The contents of the source address are not affected.  
Byte: Same as MOV. The MOVB (unique among byte instructions) extends the most significant bit of the low order byte (sign extension). Otherwise MOVB operates on bytes exactly as MOV operates on words.

**Example:** MOV XXX,R1 ; loads Register 1 with the contents of memory location; XXX represents a programmer-defined mnemonic used to represent a memory location

MOV #20,R0 ; loads the number 20 into Register 0; "#" indicates that the value 20 is the operand

MOV @ #20, -(R6) ; pushes the operand contained in location 20 onto the stack

MOV (R6) +, @ #177566 ; pops the operand off a stack and moves it into memory location 177566 (terminal print buffer)

MOV R1, R3 ; performs an inter register transfer

MOVB @ #177562, @ #177566 ; moves a character from terminal keyboard buffer to terminal printer buffer.

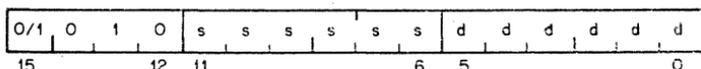
#### NOTE

The MOVB instruction performs a DATIOB bus cycle as the last bus cycle during instruction execution, even though a DATOB bus cycle would be sufficient. The DATI portion of the DATIOB bus cycle is a "don't care" condition, but the addressed memory or device must be capable of responding to the DATI cycle to avoid a bus timeout error. The MOV instruction performs only the DATO cycle as the last bus cycle.

**CMP**  
**CMPB**

compare src to dst

▣2SSDD



**Operation:** (src)-(dst)

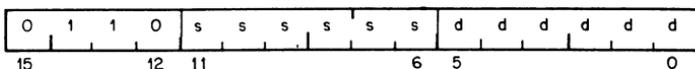
**Condition Codes:** N: set if result <0; cleared otherwise  
Z: set if result =0; cleared otherwise  
V: set if there was arithmetic overflow; that is, operands were of opposite signs and the sign of the destination was the same as the sign of the result; cleared otherwise  
C: cleared if there was a carry from the most significant bit of the result; set otherwise

**Description:** Compares the source and destination operands and sets the condition codes, which may then be used for arithmetic and logical conditional branches. Both operands are unaffected. The only action is to set the condition codes. The compare is customarily followed by a conditional branch instruction. Note that unlike the subtract instruction the order of operation is (src)-(dst), not (dst)-(src).

## ADD

add src to dst

06SSDD



**Operation:**  $(dst) \leftarrow (src) + (dst)$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise

Z: set if result  $= 0$ ; cleared otherwise

V: set if there was arithmetic overflow as a result of the operation; that is both operands were of the same sign and the result was of the opposite sign; cleared otherwise

C: set if there was a carry from the most significant bit of the result; cleared otherwise

**Description:** Adds the source operand to the destination operand and stores the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. Two's complement addition is performed.

Note: There is no equivalent byte mode.

**Examples:** Add to register:           ADD    20,R0

Add to memory:            ADD   R1,XXX

Add register to register:  ADD   R1,R2

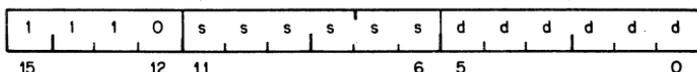
Add memory to memory:   ADD@ # 17750,XXX

XXX is a programmer-defined mnemonic for a memory location.

## SUB

subtract src from dst

16SSDD



**Operation:**  $(dst) \leftarrow (dst) - (src)$

**Condition Codes:** N: set if result <0; cleared otherwise  
 Z: set if result =0; cleared otherwise  
 V: set if there was arithmetic overflow as a result of the operation, that is if operands were of opposite signs and the sign of the source was the same as the sign of the result; cleared otherwise  
 C: cleared if there was a carry from the most significant bit of the result; set otherwise

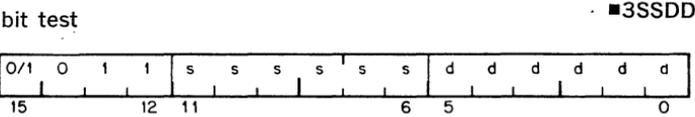
**Description:** Subtracts the source operand from the destination operand and leaves the result at the destination address. The original contents of the destination are lost. The contents of the source are not affected. In double-precision arithmetic the C-bit, when set, indicates a "borrow"

**Example:** SUB R1,R2

|               |               |
|---------------|---------------|
| Before        | After         |
| (R1) = 011111 | (R1) = 011111 |
| (R2) = 012345 | (R2) = 001234 |
| N Z V C       | N Z V C       |
| 1 1 1 1       | 0 0 0 0       |

**Logical**  
 These instructions have the same format as the double operand arithmetic group. They permit operations on data at the bit level.

## BIT BITB



**Operation:** (src)  $\wedge$  (dst)

**Condition Codes:** N: set if high-order bit of result set; cleared otherwise  
 Z: set if result =0; cleared otherwise  
 V: cleared  
 C: not affected

**Description:** Performs logical "and" comparison of the source and destination operands and modifies condition codes accordingly. Neither the source nor destination operands are affected. The BIT instruction may be used to test whether any of the corresponding bits that are set in the destination are also set in the source or whether all corresponding bits set in the destination are clear in the source.

**Example:** BIT #30.R3 ; test bits 3 and 4 of R3 to see  
; if both are off

R3 = 0 000 000 000 011 000

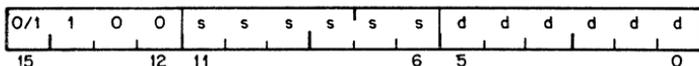
Before  
NZVC  
1111

After  
NZVC  
0001

## BIC BICB

bit clear

■4SSDD



**Operation:** (dst) ←  $\sim(\text{src}) \wedge (\text{dst})$

**Condition Codes:** N: set if high order bit of result set; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: cleared  
C: not affected

**Description:** Clears each bit in the destination that corresponds to a set bit in the source. The original contents of the destination are lost. The contents of the source are unaffected.

**Example:** BIC R3,R4

|        |        |               |
|--------|--------|---------------|
|        | Before | After         |
| (R3) = | 001234 | (R3) = 001234 |
| (R4) = | 001111 | (R4) = 000101 |
|        | NZVC   | NZVC          |
|        | 1111   | 0001          |

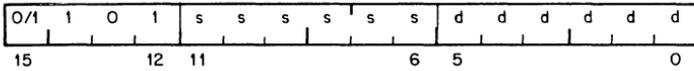
**Before:** (R3)=0 000 001 010 011 100  
(R4)=0 000 001 001 001 001

**After:** (R3)=0 000 000 001 000 001

# BIS BISB

bit set

■5SSDD



**Operation:** (dst) ← (src) v (dst)

**Condition Codes:** N: set if high-order bit of result set, cleared otherwise  
 Z: set if result = 0; cleared otherwise  
 V: cleared  
 C: not affected

**Description:** Performs "Inclusive OR" operation between the source and destination operands and leaves the result at the destination address; that is, corresponding bits set in the source are set in the destination. The contents of the destination are lost.

**Example:** BIS R0,R1

|        |         |        |         |
|--------|---------|--------|---------|
|        | Before  |        | After   |
| (R0) = | 001234  | (R0) = | 001234  |
| (R1) = | 001111  | (R1) = | 001335  |
|        | N Z V C |        | N Z V C |
|        | 0 0 0 0 |        | 0 0 0 0 |

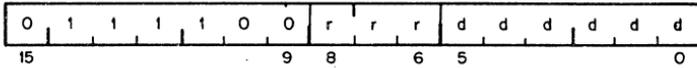
**Before:** (R0)=0 000 001 010 011 100  
 (R1)=0 000 001 001 001 001

**After:** (R1)=0 000 001 011 011 101

# XOR

exclusive OR

074RDD



**Operation:** (dst) ← Rv(dst)

**Condition Codes:** N: set if the result < 0; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: cleared  
C: unaffected

**Description:** The exclusive OR of the register and destination operand is stored in the destination address. Contents of register are unaffected. Assembler format is: XOR R,D

**Example:** XOR R0,R2

|                | Before                     | After                      |
|----------------|----------------------------|----------------------------|
| (R0) =         | 001234                     | 001234                     |
| (R2) =         | 001111                     | 000325                     |
| NZVC           |                            | NZVC                       |
|                | 1111                       | 0001                       |
| <b>Before:</b> | (R0)=0 000 001 010 011 100 | (R2)=0 000 001 001 001 001 |
| <b>After:</b>  | (R2)=0 000 000 011 010 101 |                            |

## 3.7 PROGRAM CONTROL INSTRUCTIONS

### Branches

These instructions cause a branch to a location defined by the sum of the offset (multiplied by 2) and the current contents of the Program Counter if:

- the branch instruction is unconditional
- it is conditional and the conditions are met after testing the condition codes (NZVC)

The offset is the number of words from the current contents of the PC forward or backward. Note that the current contents of the PC point to the word following the branch instruction.

Although the offset expresses a byte address the PC is expressed in words. The offset is automatically multiplied by two and sign extended to express words before it is added to the PC. Bit 7 is the sign of the offset.

If it is set, the offset is negative and the branch is done in the backward direction. Similarly if it is not set, the offset is positive and the branch is done in the forward direction.

The 8-bit offset allows branching in the backward direction by  $200_8$  words (400 bytes) from the current PC, and in the forward direction by  $177_8$  words (376 bytes) from the current PC.

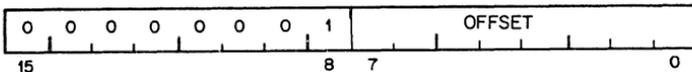
The PDP-11 assembler handles address arithmetic for the user and computes and assembles the proper offset field for branch instructions in the form:

Bxx loc

Where "Bxx" is the branch instruction and "loc" is the address to which the branch is to be made. The assembler gives an error indication in the instruction if the permissible branch range is exceeded. Branch instructions have no effect on condition codes. Conditional branch instructions where the branch condition is not met, are treated as NO OP's.

## BR

branch (unconditional) 000400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$

**Condition Codes:** Unaffected

**Description:** Provides a way of transferring program control within a range of  $-128_{10}$  to  $+127_{10}$  words with a one word instruction.

New PC address = updated PC + (2 X offset)

Updated PC = address of branch instruction + 2

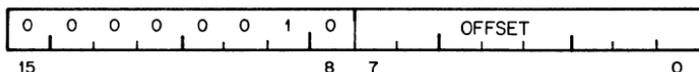
**Example:** With the Branch instruction at location 500, the following offsets apply.

| New PC Address | Offset Code | Offset (decimal) |
|----------------|-------------|------------------|
| 474            | 375         | -3               |
| 476            | 376         | -2               |
| 500            | 377         | -1               |
| 502            | 000         | 0                |
| 504            | 001         | +1               |
| 506            | 002         | +2               |

## BNE

branch if not equal (to zero)

001000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $Z = 0$

**Condition Codes:** Unaffected

**Description:** Tests the state of the Z-bit and causes a branch if the Z-bit is clear. BNE is the complementary operation to BEQ. It is used to test inequality following a CMP, to test that some bits set in the destination were also in the source, following a BIT operation, and generally, to test that the result of the previous operation was not zero.

**Example:** `CMP A,B ; compare A and B`  
`BNE C ; branch if they are not equal`

will branch to C if  $A \neq B$

and the sequence

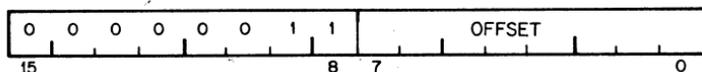
`ADD A,B ; add A to B`  
`BNE C ; Branch if the result is not equal to 0`

will branch to C if  $A + B \neq 0$

## BEQ

branch if equal (to zero)

001400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $Z = 1$

**Condition Codes:** Unaffected

**Description:**

Tests the state of the Z-bit and causes a branch if Z is set. As an example, it is used to test equality following a CMP operation, to test that no bits set in the destination were also set in the source following a BIT operation, and generally, to test that the result of the previous operation was zero.

**Example:**

```
CMP  A,B           ; compare A and B
BEQ  C             ; branch if they are equal
```

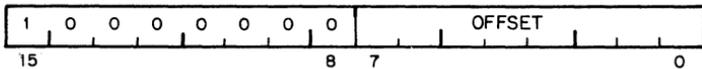
will branch to C if  $A = B$  ( $A - B = 0$ )  
and the sequence

```
ADD  A,B           ; add A to B
BEQ  C             ; branch if the result = 0
```

will branch to C if  $A + B = 0$ .

**BPL**

branch if plus 100000 Plus offset



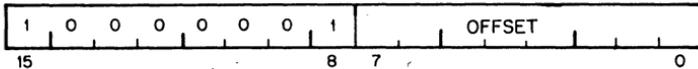
**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $N = 0$

**Condition Codes:** Unaffected

**Description:** Tests the state of the N-bit and causes a branch if N is clear, (positive result). BPL is the complementary operation of BMI.

**BMI**

branch if minus 100400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $N = 1$

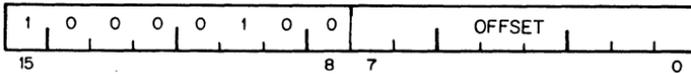
**Condition Codes:** Unaffected

**Description:** Tests the state of the N-bit and causes a branch if N is set. It is used to test the sign (most significant bit) of the result of the previous operation), branching if negative. BMI is the complementary function of BPL.

# BVC

branch if overflow is clear

102000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $V = 0$

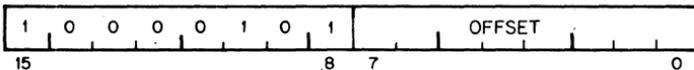
**Condition Codes:** Unaffected

**Description:** Tests the state of the V-bit and causes a branch if the V bit is clear. BVC is complementary operation to BVS.

# BVS

branch if overflow is set

102400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $V = 1$

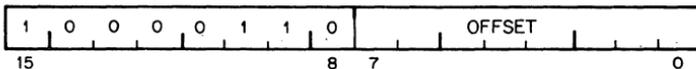
**Condition Codes:** Unaffected

**Description:** Tests the state of V-bit (overflow) and causes a branch if the V bit is set. BVS is used to detect arithmetic overflow in the previous operation.

# BCC

branch if carry is clear

103000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C = 0$

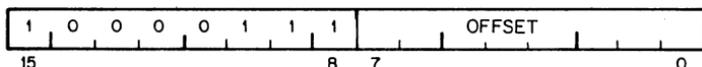
**Condition Codes:** Unaffected

**Description:** Tests the state of the C-bit and causes a branch if C is clear. BCC is the complementary operation to BCS.

# BCS

branch if carry is set

103400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C = 1$

**Condition Codes:** Unaffected

**Description:** Tests the state of the C-bit and causes a branch if C is set. It is used to test for a carry in the result of a previous operation.

## Signed Conditional Branches

Particular combinations of the condition code bits are tested with the signed conditional branches. These instructions are used to test the results of instructions in which the operands were considered as signed (two's complement) values.

Note that the sense of signed comparisons differs from that of unsigned comparisons in that in signed 16-bit, two's complement arithmetic the sequence of values is as follows:

|          |        |
|----------|--------|
| largest  | 077777 |
|          | 077776 |
| positive | .      |
|          | .      |
|          | 000001 |
|          | 000000 |
|          | 177777 |
|          | 177776 |
| negative | .      |
|          | .      |
|          | 100001 |
| smallest | 100000 |

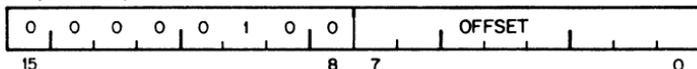
whereas in unsigned 16-bit arithmetic the sequence is considered to be

|         |        |
|---------|--------|
| highest | 177777 |
|         | .      |
|         | .      |
|         | .      |
|         | .      |
|         | 000002 |
|         | 000001 |
| lowest  | 000000 |

## BGE

branch if greater than or equal  
(to zero)

002000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $N \vee V = 0$

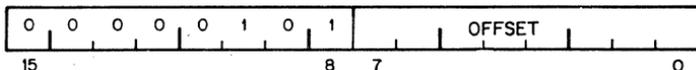
**Condition Codes:** Unaffected

**Description:** Causes a branch if N and V are either both clear or both set. BGE is the complementary operation to BLT. Thus BGE will always cause a branch when it follows an operation that caused addition of two positive numbers. BGE will also cause a branch on a zero result.

## BLT

branch if less than (zero)

002400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $N \vee V = 1$

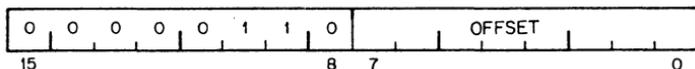
**Condition Codes:** Unaffected

**Description:** Causes a branch if the "Exclusive Or" of the N and V bits are 1. Thus BLT will always branch following an operation that added two negative numbers, even if overflow occurred. In particular, BLT will always cause a branch if it follows a CMP instruction operating on a negative source and a positive destination (even if overflow occurred). Further, BLT will never cause a branch when it follows a CMP instruction operating on a positive source and negative destination. BLT will not cause a branch if the result of the previous operation was zero (without overflow).

## BGT

branch if greater than (zero)

003000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $Z \vee (N \vee V) = 0$

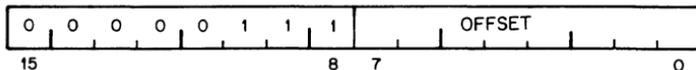
**Condition Codes:** Unaffected

**Description:** Operation of BGT is similar to BGE, except BGT will not cause a branch on a zero result.

## BLE

branch if less than or equal (to zero)

003400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $Z \vee (N \vee V) = 1$

**Condition Codes:** Unaffected

**Description:** Operation is similar to BLT but in addition will cause a branch if the result of the previous operation was zero.

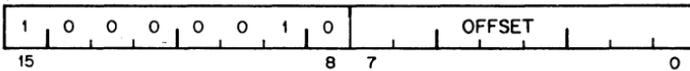
### Unsigned Conditional Branches

The Unsigned Conditional Branches provide a means for testing the result of comparison operations in which the operands are considered as unsigned values.

# BHI

branch if higher

101000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C=0$  and  $Z=0$

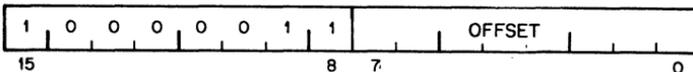
**Condition Codes:** Unaffected

**Description:** Causes a branch if the previous operation caused neither a carry nor a zero result. This will happen in comparison (CMP) operations as long as the source has a higher unsigned value than the destination.

# BLOS

branch if lower or same

101400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C \vee Z = 1$

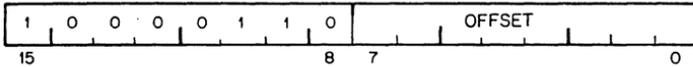
**Condition Codes:** Unaffected

**Description:** Causes a branch if the previous operation caused either a carry or a zero result. BLOS is the complementary operation to BHI. The branch will occur in comparison operations as long as the source is equal to, or has a lower unsigned value than the destination.

# BHIS

branch if higher or same

103000 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C = 0$

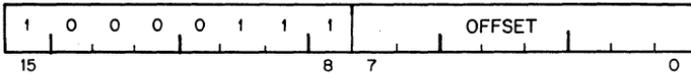
**Condition Codes:** Unaffected

**Description:** BHIS is the same instruction as BCC. This mnemonic is included only for convenience.

# BLO

branch if lower

103400 Plus offset



**Operation:**  $PC \leftarrow PC + (2 \times \text{offset})$  if  $C = 1$

**Condition Codes:** Unaffected

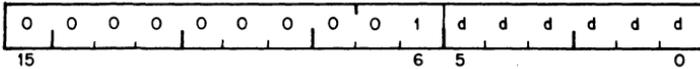
**Description:** BLO is same instruction as BCS. This mnemonic is included only for convenience.

## Jump Instructions

### JMP

jump

0001DD



**Operation:** PC  $\leftarrow$  (dst)

**Condition Codes:** unaffected

**Description:** JMP provides more flexible program branching than provided with the branch instructions. Control may be transferred to any location in memory (no range limitation) and can be accomplished with the full flexibility of the addressing modes, with the exception of register mode 0. Execution of a jump with mode 0 will cause an "illegal instruction" condition, and will cause the CPU to trap to vector address 4. (Program control cannot be transferred to a register.) Register deferred mode is legal and will cause program control to be transferred to the address held in the specified register. Note that instructions are word data and must therefore be fetched from an even-numbered address.

Deferred index mode JMP instructions permit transfer of control to the address contained in a selectable element of a table of dispatch vectors.

**Example:**

JMP FIRST ; Transfers to First

.....  
.....

**First:**

JMP @LIST ; Transfers to location pointed to at LIST

.....

**List:**

FIRST ; pointer to FIRST

JMP @(SP)+ ; Transfer to location pointed to by the top of the stack, and remove the pointer from the stack

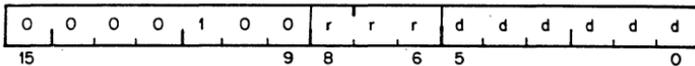
### Subroutine Instructions

The subroutine call in the PDP-11 provides for automatic nesting of subroutines, reentrancy, and multiple entry points. Subroutines may call other subroutines (or indeed themselves) to any level of nesting without making special provision for storage of return addresses at each level of subroutine call. The subroutine calling mechanism does not modify any fixed location in memory, thus providing for reentrancy. This allows one copy of a subroutine to be shared among several interrupting processes.

## JSR

jump to subroutine

004RDD



- Operation:**
- $\uparrow(\text{SP}) \leftarrow \text{reg}$  (push reg contents onto processor stack)
  - $\text{reg} \leftarrow \text{PC}$  (PC holds location following JSR; this address now put in reg)
  - $\text{PC} \leftarrow (\text{dst})$  (PC now points to subroutine destination)

**Description:**

In execution of the JSR, the old contents of the specified register (the "LINKAGE POINTER") are automatically pushed onto the processor stack and new linkage information placed in the register. Thus subroutines nested within subroutines to any depth may all be called with the same linkage register. There is no need either to plan the maximum depth at which any particular subroutine will be called or to include instructions in each routine to save and restore the linkage pointer. Further, since all linkages are saved in a reentrant manner on the processor stack execution of a subroutine may be interrupted, the same subroutine reentered and executed by an interrupt service routine. Execution of the initial subroutine can then be resumed when other requests are satisfied. This process (called nesting) can proceed to any level.

A subroutine called with a JSR reg,dst instruction can access the arguments following the call with either autoincrement addressing, (reg) +, (if arguments are accessed sequentially) or by indexed addressing, X(reg), (if accessed in random order). These addressing modes may also be deferred, @(reg) + and @X(reg) if the parameters are operand addresses rather than the operands themselves.

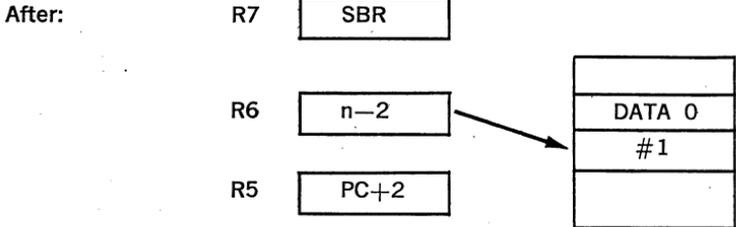
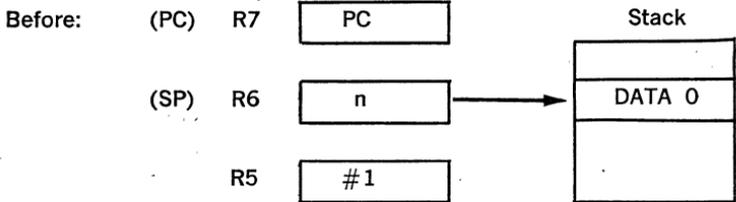
JSR PC, dst is a special case of the PDP-11 subroutine call suitable for subroutine calls that transmit parameters through the general registers. The SP and the PC are the only registers that may be modified by this call.

Another special case of the JSR instruction is JSR PC, @(SP)+ which exchanges the top element of the processor stack and the contents of the program counter. Use of this instruction allows two routines to swap program control and resume operation when recalled where they left off. Such routines are called "co-routines."

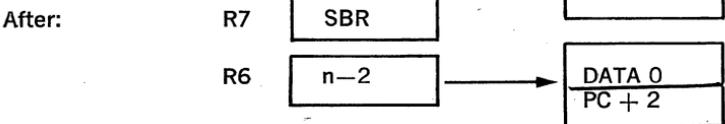
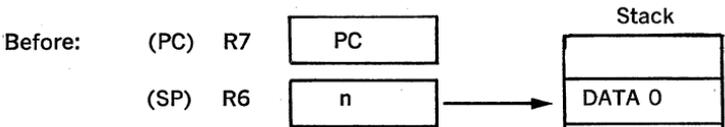
Return from a subroutine is done by the RTS instruction. RTS reg loads the contents of reg into the PC and pops the top element of the processor stack into the specified register.

**Example:**

**JSR R5, SBR**



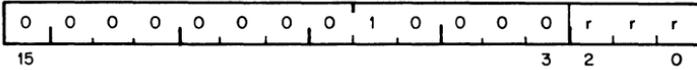
**JSR PC, SBR**



# RTS

return from subroutine

00020R

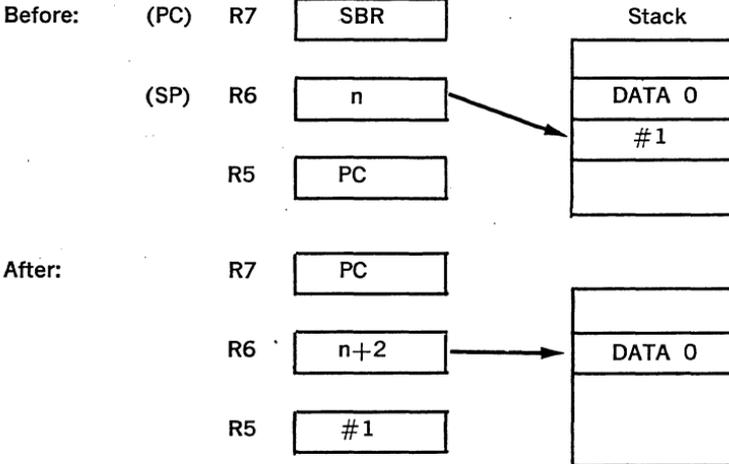


**Operation:** PC ← (reg)  
(reg) ← (SP) ↑

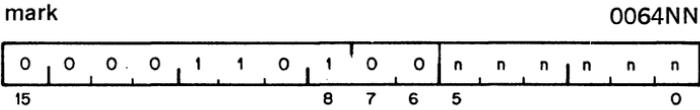
**Description:** Loads contents of register into PC and pops the top element of the processor stack into the specified register.  
Return from a non-reentrant subroutine is typically made through the same register that was used in its call. Thus, a subroutine called with a JSR PC, dst exists with a RTS PC and a subroutine called with a JSR R5, dst, may pick up parameters with addressing modes (R5) +, X(R5), or @X(R5) and finally exists, with an RTS R5.

**Example:**

RTS R5



# MARK



**Operation:** SP ← updated PC + 2 + 2n n = number of parameters  
 PC ← R5  
 R5 ← (SP) ^

**Condition Codes:** unaffected

**Description:** Used as part of the standard PDP-11 subroutine return convention. MARK facilitates the stack clean up procedures involved in subroutine exit. Assembler format is: MARK N

**Example:**

```

MOV R5,-(SP)           ;place old R5 on stack
MOV P1,-(SP)           ;place N parameters
MOV P2,-(SP)           ;on the stack to be
                       ;used there by the
                       ;subroutine

MOV PN,-(SP)
MOV #MARKN,-(SP)      ;places the instruction
                       ;MARK N on the stack
MOV SP,R5              ;set up address at MARK N in-
                       ;struction
JSR PC,SUB             ;jump to subroutine
  
```

At this point the stack is as follows:

|        |
|--------|
| OLD R5 |
| P1     |
| PN     |
| MARK N |
| OLD PC |

And the program is at the address SUB which is the beginning of the subroutine.

SUB: ;execution of the subroutine itself

RTS R5 ;the return begins: this

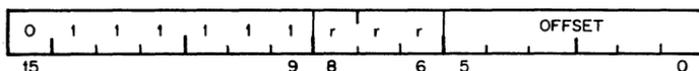
causes the contents of R5 to be placed in the PC which then results in the execution of the instruction MARK N. The contents of old PC are placed in R5

MARK N causes: (1) the stack pointer to be adjusted to point to the old R5 value; (2) the value now in R5 (the old PC) to be placed in the PC; and (3) contents of the old R5 to be popped into R5 thus completing the return from subroutine.

## SOB

subtract one and branch (if  $\neq 0$ )

077RNN



**Operation:**  $(R) \leftarrow (R) - 1$ ; if this result  $\neq 0$  then  $PC \leftarrow PC - (2 \times \text{offset})$  if  $(R) = 0$ ;  $PC \leftarrow PC$

**Condition Codes:** unaffected

**Description:** The register is decremented. If it is not equal to 0, twice the offset is subtracted from the PC (now pointing to the following word). The offset is interpreted as a sixbit positive number. This instruction provides a fast, efficient method of loop control. Assembler syntax is:

SOB R,A

where A is the address to which transfer is to be made if the decremented R is not equal to 0. Note that the SOB instruction can not be used to transfer control in the forward direction.

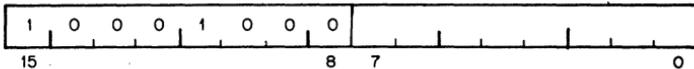
## Traps

Trap instructions provide for calls to emulators, I/O monitors, debugging packages, and user-defined interpreters. A trap is effectively an interrupt generated by software. When a trap occurs the contents of the current Program Counter (PC) and processor Status Word (PS) are pushed onto the processor stack and replaced by the contents of a two-word trap vector containing a new PC and new PS. The return sequence from a trap involves executing an RTI or RTT instruction which restores the old PC and old PS by popping them from the stack. Trap instruction vectors are located at permanently assigned fixed addresses.

## EMT

emulator trap

104000—104377



**Operation:**

- ▼(SP)←PS
- ▼(SP)←PC
- PC←(30)
- PS←(32)

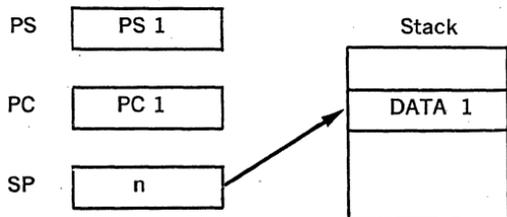
**Condition Codes:**

- N: loaded from trap vector
- Z: loaded from trap vector
- V: loaded from trap vector
- C: loaded from trap vector

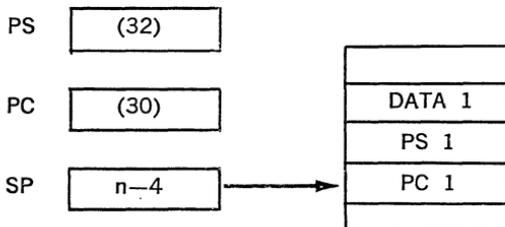
**Description:** All operation codes from 104000 to 104377 are EMT instructions and may be used to transmit information to the emulating routine (e.g., function to be performed). The trap vector for EMT is at address 30. The new PC is taken from the word at address 30; the new processor status (PS) is taken from the word at address 32.

Caution: EMT is used frequently by DEC system software and is therefore not recommended for general use.

Before:



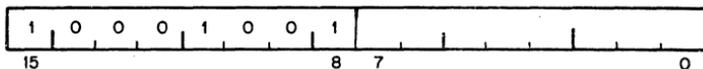
After:



## TRAP

trap

104400—104777



**Operation:**

- ▼(SP)←PS
- ▼(SP)←PC
- PC←(34)
- PS←(36)

**Condition Codes:**

- N: loaded from trap vector
- Z: loaded from trap vector
- V: loaded from trap vector
- C: loaded from trap vector

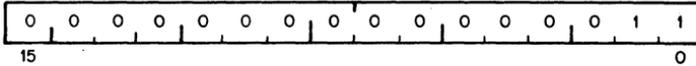
**Description:** Operation codes from 104400 to 104777 are TRAP instructions. TRAPs and EMTs are identical in operation, except that the trap vector for TRAP is at address 34.

**Note:** Since DEC software makes frequent use of EMT, the TRAP instruction is recommended for general use.

# BPT

breakpoint trap

000003



**Operation:**      $\downarrow(\text{SP}) \leftarrow \text{PS}$   
                   $\downarrow(\text{SP}) \leftarrow \text{PC}$   
                   $\text{PC} \leftarrow (14)$   
                   $\text{PS} \leftarrow (16)$

**Condition Codes:** N: loaded from trap vector  
                      Z: loaded from trap vector  
                      V: loaded from trap vector  
                      C: loaded from trap vector

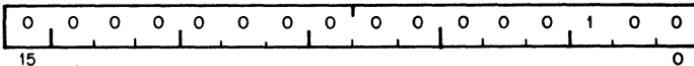
**Description:**     Performs a trap sequence with a trap vector address of 14.  
                      Used to call debugging aids. The user is cautioned against  
                      employing code 000003 in programs run under these de-  
                      bugging aids.

(No information is transmitted in the low byte.)

# IOT

input/output trap

000004



**Operation:**      $\downarrow(\text{SP}) \leftarrow \text{PS}$   
                   $\downarrow(\text{SP}) \leftarrow \text{PC}$   
                   $\text{PC} \leftarrow (20)$   
                   $\text{PS} \leftarrow (22)$

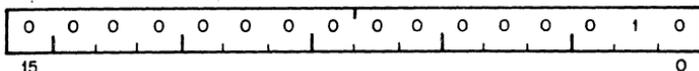
**Condition Codes:** N:loaded from trap vector  
                      Z:loaded from trap vector  
                      V:loaded from trap vector  
                      C:loaded from trap vector

**Description:**     Performs a trap sequence with a trap vector address of 20.  
                      (No information is transmitted in the low byte.)

## RTI

return from interrupt

000002



**Operation:** PC $\leftarrow$ (SP) $\wedge$   
PS $\leftarrow$ (SP) $\wedge$

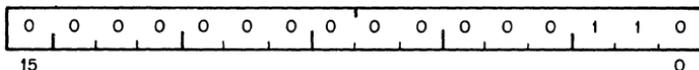
**Condition Codes:** N: loaded from processor stack  
Z: loaded from processor stack  
V: loaded from processor stack  
C: loaded from processor stack

**Description:** Used to exit from an interrupt or TRAP service routine. The PC and PS are restored (popped) from the processor stack. If a trace trap is pending, the first instruction after RTI will not be executed prior to the next T traps.

## RTT

return from interrupt

000006



**Operation:** PC $\leftarrow$ (SP)  $\wedge$   
PS $\leftarrow$ (SP)  $\wedge$

**Condition Codes:** N: loaded from processor stack  
Z: loaded from processor stack  
V: loaded from processor stack  
C: loaded from processor stack

**Description:** Operation is the same as RTI except that it inhibits a trace trap while RTI permits trace trap. If new PS has T bit set, trap will occur after execution of first instruction after RTT.

**Reserved Instruction Traps**—These are caused by attempts to execute instruction codes reserved for future processor expansion (reserved instructions) or instructions with illegal addressing modes (illegal instructions). Order codes not corresponding to any of the instructions described are considered to be reserved instructions. JMP and JSR with register mode destinations are illegal instructions, and trap to vector address 4. Reserved instructions trap to vector address 10.

**Bus Error Traps**—Bus Error Traps are time-out errors; attempts to reference addresses on the bus that have made no response within a certain length of time. In general, these are caused by attempts to reference non-existent memory, and attempts to reference non-existent peripheral devices. Bus error traps cause processor traps through the trap vector address 4.

**Trace Trap**—Trace Trap is enabled by bit 4 of the PSW and causes processor traps at the end of instruction execution. The instruction that is executed after the instruction that set the T-bit will proceed to completion and then trap through the trap vector at address 14. Note that the trace trap is a system debugging aid and is transparent to the general programmer.

#### NOTE

Bit 4 of the PSW can only be set indirectly by executing a RTI or RTT instruction with the desired PSW on the stack.

The following are special cases of the T-bit and are detailed in subsequent paragraphs.

1. The traced instruction cleared the T-bit.
2. The traced instruction set the T-bit.
3. The traced instruction caused an instruction trap.
4. The traced instruction caused a bus error trap.
5. The processor was interrupted between the time the T-bit was set and the fetching of the instruction that was to be traced.
6. The traced instruction was a WAIT.
7. The traced instruction was a HALT.
8. The traced instruction was a Return from Interrupt.

#### NOTE

The traced instruction is the instruction after the one that set the T-bit.

**An instruction that cleared the T bit**—Upon fetching the traced instruction, an internal flag, the trace flag, was set. The trap will still occur at the end of execution of this instruction. The status word on the stack, however, will have a clear T-bit.

**An instruction that set the T-bit**—Since the T-bit was already set, setting it again has no effect. The trap will occur.

**An instruction that caused an Instruction Trap**—The instruction trap is performed and the entire routine for the service trap is executed. If the service routine exists with an RTI or in any other way restores the stacked status word, the T-bit is set again, the instruction following the traced instruction is executed and, unless it is one of the special cases noted previously, a trace trap occurs.

**An instruction that caused a Bus Error Trap**—This is treated as an Instruction Trap. The only difference is that the error service is not as likely to exit with an RTI, so that the trace trap may not occur.

Note that interrupts may be acknowledged immediately after the loading of the new PC and PS at the trap vector location. To lock out all interrupts, the PSW at the trap vector should set Bit 7.

**A WAIT**—T bit trap is not honored during a wait.

**A HALT**—The processor halts. The PC points to the next instruction to be executed. The trap will occur immediately following execution resumption.

**A Return from Interrupt**—The return from interrupt instruction either clears or sets the T-bit. If the T-bit was set and RTT is the traced instruction, the trap is delayed until completion of the next instruction.

**Power Failure Trap**—Occurs when AC power fail signal is received while processor is in run mode. Trap vector for power failure is location 24 and 26. Trap will occur if an RTI instruction is executed in power fail service routine.

**Trap Priorities**—In case of internal and external multiple processor trap conditions, occurring simultaneously, the following order of priorities is observed (from high to low):

- Bus Error Trap
- Memory Refresh
- Instruction Traps
- Trace Trap
- Power Fail Trap
- Halt Line
- Event Line Interrupt
- Device (Bus) Interrupt Request

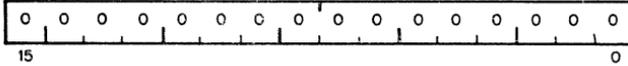
If a bus error is caused by the trap process handling instruction traps, trace traps, or a previous bus error, the processor is halted. This is called a double bus error.

### 3.8 MISCELLANEOUS

## HALT

halt

000000



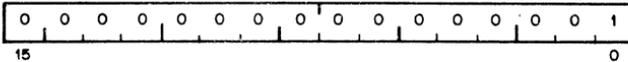
**Condition Codes:** not affected

**Description:** Causes the processor to leave RUN mode. The PC points to the next instruction to be executed. The processor goes into the HALT mode. The contents of the PC are displayed on the console terminal and the console mode of operation is enabled.

## WAIT

wait for interrupt

000001



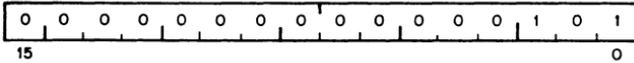
**Condition Codes:** not affected

**Description:** Provides a way for the processor to relinquish use of the bus while it waits for an external interrupt request. Having been given a WAIT command, the processor will not compete for bus use by fetching instructions or operands from memory. This permits higher transfer rates between a device and memory, since no processor-induced latencies will be encountered by interrupt requests from devices. In WAIT, as in all instructions, the PC points to the next instruction following the WAIT instruction. Thus when an interrupt causes the PC and PS to be pushed onto the processor stack, the address of the next instruction following the WAIT is saved. The exit from the interrupt routine (i.e. execution of an RTI instruction) will cause resumption of the interrupted process at the instruction following the WAIT.

# RESET

reset external bus

000005



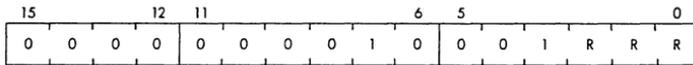
**Condition Codes:** not affected

**Description:** Sends INIT on the BUS for 10  $\mu$ sec. All devices on the BUS are reset to their state at power-up. The processor remains in an idle state for 90  $\mu$ sec following issuance of INIT.

## 3.9 RESERVED INSTRUCTIONS

(NO ASSIGNED MNEMONIC)

00021R



**Operation:** (R)  $\leftarrow$  gets contents of 5 internal 16 bit registers R  $\leftarrow$  R + 12 at end of inst.

**Condition Codes:** Unaffected

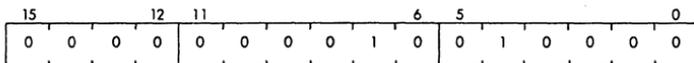
**Description:** Contents of register R (low order 3 bits of inst.) is used as a pointer. The contents of the internal hidden temporary registers are consecutively written into memory and the contents of R are incremented by 2 until the five 16 bit registers have been written. (R)  $\leftarrow$  (R) + 12. Primarily used as a maintenance aid in diagnostic routines. The interpretation of the five words in memory is as follows:

| Memory Location | Microlevel Register Symbol | Function  |
|-----------------|----------------------------|---|
| (R)             | RBA                        | Bus Address Register. It contains the last non-instruction fetch bus address for destination modes, 3, 5, 6, and 7.                                     |
| (R)+2           | RSRC                       | Source Operand Register. It contains the last source operand of a double operand instruction. The high byte may not be correct if it was source mode 0. |

|        |      |   |
|--------|------|---|
| (R)+4  | RDST | Destination Operand Register. It contains the last destination operand fetched by the processor.  |
| (R)+6  | RPSW | PSW and Scratch Register. The top 4 bits are PSW bits 4 through 7. The remaining bit interpretation is a function of the last instruction and may not be that useful for all cases.   |
| (R)+10 | RIR  | Instruction Register. It contains the present, not past, instruction being executed, and will always be 36R where R is the register in the format. The 360 is a result of firmware instruction decoding and is caused by 150 being added to the opcode ( $21R+150=36R$ ). |

(NO ASSIGNED MNEMONIC)

00022N



**Operation:** Causes Micro Instruction Control Transfer to Microlocation 3000

**Condition Codes:** Unaffected

**Description:** This instruction can be used to transfer Microcontrol to Microcode address 3000 in the Microprocessor. If Microaddress 3000 does not exist this opcode will cause a reserved instruction trap through memory location 10.

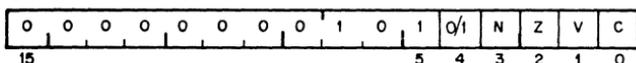
This is a reserved DEC instruction.

### 3.10 CONDITION CODE OPERATORS

|            |            |
|------------|------------|
| <b>CLN</b> | <b>SEN</b> |
| <b>CLZ</b> | <b>SEZ</b> |
| <b>CLV</b> | <b>SEV</b> |
| <b>CLC</b> | <b>SEC</b> |
| <b>CCC</b> | <b>SCC</b> |

condition code operators

0002XX



**Description:**

Set and clear condition code bits. Selectable combinations of these bits may be cleared or set together. Condition code bits corresponding to bits in the condition code operator (Bits 0-3) are modified according to the sense of bit 4, the set/clear bit of the operator. i.e. set the bit specified by bit 0, 1, 2 or 3, if bit 4 is a 1. Clear corresponding bits if bit 4 = 0.

Mnemonic  
Operation

OP Code

|     |                |        |
|-----|----------------|--------|
| CLC | Clear C        | 000241 |
| CLV | Clear V        | 000242 |
| CLZ | Clear Z        | 000244 |
| CLN | Clear N        | 000250 |
| SEC | Set C          | 000261 |
| SEV | Set V          | 000262 |
| SEZ | Set Z          | 000264 |
| SEN | Set N          | 000270 |
| SCC | Set all CC's   | 000277 |
| CCC | Clear all CC's | 000257 |
|     | Clear V and C  | 000243 |
| NOP | No Operation   | 000240 |

Combinations of the above set or clear operations may be ORed together to form combined instructions.



## EXTENDED ARITHMETIC OPTION

## 4.1 GENERAL

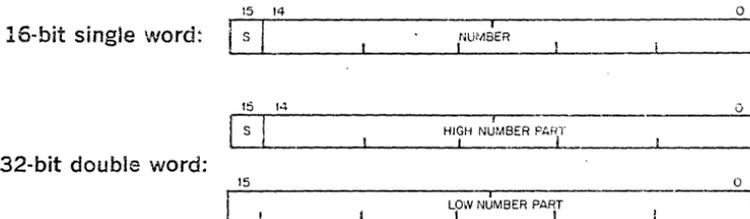
This chapter describes the Extended Arithmetic Chip, which is an option on the KD11-F, KD11-J Microcomputer Module. The KEV11 option allows extended manipulation of fixed point numbers (fixed point arithmetic) and enables direct operations on single precision 32-bit words (floating point arithmetic).

## 4.2 FIXED POINT ARITHMETIC (EIS)

The following instructions apply to fixed point numbers:

| Mnemonic | Instruction               | Op Code |
|----------|---------------------------|---------|
| MUL      | multiply                  | 070RSS  |
| DIV      | divide                    | 071RSS  |
| ASH      | shift arithmetically      | 072RSS  |
| ASCH     | arithmetic shift combined | 073RSS  |

Operand formats are:



S is the sign bit. S = 0 for positive quantities  
S = 1 for negative quantities; number is in 2's complement notation

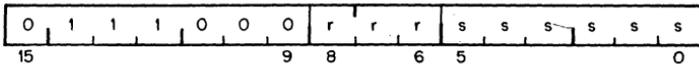
## NOTE

When executing an EIS instruction, the LSI-11 processor fetches the source operand via the DATIO bus cycle, rather than the DATI bus cycle. If the source operand is contained in a PROM or ROM location, a bus error (timeout) will occur because the processor will attempt to write into the addressed location after fetching the operand. When using EIS instructions in programs that will be stored in PROM or ROM, refer to Section I, Chapter 7, paragraph 7.3.

# MUL

multiply

070RSS



**Operation:** R, Rv1 ← R x(src)

**Condition Codes:** N: set if product is <0; cleared otherwise  
Z: set if product is 0; cleared otherwise  
V: cleared  
C: set if the result is less than  $-2^{15}$  or greater than or equal to  $2^{15}-1$ .

**Description:** The contents of the destination register and source taken as two's complement integers are multiplied and stored in the destination register and the succeeding register (if R is even). If R is odd only the low order product is stored. Assembler syntax is : MUL S,R.  
(Note that the actual destination is R,Rv1 which reduces to just R when R is odd.)

**Example:** 16-bit product (R is odd)

```
CLC ;Clear carry condition code
MOV #400,R1
MUL #10,R1
BCS ERROR ;Carry will be set if
           ;product is less than
           ; $-2^{15}$  or greater than or equal to  $2^{15}$ 
           ;no significance lost
```

Before After

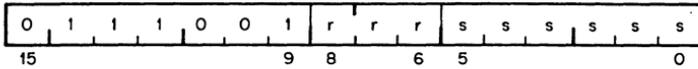
(R1) = 000400 (R1) = 004000

Assembler format for all EIS instructions is:  
OPR src, R

# DIV

divide

071RSS



**Operation:** R, Rv1  $\leftarrow$  R, Rv1 / (src)

**Condition Codes:** N: set if quotient  $< 0$ ; cleared otherwise  
Z: set if quotient = 0; cleared otherwise  
V: set if source = 0 or if the absolute value of the register is larger than the absolute value of the source. (In this case the instruction is aborted because the quotient would exceed 15 bits.)  
C: set if divide 0 attempted; cleared otherwise

**Description:** The 32-bit two's complement integer in R and Rv1 is divided by the source operand. The quotient is left in R; the remainder in Rv1. Division will be performed so that the remainder is of the same sign as the dividend. R must be even.

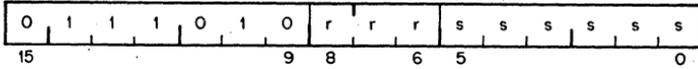
**Example:**  
CLR R0  
MOV #20001,R1  
DIV #2,R0

| Before        | After         |           |
|---------------|---------------|-----------|
| (R0) = 000000 | (R0) = 010000 | Quotient  |
| (R1) = 020001 | (R1) = 000001 | Remainder |

# ASH

shift arithmetically

072RSS



**Operation:**

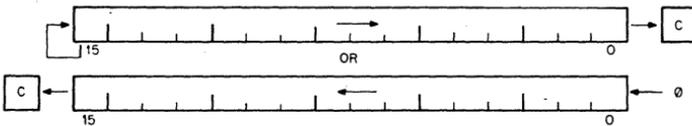
$R \leftarrow R$  Shifted arithmetically NN places to right or left  
Where NN = low order 6 bits of source

**Condition Codes:**

N: set if result < 0; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: set if sign of register changed during shift; cleared otherwise  
C: loaded from last bit shifted out of register

**Description:**

The contents of the register are shifted right or left the number of times specified by the shift count. The shift count is taken as the low order 6 bits of the source operand. This number ranges from -32 to +31. Negative is a right shift and positive is a left shift.



**6 LSB of source**

011111  
000001  
111111  
100000

**Action in general register**

Shift left 31 places  
shift left 1 place  
shift right 1 place  
shift right 32 places

**Example:**

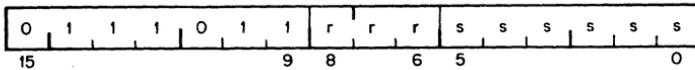
ASH R0, R3

Before  
(R0) = 001234  
(R3) = 000003

After  
(R0) = 012340  
(R3) = 000003

arithmetic shift combined

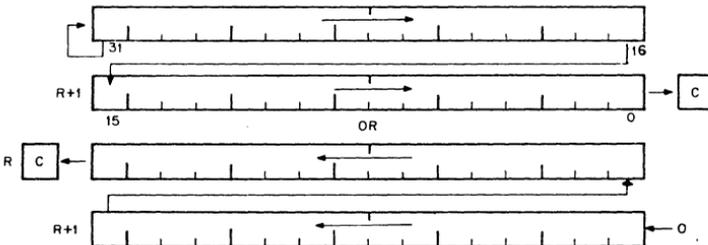
073RSS



**Operation:**  $R, Rv1 \leftarrow R, Rv1$  The double word is shifted NN places to the right or left, where NN = low order six bits of source

**Condition Codes:**  
 N: set if result  $< 0$ ; cleared otherwise  
 Z: set if result  $= 0$ ; cleared otherwise  
 V: set if sign bit changes during the shift; cleared otherwise  
 C: loaded with high order bit when left Shift; loaded with low order bit when right shift (loaded with the last bit shifted out of the 32-bit operand)

**Description:** The contents of the register and the register ORed with one are treated as one 32 bit word, R + 1 (bits 0-15) and R (bits 16-31) are shifted right or left the number of times specified by the shift count. The shift count is taken as the low order 6 bits of the source operand. This number ranges from -32 to +31. Negative is a right shift and positive is a left shift. When the register chosen is an odd number the register and the register OR'ed with one are the same. In this case the right shift becomes a rotate (for up to a shift of 16). The 16 bit word is rotated right the number of bits specified by the shift count.

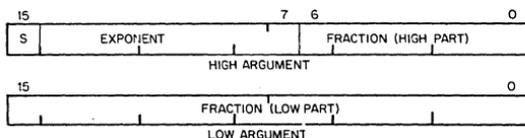


### 4.3 FLOATING POINT ARITHMETIC (FIS)

The Floating Point instructions used are unique to the LSI-11 and PDP-11/35 & 40. However, the OP Codes used do not conflict with any other instructions.

| Mnemonic | Instruction       | Op Code |
|----------|-------------------|---------|
| FADD     | floating add      | 07500R  |
| FSUB     | floating subtract | 07501R  |
| FMUL     | floating multiply | 07502R  |
| FDIV     | floating divide   | 07503R  |

The operand format is:



S = sign of fraction; 0 for positive, 1 for negative

Exponent = 8 bits for the exponent, in excess (200)<sub>8</sub> notation

Fraction = 23 bits plus 1 hidden bit (all numbers are assumed to be normalized)

The number format is essentially a sign and magnitude representation. The format is identical with the 11/45 for single precision numbers.

#### Fraction

The binary radix point is to the left (in front of bit 6 of the High Argument), so that the value of the fraction is always less than 1 in magnitude. Normalization would always cause the first bit after the radix point to be a 1, such that the fractional value would be between  $\frac{1}{2}$  and 1. Therefore, this bit can be understood and not be represented directly, to achieve an extra 1 bit of resolution.

The first bit to the right of the radix point (hidden bit) is always a 1: The next bit for the fraction is taken from bit 6 of the High Argument. The result of a Floating Point operation is always rounded away from zero, increasing the absolute value of the number.

#### Exponent

The 8-bit exponent field (bits 14 to 7) allow exponent values between -128 and +127. Since an excess (200)<sub>8</sub> or (128)<sub>10</sub> number system is used, the correspondence between actual values and coded representation is as follows:

| Actual Value | Representation |                     |
|--------------|----------------|---------------------|
|              | Decimal        | Octal      Binary   |
| +127         |                | 377      11 111 111 |
| +1           |                | 201      10 000 001 |
| 0            |                | 200      10 000 000 |
| -1           |                | 177      01 111 111 |
| -128         |                | 000      00 000 000 |



Traps occur through the vector at location 244. A Floating Point instruction will be aborted if an interrupt request is issued before the instruction is near completion. The Program Counter will point to the aborted Floating Point instruction so that the Interrupt will look transparent.

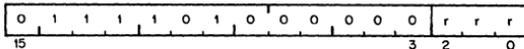
Assembler format is: OPR R

## INSTRUCTIONS

### FADD

floating add

07500R



**Operation:**  $[(R)+4, (R)+6] \leftarrow [(R)+4, (R)+6] + [(R), (R)+2]$ , if result  $\geq 2^{-128}$ ; else  $[(R)+4, (R)+6] \leftarrow 0$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
 Z: set if result = 0; cleared otherwise  
 V: cleared  
 C: cleared

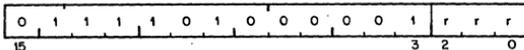
**Description:** Adds the A argument to the B argument and stores the result in the A Argument position on the stack. General register R is used as the stack pointer for the operation.

$A \leftarrow A + B$

### FSUB

floating subtract

07501R



**Operation:**  $[(R)+4, (R)+6] \leftarrow [(R)+4, (R)+6] - [(R), (R)+2]$ , if result  $\geq 2^{-128}$ ; else  $[(R)+4, (R)+6] \leftarrow 0$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
 Z: set if result = 0; cleared otherwise  
 V: cleared  
 C: cleared

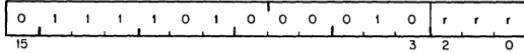
**Description:** Subtracts the B Argument from the A Argument and stores the result in the A Argument position on the stack.

$A \leftarrow A - B$

# FMUL

floating multiply

07502R



**Operation:**  $[(R)+4, (R)+6] \leftarrow [(R)+4, (R)+6] \times [(R), (R)+2]$  if result  $\geq 2^{-128}$ ; else  $[(R)+4, (R)+6] \leftarrow 0$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: cleared  
C: cleared

**Description:** Multiplies the A Argument by the B Argument and stores the result in the A Argument position on the stack.

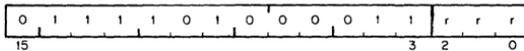
$A \leftarrow A \times B$

(refer to note below)

# FDIV

floating divide

07503R



**Operation:**  $[(R)+4, (R)+6] \leftarrow [(R)+4, (R)+6] / [(R), (R)+2]$  if result  $\geq 2^{-128}$ ; else  $[(R)+4, (R)+6] \leftarrow 0$

**Condition Codes:** N: set if result  $< 0$ ; cleared otherwise  
Z: set if result = 0; cleared otherwise  
V: cleared  
C: cleared

**Description:** Divides the A Argument by the B Argument and stores the result in the A Argument position on the stack. If the divisor (B Argument) is equal to zero, the stack is left untouched.

$A \leftarrow A/B$

## NOTE

Unlike the PDP-11/40 (and PDP-11/35), the LSI-11 processor pushes one word onto the stack during execution of FMUL and FDIV instructions and pops the word from the stack when completed. Thus, the SP (R6) must point to a read/write memory location; otherwise, a bus error (timeout) will occur.



## PROGRAMMING TECHNIQUES

In order to produce programs which fully utilize the power and flexibility of the LSI-11, the reader should become familiar with the various programming techniques which are part of the basic design philosophy of the LSI-11. Although it is possible to program the LSI-11 along traditional lines such as "accumulator orientation" this approach does not fully exploit the architecture and instruction set of the LSI-11.

### 5.1 THE STACK

A "stack," as used on the LSI-11, is an area of memory set aside by the programmer for temporary storage or subroutine/interrupt service linkage. The instructions which facilitate "stack" handling are useful features not normally found in low-cost computers. They allow a program to dynamically establish, modify, or delete a stack and items on it. The stack uses the "last-in, first-out" concept, that is, various items may be added to a stack in sequential order and retrieved or deleted from the stack in reverse order. On the LSI-11, a stack starts at the highest location reserved for it and expands linearly downward to the lowest address as items are added to the stack.

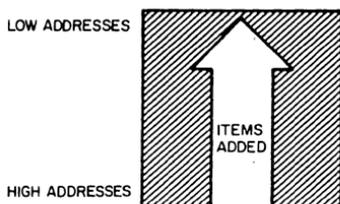


Figure 5-1 Stack Addresses

The programmer does not need to keep track of the actual locations his data is being stacked into. This is done automatically through a "stack pointer." To keep track of the last item added to the stack (or "where we are" in the stack) a General Register always contains the memory address where the last item is stored in the stack. In the LSI-11 any register except Register 7 (the Program Counter-PC) may be used as a "stack pointer" under program control; however, instructions associated with subroutine linkage and interrupt service automatically use Register 6 (R6) as a hardware "Stack Pointer." For this reason R6 is frequently referred to as the system "SP."

Stacks in the LSI-11 may be maintained in either full word or byte units. This is true for a stack pointed to by any register except R6, which must be organized in full word units only.

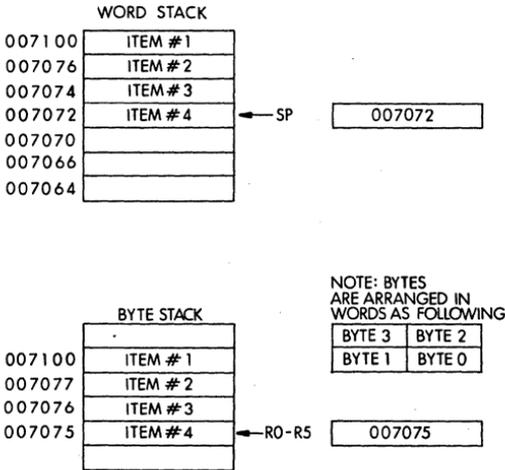


Figure 5-2 Word and Byte Stacks

Items are added to a stack using the autodecrement addressing mode with the appropriate pointer register. (See Chapter 2 for description of the autoincrement/decrement modes).

This operation is accomplished as follows;

MOV Source,—(SP) ;MOV Source Word onto the stack  
or

MOVB Source,—(R) ;MOVB Source Byte onto a stack

This is called a “push” because data is “pushed onto the stack.”

To remove an item from a stack the autoincrement addressing mode with the appropriate R is employed. This is accomplished in the following manner:

MOV (SP), +, Destination ;MOV Destination Word off the stack

or

MOVB (R) +, Destination ;MOVB Destination Byte off the stack

Removing an item from a stack is called a "pop" for "popping from the stack." After an item has been "popped," its stack location is considered free and available for other use. The stack pointer points to the last-used location implying that the next (lower) location is free. Thus a stack may represent a pool of share-able temporary storage locations.

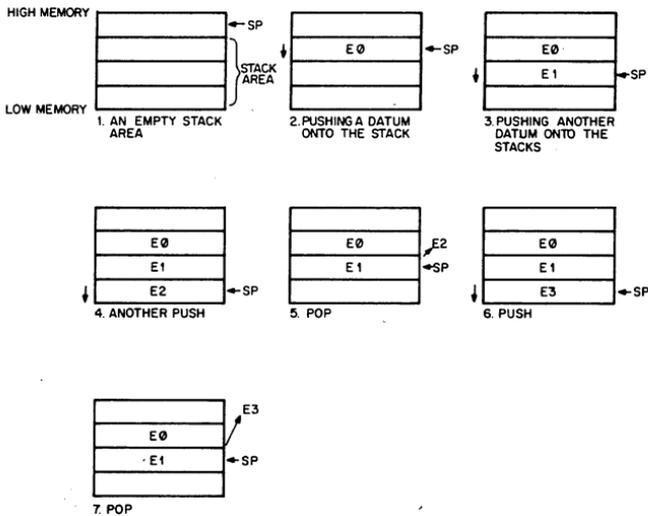


Figure 5-3 Illustration of Push and Pop Operations

As an example of stack usage consider this situation: a subroutine (SUBR) wants to use registers 1 and 2, but these registers must be returned to the calling program with their contents unchanged. The subroutine could be written as follows:

| Address | Octal Code | Assembler Syntax            |
|---------|------------|-----------------------------|
| 076322  | 010167     | SUBR: MOV R1,TEMP1 ;save R1 |
| 076324  | 000074     | *                           |
| 076326  | 010267     | MOV R2,TEMP2 ;save R2       |
| 076330  | 000072     | *                           |
| .       | .          | .                           |
| .       | .          | .                           |
| 076410  | 016701     | MOV TEMP1, R1 ;Restore R1   |
| 076412  | 000006     | *                           |
| 076414  | 016702     | MOV TEMP2, R2 ;Restore R2   |
| 076416  | 000004     | *                           |
| 076420  | 000207     | RTS PC                      |
| 076422  | 000000     | TEMP1: 0                    |
| 076424  | 000000     | TEMP2: 0                    |

\*Index Constants

Figure 5-4 Register Saving Without the Stack

OR: Using a Stack

| Address | Octal Code | Assembler Syntax             |
|---------|------------|------------------------------|
| 010020  | 010143     | SUBR: MOV R1, -(R3) ;push R1 |
| 010022  | 010243     | MOV R2, -(R3) ;push R2       |
| .       | .          | .                            |
| .       | .          | .                            |
| 010130  | 012301     | MOV (R3) +, R2 ;pop R2       |
| 010132  | 012302     | MOV (R3) +, R1 ;pop R1       |
| 010134  | 000207     | RTS PC                       |

Note: In this case R3 was used as a Stack Pointer:

Figure 5-5 Register Saving Using the Stack

The second routine uses four less words of instruction code and two words of temporary "stack" storage. Another routine could use the same stack space at some later point. Thus, the ability to share temporary storage in the form of a stack is a very economical way to save on memory usage.



When a JSR is executed, the contents of the linkage register are saved on the system R6 stack as if a MOV reg.-(SP) had been performed. Then the same register is loaded with the memory address following the JSR instruction (the contents of the current PC) and a jump is made to the entry location specified by the DST operand.

| Address | Assembler Syntax        | Octal Code |
|---------|-------------------------|------------|
| 001000  | JSRR5, SUBR             | 004567     |
| 001002  | index constant for SUBR | 000060     |
| 001064  | SUBR: MOV A,B           | 0lnnmm     |

Figure 5-7 JSR Using R5

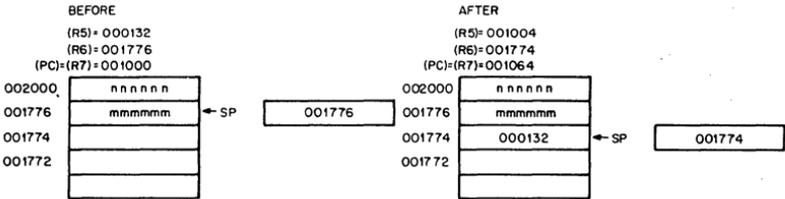


Figure 5-8 JSR

Note that the instruction JSR R6, SUBR is not normally considered to be a meaningful combination.

### 5.2.2 Argument Transmission

The memory location pointed to by the linkage register of the JSR instruction may contain arguments or addresses of arguments. These arguments may be accessed from the subroutine in several ways. Using Register 5 as the linkage register, the first argument could be obtained by using the addressing modes indicated by (R5), (R5) + ,X(R5) for actual data, or @(R5) + , etc. for the address of data. If the autoincrement mode is used, the linkage register is automatically updated to point to the next argument.

Figures 5-9 and 5-10 illustrate two possible methods of argument transmission.

#### Address Instructions and Data

|        |                         |  |  |
|--------|-------------------------|--|--|
| 010400 | JSR R5, SUBR            |  |  |
| 010402 | Index constant for SUBR |  | SUBROUTINE CALL                          |
| 010404 | arg # 1                 |  | ARGUMENTS                                |
| 010406 | arg # 2                 |  |  |
| .      | .                       |  |  |
| .      | .                       |  |  |
| .      | .                       |  |  |
| .      | .                       |  |  |
| 020306 | SUBR: MOV (R5) + ,R1    |  | ;get arg # 1                             |
| 020310 | MOV (R5) + ,R2          |  | ;get arg # 2 Retrieve Arguments from SUB |

Figure 5-9 Argument Transmission -Register Autoincrement Mode

| Address | Instructions and Data   |                     |
|---------|-------------------------|---------------------|
| 010400  | JSR R5,SUBR             |                     |
| 010402  | index constant for SUBR | SUBROUTINE CALL     |
| 010404  | 077722                  | Address of Arg # 1  |
| 010406  | 077724                  | Address of Arg. # 2 |
| 010410  | 077726                  | Address of Arg. # 3 |
| .       | .                       | .                   |
| .       | .                       | .                   |
| 077722  | Arg # 1                 |                     |
| 077724  | arg # 2                 | arguments           |
| 077726  | arg # 3                 |                     |
| .       | .                       | .                   |
| .       | .                       | .                   |
| 020306  | SUBR: MOV @(R5) + ,R1   | ;get arg # 1        |
| 020301  | MOV @(R5) + ,R2         | ;get arg # 2        |

Figure 5-10 Argument Transmission-Register Autoincrement Deferred Mode

Another method of transmitting arguments is to transmit only the address of the first item by placing this address in a general-purpose register. It is not necessary to have the actual argument list in the same general area as the subroutine call. Thus a subroutine can be called to work on data located anywhere in memory. In fact, in many cases, the operations performed by the subroutine can be applied directly to the data located on or pointed to by a pointer without the need to ever actually move this data into the subroutine area.

```

Calling Program: MOV    POINTER, R1
                  JSR    PC,SUBR

SUBROUTINE  ADD    (R1) + ,(R1) ;Add item # 1 to item # 2, place
                               result in item # 2, R1 points
                               to item # 2 now

                               etc.
                               or

                  ADD    (R1),2(R1) ;Same effect as above except that
                               R1 still points to item # 1
                               etc.

```

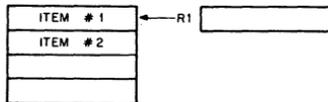


Figure 5-11 Transmitting Stacks as Arguments

Because the LSI-11 hardware already uses general purpose register R6 to point to a stack for saving and restoring PC and PS (processor status word) information, it is quite convenient to use this same stack to save and restore intermediate results and to transmit arguments to and from subroutines. Using R6 in this manner permits extreme flexibility in nesting subroutines and interrupt service routines.

Since arguments may be obtained from the stack by using some form of register indexed addressing, it is sometimes useful to save a temporary copy of R6 in some other register which has already been saved at the beginning of a subroutine. In the previous example R5 may be used to index the arguments while R6 is free to be incremented and decremented in the course of being used as a stack pointer. If R6 had been used directly as the base for indexing and not "copied," it might be difficult to keep track of the position in the argument list since the base of the stack would change with every autoincrement/decrement which occurs.



Figure 5-12 Shifting Indexed Base

However, if the contents of R6 (SP) are saved in R5 before any arguments are pushed onto the stack, the position relative to R5 would remain constant.



Figure 5-13 Constant Index Base Using "R6 Copy"

### 5.2.3 Subroutine Return

In order to provide for a return from a subroutine to the calling program an RTS instruction is executed by the subroutine. This instruction should specify the same register as the JSR used in the subroutine call. When executed, it causes the register, specified, to be moved to the PC and the top of the stack to be then placed in the register specified. Note that if an RTS PC is executed, it has the effect of returning to the address specified by the contents of the top of the stack.

Note that the JSR and the JMP instructions differ in that a linkage register is always used with a JSR; there is no linkage register with a JMP and no way to return to the calling program.

When a subroutine finishes, it is necessary to "clean-up" the stack by eliminating or skipping over the subroutine arguments. One way this can be done is by making the subroutine keep the number of arguments as its first stack item. Returns from subroutines would then involve calculating the amount by which to reset the stack pointer. Resetting the stack pointer then restores the original contents of the register which was used as the copy of the stack pointer. The LSI-11 however, has a specific instruction (MARK instruction) used to perform the clean-up task. The MARK instruction which is stored on a stack in place of "number of argument" information may be used to automatically perform these "clean-up" chores.

### 5.2.4 LSI-11 Set Subroutine Advantages

There are several advantages to the LSI-11 Set subroutine calling procedure.

- a. arguments can be quickly passed between the calling program and the subroutine.
- b. if the user has no arguments or the arguments are in a general register or on the stack, the JSR PC, DST mode can be used so that none of the general-purpose registers are taken up for linkage.
- c. many JSRs can be executed without the need to provide any saving procedure for the linkage information since all linkage information is automatically pushed onto the stack in sequential order. Returns can simply be made by automatically popping this information from the stack in the opposite order of the JSRs.

Such linkage address bookkeeping is called automatic "nesting" of subroutine calls. This feature enables the programmer to construct fast, efficient linkages in a simple, flexible manner. It even permits a routine to call itself in those cases where this is meaningful. It also allows subroutines to be interrupted by external devices without losing the proper return linkage registers.

### 5.2.5 Trap Subroutine Calls

The TRAP instruction may be used to call subroutines. The TRAP instruction is typically used with a package of many different subroutines such as the software floating-point math package. The subroutines in the package are assigned a unique number which is to be included in the TRAP instruction. When a subroutine is called, a "TRAP n" instruction is executed, where "n" is the number ( $0 \leq n \leq 255$ ) which designates

the subroutine to be invoked. Arguments are typically passed on the stack, in the registers, or they may follow the TRAP instruction. The advantages of using the TRAP instruction are that a program using a TRAP subroutine package may be assembled and linked independent of the TRAP package and the subroutine call only requires one word, as opposed to two words which are normally required using the JSR instruction. The disadvantage of using the TRAP instruction is the extra overhead incurred in the software decoding of the TRAP instruction.

```

Calling      MOV ARG, -(SP)      ; Push argument onto the
Program:     TRAP 3             ; Invoke subroutine #3

Trap        TRAPH: MOV R0, -(SP) ; Save register
handler:     MOV 2(SP), R0      ; Copy address of the
                                     TRAP instruction +2
             MOVB -2(R0), R0    ; Copy subroutine number
                                     in TRAP instruction
             BIC #177400, R0    ; Clear possible sign
                                     extension bits
             ASL R0             ; Convert to word offset
             JSR PC,@TRPTBL(R0) ; Call subroutine
             MOV (SP)+, R0      ; Restore register
             RTT                ; Return to user

             TRPTBL: SUB0       ; Table of pointer to
             SUB1               subroutines
             SUB2
             SUB3
             . . .

```

## 5.3 INTERRUPTS

### 5.3.1 General Principles

Interrupts are in many respects very similar to subroutine calls. However, they are forced, rather than controlled, transfers of program execution occurring because of some external and program-independent event (such as a stroke on the teleprinter keyboard). Like subroutines, interrupts have linkage information such that a return to the interrupted program can be made. More information is actually necessary for an interrupt than a subroutine because of the random nature of interrupts. The complete machine state of the program immediately prior to the occurrence of the interrupt must be preserved in order to return to the program without any noticeable effects (i.e. was the previous operation zero or negative, etc.). This information is stored in the Processor Status Word (PS). Upon interrupt, the contents of the Program Counter (PC) (address of next instruction) and the PS are automatically pushed onto the R6 system stack. The effect is the same as if:

```

MFPS, -(SP)      ; Push PS
MOV R7, -(SP)    ; Push PC

```

had been executed.

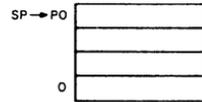
The new contents of the PC and PS are loaded from two preassigned consecutive memory locations which are called an "interrupt vector." The actual locations are chosen by the device interface designer and are located in low memory addresses. The first word contains the interrupt service routine address (the address of the new program sequence) and the second word contains the new PS which will determine the machine status including the operational mode and register set to be used by the interrupt service routine.

After the interrupt service routine has been completed, an RTI (return from interrupt) is performed. The two top words of the stack are automatically "popped" and placed in the PC and PS, respectively, thus resuming the interrupted program.

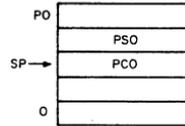
### 5.3.2 Nesting

Interrupts can be nested in much the same manner that subroutines are nested. In fact, it is possible to nest any arbitrary mixture of subroutines and interrupts without any confusion. By using the RTI and RTS instructions, respectively, the proper returns are automatic.

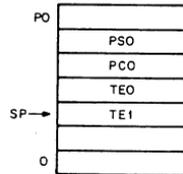
1. Process 0 is running; SP is pointing to location P0.



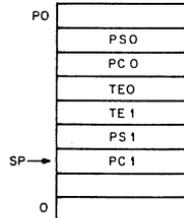
2. Interrupt stops process 0 with PC = PC0, and status = PS0; starts process 1.



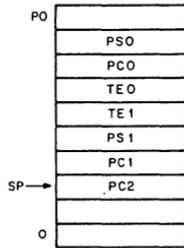
3. Process 1 uses stack for temporary storage (TE0, TE1).



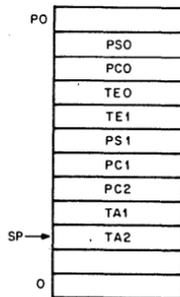
4. Process 1 interrupted with PC = PC1 and status = PS1; process 2 is started



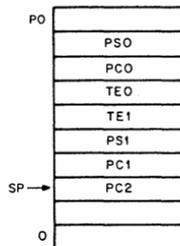
5. Process 2 is running and does a JSR R7,A to Subroutine A with PC = PC2.



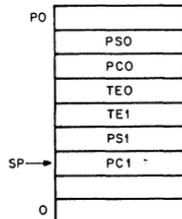
6. Subroutine A is running and uses stack for temporary storage.



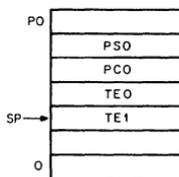
7. Subroutine A releases the temporary storage holding TA1 and TA2.



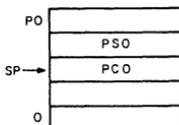
8. Subroutine A returns control to process 2 with an RTS R7,PC is reset to PC2.



9. Process 2 completes with an RTI instruction (dismisses interrupt) PC is reset to PC(1) and status is reset to PS1; process 1 resumes.



10. Process 1 releases the temporary storage holding TE0 and TE1.



11. Process 1 completes its operation with an RTI PC is restored to PC0 and status is reset to PS0.

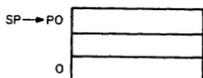


Figure 5-14 Nested Interrupt Service Routines and Subroutines

Note that the area of interrupt service programming is intimately involved with the concept of CPU and device priority levels.

#### 5.4 PROGRAMMING PERIPHERALS

Programming of LSI-11 modules (devices) is simple. A special class of instructions to deal with input/output operations is unnecessary. The bus structure permits a unified addressing structure in which control, status, and data registers for devices are directly addressed as memory locations. Therefore, all operations on these registers, such as transferring information into or out of them or manipulating data within them, are performed by normal memory reference instructions.

The use of all memory reference instructions on device registers greatly increases the flexibility of input/output programming. For example, information in a device register can be compared directly with a value and a branch made on the result:

```
CMP RBUF, #101
BEQ SERVICE
```

In this case, the program looks for 101 in the DLV11 Receiver Data Buffer Register (RBUF) and branches if it finds it. There is no need to transfer the information into an intermediate register for comparison.

When the character is of interest, a memory reference instruction can transfer the character into a user buffer in memory or to another peripheral device. The instruction:

```
MOV DRINBUF LOC
```

transfers a character from the DRV11 Data Input Buffer (DRINBUF) into a user-defined location.

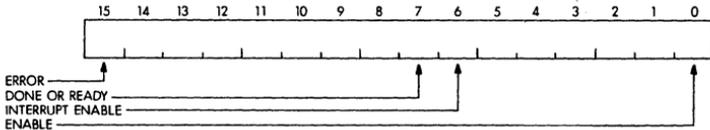
All arithmetic operations can be performed on a peripheral device register. For example, the instruction ADD #10, DROUT BUF will add 10 to the DRV11's Output Buffer.

All read/write device registers can be treated as accumulators. There is no need to funnel all data transfers, arithmetic operations, and comparisons through a single or small number of accumulator registers.

### 5.5 DEVICE REGISTERS

All devices are specified by a set of registers which are addressed as memory and manipulated as flexibly as an accumulator. There are two types of registers associated with each device: (1) control and status registers; (2) data registers. The following examples are general, for specific device register information refer to the applicable manual.

**Control and Status Registers**—Each device can have one or more control and status registers that contain the information necessary to communicate with that device. The general form, shown below does not necessarily apply to every device, but is presented as a guide.



| BIT | NAME             | DESCRIPTION  |
|-----|------------------|--|
| 15  | Error            | Set when an error occurs.  |
| 7   | Done or Ready    | Set when the device is either ready to accept new information, or has completed an operation and has data available. |
| 6   | Interrupt Enable | When set, an interrupt will be requested when a done or error condition occurs.                                      |
| 0   | Enable           | Set to allow the peripheral device to perform a function.  |

Many devices require less than sixteen status bits. Other devices will require more than sixteen bits and therefore will require additional status and control registers.

**Data Buffer Registers**—Each device has at least one buffer register for temporarily storing data to be transferred into or out of the processor. The number and type of data registers is a function of the device. The DLV11, for example uses single 8-bit data buffer registers. The DRV11 uses 16-bit data registers and some devices may use more than 1 register for data buffers.

**Interrupt Structure**—If the appropriate interrupt enable bit is set, in the control and status register of a device, transition from 0 to 1 of the READY or ERROR bit, where applicable, should cause an interrupt request to be issued to the processor. Also if READY or ERROR is a 1 when the interrupt enable is turned on, an interrupt request is made. If the device makes the request and the processor's priority is zero, and no higher priority devices are requesting an interrupt, the request is granted, and the interrupt sequence takes place.

- a. the current program counter (PC) and processor status (PS) are pushed onto the processor stack;
- b. the new PC and PS are loaded from a pair of locations (the interrupt vector) in addressed memory, unique to the interrupting device.

Since each device has a unique interrupt vector which dispatches control to the appropriate interrupt handling routine immediately, no device polling is required. The Return from Interrupt Instruction (RTI) is used to reverse the action of the interrupt sequence. The top two words on the stack are popped into the PC and PS, returning control to the interrupted sequence.

**Programming Example**—A DLV11 interrupt routine to service a low-speed paper tape reader, could appear as follows (assume the DLV11's interrupt vector is 60, and PRSER is the service routine for the device):

First the user must initialize the Stack Pointer (R6) and device vector locations. Then the user must initialize the service routine by specifying an address pointer and a word count:

```
INIT: MOV # BUFADR, R0           ; set address pointer into
      MOV # COUNT, COUNTR       ; register
      MOV # 101, RCSR           ; set counter
                                   ; enable DLV11
                                   ; interrupt enable &
                                   ; reader run enable,
                                   ; Program continues until
                                   ; interrupt occurs
```

When the interrupt occurs and is acknowledged, the processor stores the current PC and PS on the stack. Next it goes to the interrupt vector and picks up the new PC and PS location 60, 62. When the program was loaded, the address of PRSER would be put in location 60 and 200, in 72 (to set the processor's priority to 4 and inhibit new interrupts). The next instruction executed is the first instruction of the device service routine at PRSER.

```
PRSER:  MOVB RBUF, (R0) +        ; move character from
                                   ; DLV11's receiver data
                                   ; buffer register to buffer and
                                   ; increment pointer
      DEC COUNTR                 ; decrement character count
      BEQ DONE                   ; branch when COUNTR equals 0
      INC RCSR                   ; set reader enable for next
                                   ; character input

DONE:   RTI                      ; return to interrupted program
```



## CHAPTER 1

# INTRODUCTION

### 1.1 SOFTWARE SYSTEMS

This section describes the operating systems and programming languages available for the LSI-11 family of processors and systems, including the LSI-11, PDP-11/03, and PDP-11V03. It is intended to provide a brief introduction for the system designer or programmer using an LSI-11 system. Detailed software documentation is provided with the software options when purchased. Hence, no attempt is made to include complete programming and operating instructions in this handbook.

Software systems include the operating systems and programming languages described in this section, and diagnostic software, paper tape software, and special-purpose software options that are not described in this section. A list of currently available software options for the LSI-11, PDP-11/03, and PDP-11V03 is provided in Table 1-1.

### 1.2 OPERATING SYSTEMS

An operating system organizes a processor and peripherals into a useful tool for a range of applications. The operating systems, when used in LSI-11, PDP-11/03, or PDP-11V03 hardware is generally intended for a dedicated application. Two operating systems are available for use on these hardware configurations as listed below:

#### RT-11

A small, single-user foreground/background system that can support a real-time application job's execution in the foreground and an interactive or batch program development job in the background.

#### RSX-11S

A small, execute-only member of the RSX-11 family for dedicated real-time multiprogramming applications (requires a host RSX-11M system).

Chapters 4 and 5, respectively, describe these operating systems in detail. Included in each chapter are a general description of the requirements for the system, the monitor/executive characteristics, the file structures and data handling facilities, the user interfaces, the programmed monitor services, the system utilities, and the language processor options supported. For users who are not familiar with DIGITAL PDP-11 system components and services, Chapter 2 provides a summary description of the operating system's components and common facilities.

**Table 1-1 Software Options**

| <b>Option No.</b>      | <b>System Software</b>  | <b>Media</b> | <b>Reference</b>                                |
|------------------------|---|--------------|---|
| *QJ003-AY,<br>QJ003-CY | RT-11 Operating System, including Single Job and Foreground/Background Monitors and programs described in Chapter 4, and MACRO described in Chapter 6.  | Floppy Disk  | Chapters 4 & 6                                  |
| *QJ925-AY,<br>QJ925-CY | RT-11/FORTRAN   | Floppy Disk  | Chapters 4 & 7                                  |
| *QJ920-AY,<br>QJ920-CY | RT-11/BASIC   | Floppy Disk  | Chapters 4 & 8                                  |
| *QJ921-AY,<br>QJ921-CY | RT-11/MULTI-USER BASIC  | Floppy Disk  | Chapters 4 & 8                                  |
| *QJ922-AY              | RT-11/FOCAL   | Floppy Disk  | Chapter 4                                       |
| *QJ960-AY              | RT-11/SSP-11 Scientific Subroutine Package for RT-11/FORTRAN  | Floppy Disk  | —   |
| *QJ640-AY,<br>QJ640-CY | RSX-11S Operating System  | Floppy Disk  | Chapter 5                                       |
| *QJ945-AY              | RT-11/REMOTE-11   | Floppy Disk  | —   |
| ZJ215-RY               | LSI-11 Diagnostics  | Floppy Disk  | Section I,<br>Chapter 9                         |
| QJV10-CB               | Paper Tape Software, including:<br>ED-11 Text Editor<br>PAL11S Assembler<br>LINK11 Linker<br>DUMPAB Memory Dump Utility<br>ODT-11 On-Line Debugging Technique<br>IOX Input/Out Executive<br>Absolute Loader | Paper Tape   | PDP-11 Paper Tape Software Programming Handbook |
| QJV11-CB               | PROM Formatter  | Paper Tape   | Section I,<br>Chapter 7                         |
| ZJV01-RB               | LSI-11 Diagnostics  | Paper Tape   | Section I,<br>Chapter 9                         |

## **\*SOFTWARE SUPPORT CATEGORIES**

### **Category A (-AY Option No. Suffix)**

1. Upon notification by customer to the nearest DIGITAL office that the computer system, including all required prerequisite hardware and software is ready for the installation of the software, DIGITAL will install such software in any location within the contiguous United States, the District of Columbia, or a country in which DIGITAL or a subsidiary of DIGITAL has a software service facility. The notification must be received by DIGITAL and the system must be ready for installation within 30 days after the delivery of the software to customer or DIGITAL will have no obligation to install. Installation will consist of: (1) verification that all components of the software have been received by customer, (2) loading the software, and (3) executing a DIGITAL sample procedure.
2. During the 90-day period after installation, if the customer encounters a problem with the current unaltered release of the software which DIGITAL determines to be a defect in the software, DIGITAL will provide the following remedial service (on-site where necessary): (1) if the software is inoperable, apply a temporary correction or make a reasonable attempt to develop an emergency bypass, and (2) assist the customer to prepare a Software Performance Report (SPR) and submit it to DIGITAL.
3. During the 1-year period following installation, if the customer encounters a problem with the software which his diagnosis indicates is caused by a software defect, the customer may submit an SPR to DIGITAL. DIGITAL will respond to problems reported in SPRs which are caused by defects in the current unaltered release of the software via the maintenance periodical for the software, which reports SPRs received, code corrections, temporary corrections, generally useful emergency bypasses and/or notice of the availability of corrected code. Software Updates, if any, released by DIGITAL during the 1-year period, will be provided to the customer on DIGITAL's standard distribution media as specified in the applicable SPD. The customer will be charged only for the media on which such updates are provided, unless otherwise stated in the applicable SPD, at DIGITAL's then current media prices.

### **Category C (-CY Option No. Suffix)**

Software is provided on an 'as is' basis. Any software services, if available, will be provided at the then current charges.

## **1.3 LANGUAGES AND LANGUAGE PROCESSORS**

All PDP-11 operating systems offer a variety of programming language processors. A programming language is a tool that enables the user to state a problem that a computer can solve. A programming language is designed to be easily understood and manipulated by humans, while a language processor translates the instructions into the machine's language.

In general, the language processors available to run under an operating system are commensurate with the kind of applications for which the operating system is designed. For example, a real-time application environment could be a laboratory in which a scientific programming language is useful for problem solving. The real-time application operating systems offer FORTRAN IV language processors.

The programming languages that are discussed in this handbook are:

- |                   |  |
|-------------------|--|
| <b>MACRO</b>      | The general-purpose assembly language for PDP-11 computers.                          |
| <b>FORTRAN IV</b> | The language most used for scientific problem solving.                               |
| <b>BASIC</b>      | A language well-known in the educational/scientific communities for its ease-of-use. |

## OPERATING SYSTEMS

Operating systems have two basic functions: they provide services for application program development and act as an environment in which application programs run. The character that an operating system has, that is, the services and environment it supplies, is appropriate only for a certain range of program development and application requirements in order to serve selected needs efficiently. Operating systems for the PDP-11 family of computers, however, share many similar program development techniques and processing environments. This chapter describes some of the common features and characteristics of PDP-11 operating systems as well as some of their differences.

## 2.1 COMPONENTS AND FUNCTIONS

An operating system is a collection of programs that organizes a set of hardware devices into a working unit that people can use. Figure 2-1 illustrates the relationship between users, the operating system, and the hardware. PDP-11 operating systems basically consist of two sets of software: the monitor (or executive) software and the system utilities.

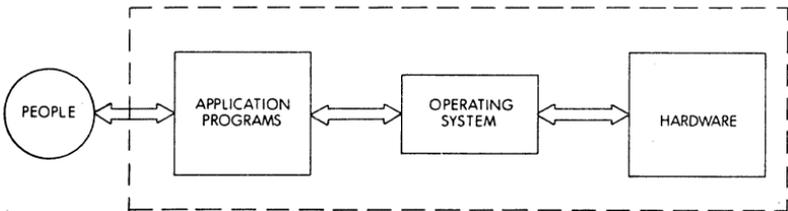


Figure 2-1 Computer System Components

An operating system monitor is an integrated set of routines that acts as the primary interface between the hardware and a program running on the system, and between the hardware and the people who use the system. The monitor's basic functions can be divided among the routines that provide the following services:

- device and data management
- user interface
- programmed processing services
- memory allocation
- processor time allocation

In general, a monitor can have two distinct operating components: a permanently-resident portion and a transient portion. When a monitor is loaded into memory and started, all of the monitor is resident in memory. Its first duty is to interface with the operator running the system. The monitor simply waits until an operator requests some service, and then performs that service. In general, these services include loading and starting programs, controlling program execution, modifying or retrieving system information, and setting system parameters. In most systems, these functions are serviced by transient portions of the monitor.

In some cases, when the monitor initiates another program's execution, the transient portion of the monitor can be over-written by the loaded program or swapped out. The permanently-resident portion remains in memory to act on requests from the program. These generally include I/O services such as file management, device dependent operations, blocking and unblocking data, allocating storage space, and managing memory areas. In large systems, these services might also include inter-task communication and coordination, memory protection and parity checking, and task execution scheduling.

The dividing line between permanently-resident and transient portions of the monitor, however, is not strictly based on user-interface functions and program-interface functions. In some systems, special monitor routines that service either the operator or programs might be stored on the system device, and are called in to memory only as needed. The concern for space in small systems usually determines what portions of the monitor are resident at any time. The programmer or operator can control the size of the monitor, based on the needs for memory.

In some cases, the user can adjust the size of the monitor by eliminating certain features that are not needed in an application environment. RSX-11S is an example of such a system. The RSX-11S system's monitor (called an executive) is always permanently-resident when the system is operating. In this case, the user concerned with size can eliminate routines that perform unneeded operations. In general, however, all PDP-11 operating systems are designed to be flexible enough to operate in a relatively wide range of hardware environments.

System utilities are the individual programs that are run under control of the monitor to perform useful system-level operations such as source program assembly or compilation, object program linking, and file management.

System utility programs enhance the capabilities of an operating system by providing users with commonly-performed general services. There are three classes of system utilities: those used solely or primarily for program development, those used for file management, and those used to perform special system management functions.

Program development utilities include text editors, assemblers and compilers, linkers, program librarians, and debuggers. File management utilities include file copy, transfer, and deletion programs, file format

translators, and media verification and clean-up programs. System management utilities vary from system to system, depending on the purpose and functions the system services. Some examples are system information programs, user accounting programs, and error logging and on-line diagnostic programs.

## **2.2 PROCESSING METHODS**

The basic distinctions among operating systems are the processing methods they use to execute programs. The basic distinctions among processing methods to be discussed here are:

- single-user and multi-user
- single-job and foreground/background
- foreground/background and multi-programming
- timesharing and event-driven multi-programming

A single-user operating system views demands upon its resources as emanating from a single source. It has only to manage the resources based on these demands. As an effect, these systems do not require account numbers to access the system or data files. RT-11 is a single-user operating system.

An RT-11 system can operate in two modes: as a single-job system, or as a foreground/background system. In a foreground/background system, memory for user programs is divided into two separate regions. The foreground region is occupied by a program requiring fast response to its demands and priority on all resources while it is processing; for example, a real-time application program. The background region is available for a low-priority, pre-emptable program; for example, a compiler.

Two independent programs, therefore, can reside in memory, one in the foreground region and one in the background region. The foreground program is given priority and executes until it relinquishes control to the background program. The background program is allowed to execute until the foreground program again requires control. The two programs effectively share the resources of the system. When the foreground program is idle, the system does not go unused. Yet, when the foreground program requires service, it is immediately ready to execute. I/O operations are processed independently of the requesting job to ensure that the processor is used efficiently as well as enable fast response to all I/O interrupts.

The basis of foreground/background processing is the sharing of a system's resources between two tasks. An extension of foreground/background processing is multi-programming. In multi-programmed processing, many jobs, instead of only two, compete for the system's resources. While it is still true that only one program can have control of the CPU at a time, concurrent execution of several tasks is achieved because other system resources, particularly I/O device operations, can execute in parallel. While one task is waiting for an I/O operation to complete, for example, another task can have control of the CPU.

The RSX-11 family of operating systems employs multi-programmed processing based on a priority-ordered queue of programs demanding system resources. In this case, memory is divided into several regions called partitions, and all tasks loaded in the partitions can execute in parallel. Program execution, as in the RT-11 foreground/background system, is event-driven. That is, a program retains control of the CPU until it declares a significant event—normally meaning that it can no longer run, either because it has finished processing, or because it is waiting for another operation to occur. When a significant event is declared, the RSX-11 executive gives control of the CPU to the highest priority task ready to execute. Furthermore, a high-priority task can interrupt a lower-priority task if it requires immediate service.

### **2.3 DATA MANAGEMENT**

Digital computers deal with binary information only. The way in which people interpret and manipulate the binary information is called data management.

This section describes PDP-11 software data management structures and techniques, from the physical storage and transfer level to the logical organization and processing level. This includes:

- ASCII and binary storage formats—how binary data can be interpreted
- physical and logical data structures—the difference between how data storage devices operate and how people use them
- file structures—how physical units of data are logically organized for ease of user reference
- file directories—how files are located and retrieved
- file protection—how files are protected from unauthorized users
- file naming conventions—how files are identified

#### **2.3.1 Physical and Logical Units of Data**

Physical units of data are the elements which digital computer devices use to store, transfer and retrieve binary information.

A bit (binary digit) is the smallest unit of data that computer systems handle. An example of a bit is the magnetic core used in some processor memories that is polarized in one direction to represent the binary number 0 and in the opposite direction to represent the binary number 1.

In PDP-11 computers, a byte is the smallest memory-addressable unit of data. A byte consists of eight binary bits. An ASCII character code can be stored in one byte. Two bytes comprise a 16-bit word. A word is the largest memory-addressable unit of data. Some machine instructions are stored in one word.

The smallest unit of data that an I/O peripheral device can transfer is called its physical record. The size of a physical record is usually fixed and depends on the type of device being referenced. For example, a card reader can read and transfer 80 bytes of information, stored on an 80-column punched card. The card reader's physical record length is 80 bytes.

A block is the name for the physical record of a mass-storage device such as disk or magnetic tape. Data blocks associated with the RXV11 floppy disk system are actually 128 bytes of information comprising one sector. Each floppy disk is formatted into 77 concentric tracks, each track containing 26 sectors.

Physical blocks can be grouped into a collection called a device or a physical volume. This collection generally has a size equal to the capacity of the device media. The term physical volume is generally used with removable media, such as a floppy disk or magnetic tape.

Logical units of data are the elements manipulated by people and user programs to store, transfer and retrieve information. The information has logical characteristics, for example, data type (alphabetic, decimal, etc.) and size. The logical characteristics are not device dependent; they are determined by the people using the system.

A field is the smallest logical unit of data. For example, the field on a punched card used to contain a person's name is a logical unit of data. It can have any length arbitrarily determined by the programmer who defines the field.

A logical record is defined as a collection of fields treated as a unit. It can contain any logically-related information, in any one or several data types, and it can be any user-determined length. Its characteristics are not device dependent, but can be physically defined. For example, a logical record can occupy several blocks, or it can reside in a single block, or several logical records can reside in a single block. Its characteristics are determined by the programmer.

A file is a logical collection of data that occupies one or more blocks on a mass-storage device such as a floppy disk. A file is a system-recognized logical unit of data. Its characteristics can be determined by the system or the programmer.

A file can be a collection of logical records treated as a unit. An example of a Employee File which contains one logical record in the file for each employee. Each record contains an employee's name and address and other pertinent information. If the logical record length is 50 bytes, and there are 200 employees, the complete Employee file could be stored in 80 128-byte sections. Depending on the file structure used in the system, the data could be scattered over the disk, or could be located in sequential sectors and tracks.

A logical volume is a collection of files that reside on a single disk or tape. It is the logical equivalent of a physical device unit (a physical volume) consisting of physical records, such as a disk. The files on a volume may have no specific relationship other than their residence on the same magnetic media. In some cases, however, the files on a volume may all belong to the same user of the system.

Figure 2-2 illustrates some of the kinds of physical and logical units of data that PDP-11 computer systems handle.

### 2.3.2 Data Storage and Transfer Modes

All PDP-11 operating systems use two basic methods of data storage: ASCII and binary. Data stored in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. The 7-bit code occupies the low-order seven bits of an 8-bit byte. Depending on the operating system's storage techniques, the high-order bit may be used for parity checking, special formatting, or it may be ignored. Text files such as source programs are examples of data stored in ASCII format.

Binary storage always uses all eight bits of a byte to store information. The significance of any bit varies depending on the kind of information to be stored. Machine instructions, two's complement integer data, and

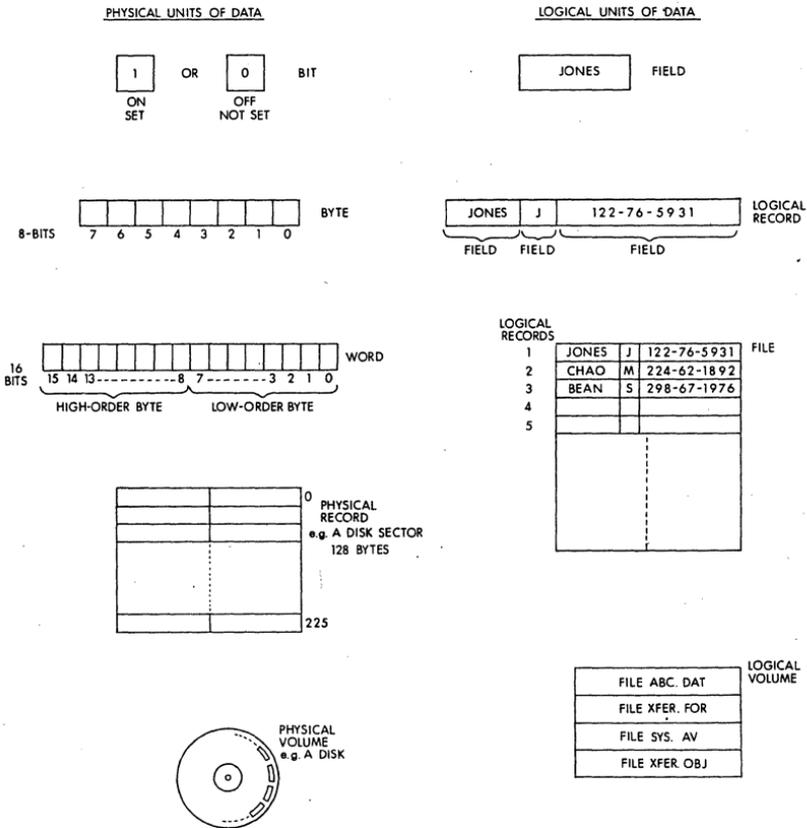


Figure 2-2 Physical and Logical Data Storage

floating-point numeric data are some examples of data stored in binary format.

Figure 2-3 illustrates the way in which binary data can be interpreted as either ASCII data or machine instructions. The figure shows two examples of a word of storage containing the same sequence of bits, interpreted first as two ASCII characters and second as a machine instruction. When a word of storage is interpreted as two ASCII characters, the binary digits are grouped into octal digits in a bitwise manner. Each byte is grouped into three octal digits. The low-order two octal digits contain three binary digits. The high-order octal digit contains two binary digits. When a word of storage is interpreted as a machine instruction, the binary digits are grouped into six octal digits in a wordwise manner. Proceeding from the low-order binary digit, each group of three binary digits is interpreted as an octal digit. The single remaining high-order binary digit is interpreted as an octal digit.

In large, sophisticated systems the way in which data is stored on the byte or bit level is rarely a concern of the application programmer. The operating system handles all data storage and transfer operations. In small systems such as RT-11, the programmer can become involved in data storage formats. A particular application may require the selection of a particular storage format.

Formatting can be applied to define the type of data file being processed. In the RT-11 system, there are four types of binary files; each type signifies that a special interpretation applies to the kind of binary data stored. For example, a memory image file is an exact picture of what memory will look like when the file is loaded to be executed. A relocatable image file, however, is an executable program image whose instructions have been linked as if the base address were zero. When the file is loaded for execution, the system has to change all the instructions according to the offset from base address zero.

### **2.3.3 I/O Devices and Physical Data Access Characteristics**

In a PDP-11 computer system, data moves from external storage devices into memory, from memory into the CPU registers, and back out again. The "window" from external devices to memory and the CPU is called the I/O page. Each external I/O device in a computing system has an external page address assigned to it. Figure 2-4 illustrates the data movement path in a PDP-11 computing system.

Although all external devices transmit and receive data through the LSI-11 BUS devices differ in their ability to store, retrieve or transfer data. Almost all PDP-11 operating systems provide device independence between devices that have similar characteristics and, where possible, between differing devices in situations where the data manipulation operations are functionally identical. Primarily, PDP-11 operating systems differentiate between:

- file-structured and non-file-structured devices
- block-replacable and non-block-replacable devices

Terminals, card readers, paper tape readers, paper tape punches and line printers are examples of devices that do not provide any means to selectively store or retrieve physical records. They can transfer data only in the sequence in which it physically occurs.

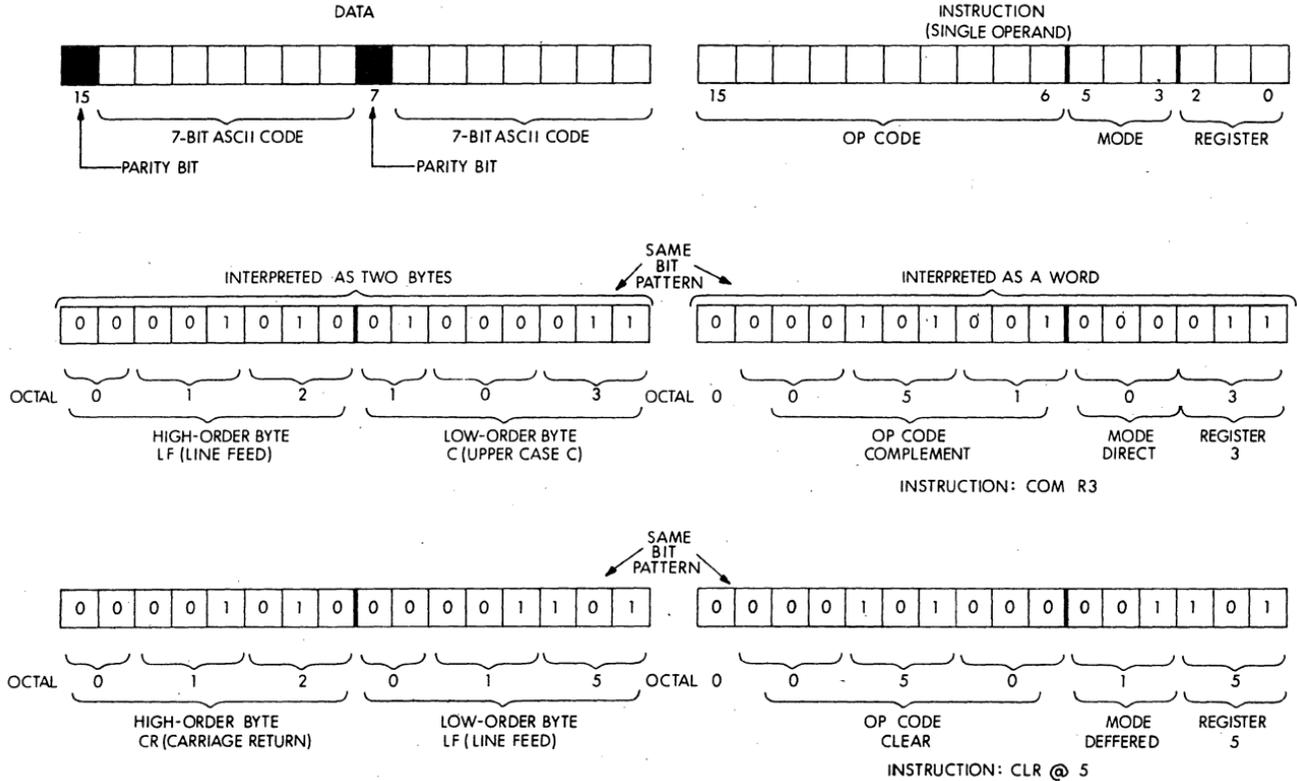


Figure 2-3 ASCII and Binary Storage

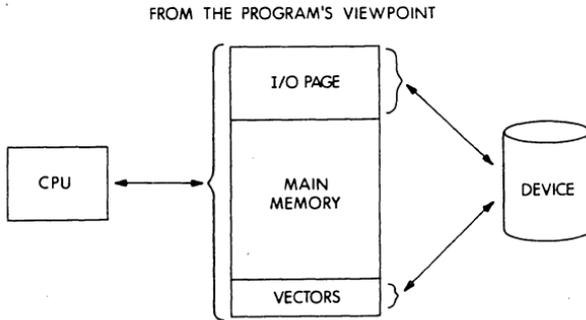
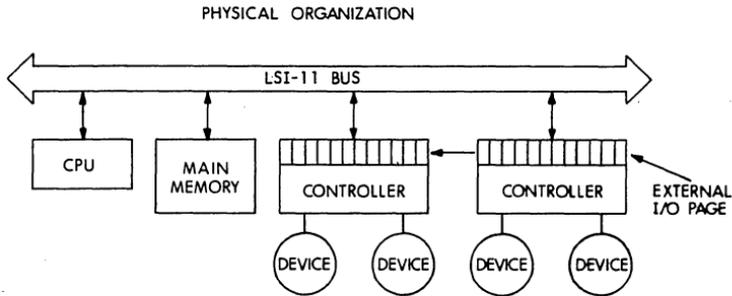


Figure 2-4 Memory and I/O Devices

In contrast, mass-storage devices such as floppy disk, and magnetic tape have the ability to store and retrieve physical records selectively. For example, an operating system can select a file from among many logical collections of data stored on the medium.

Mass-storage devices are called file-structured devices, since a file, consisting of a group of physical records, can be stored on and retrieved from the device. Terminals, card readers, paper tape readers/punches and line printers are called non-file-structured devices because they do not have the ability to selectively read or write the physical records comprising a file.

Finally, mass-storage devices differ in their ability to read and write physical records. Disk devices are block-replaceable devices, because a given block can be read or written without accessing or disturbing all the other blocks on the medium. Magnetic tape is not a block-replaceable device.

A device's physical data access characteristics determine which data transfer methods are possible for that device. Non-file-structured devices allow sequential read or write operations only. Non-block replaceable devices allow sequential or random read operations, but allow sequential write operations only. Block-replaceable devices allow both sequential and random read or write operations. Figure 2-5 summarizes the read/write capabilities of each category of I/O device.

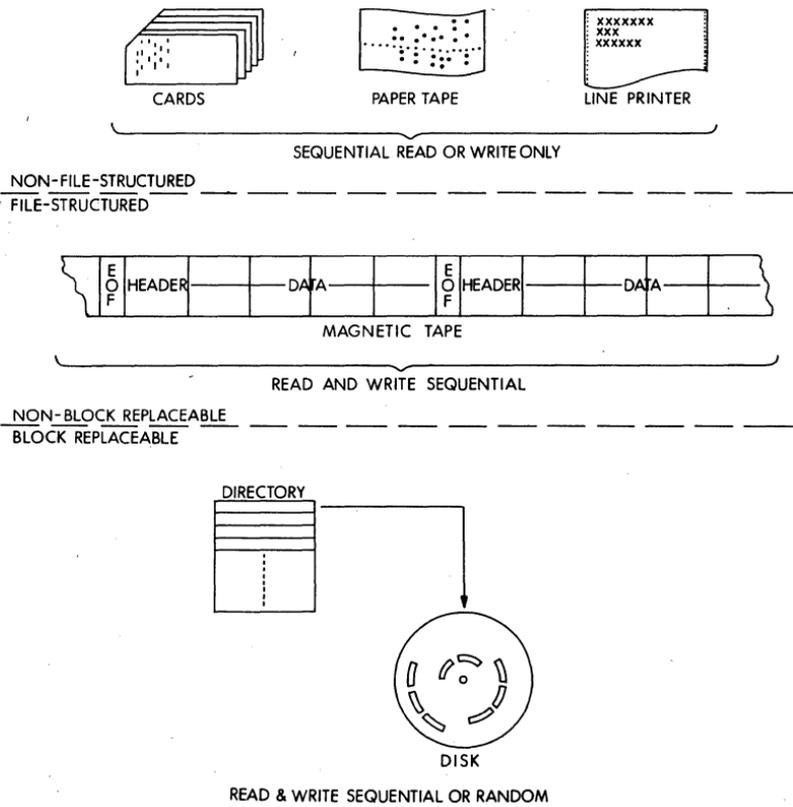


Figure 2-5 Device Characteristics

**2.3.4 Physical Device Characteristics and Logical Data Organizations**

One of the most important services an operating system provides is the mapping of physical device characteristics into logical data organizations. Users do not have to write the software needed to handle input and output to all standard peripheral devices, since appropriate routines are supplied with the operating system.

There are generally two sets of routines provided in any operating system, depending on its complexity. They are:

- device drivers or handlers
- file management services

Device drivers and handlers can perform the following operations to relieve the user of the burden of I/O services, file management, overlapping I/O considerations and device dependence:

- drive I/O devices
- provide device independence
- block and unblock data records for devices, if necessary
- allocate or deallocate storage space on the device
- manage memory buffers

These routines may exist in the system as part of the monitor or executive, as in RT-11.

An operating system can also provide a uniform set of file management services. For example, the RT-11 system provides file management services through the part of the monitor called the User Service Routine (USR). The User Service Routine provides support for the RT-11 file structure. USR loads device handlers, opens files for read/write operations, and closes, deletes and renames files.

In summary, an operating system maps physical device characteristics into logical file organizations by providing routines to drive I/O devices and interface with user programs. Figure 2-6 illustrates the transition between the user interface routines and the I/O devices.

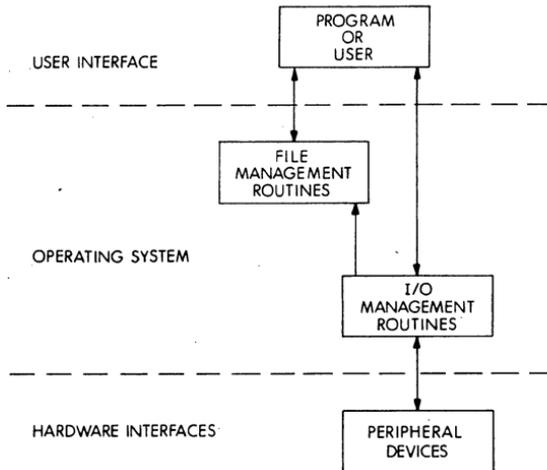


Figure 2-6 Device Control and File Management Services

### 2.3.5 File Structures and Access Methods

A file structure is a method of organizing logical records into files. It describes the relative physical locations of the blocks comprising a file. The file structure or structures a particular operating system employs is a product of the way in which the system views the particular I/O devices and the kinds of data processing requirements the system fulfills.

File structure is important because a file can be effective in an application only if it meets specific requirements involving:

|            |  |
|------------|--|
| SIZE       | Growth of the file may require a change in the file structure or repositioning of the file.                                  |
| ACTIVITY   | The need to access many different records in a file or frequently access the same file influences data retrieval efficiency. |
| VOLATILITY | The number of additions or deletions made to a file may affect the access efficiency.  |

An access method is a set of rules for selecting logical records from a file. The simplest access method is sequential: each record is processed in the order in which it appears. Another common access method is direct access: any record can be named for the access. The non-block replaceable devices, such as paper tape and magnetic tape, can only be processed sequentially. The block replaceable devices, such as disk, can be processed by either access method, but direct access takes fullest advantage of the device characteristics.

PDP-11 operating systems provide a variety of file structures and access methods appropriate to their processing services: All PDP-11 file structures are, however, based on some form of the following basic file structures:

| FILE STRUCTURE | ACCESS METHODS              |
|----------------|-----------------------------|
| Linked         | Sequential                  |
| Contiguous     | Sequential or Direct Access |
| Mapped         | Sequential or Direct Access |

Linked files are self-expanding series of blocks which are not physically adjacent to one another on the device. The operating system records data blocks for a linked file by skipping several blocks between each recording. The system then has enough time to process one block while the medium moves to the next block to be used for recording. In order to connect the blocks together, each block contains a pointer to the next block of the file. Figure 2-7 shows the format of a linked file.

Linked files are especially suited for sequential processing where the final size of the file is not known. It readily allows later extension, since the user can add more blocks in the same way the file was created. In this way, linked files make efficient use of storage space. Linked files can also be easily joined together.

The blocks of contiguous files are physically adjacent on the recording medium. This format is especially suited for random (direct access) pro-

cessing, since the order of the blocks is not relevant to the order in which the data is processed. The system can readily determine the physical location of a block without reference to any other blocks in the file. Figure 2-7 also shows the format of a contiguous file.

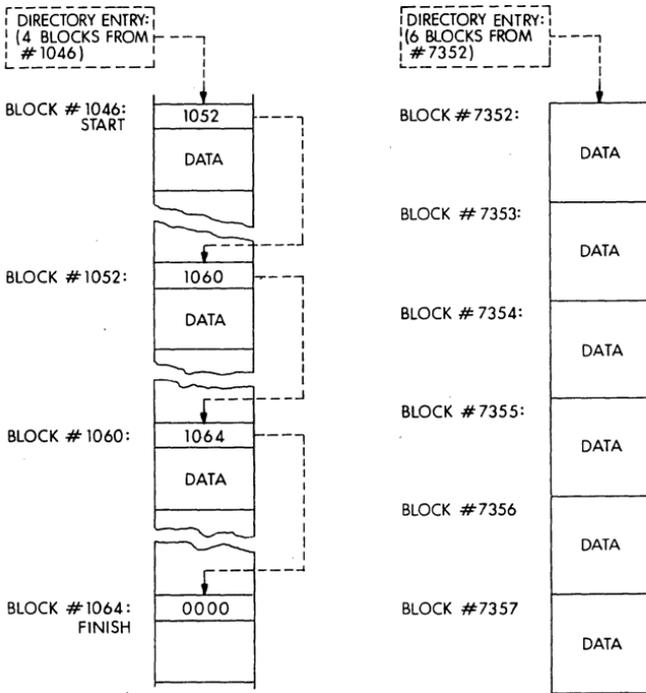


Figure 2-7 Linked and Contiguous File Structures

Mapped files are virtually contiguous files, that is, they appear to the user program to be directly-addressable sets of adjacent blocks. The files may not, however, actually occupy physically contiguous blocks on the device. The blocks can be scattered anywhere on the device. Separate information, called a file header block, is maintained to identify all the blocks comprising a file. This method provides an efficient use of storage space and allows files to be extended easily, while still maintaining a uniform program interface. Figure 2-8 illustrates a mapped file format.

If desired, a mapped file can be created as a contiguous file to ensure the fastest random accessing, in which case it is both virtually and physically contiguous.

The basic file structure discussed above can be modified or combined to extend the features of each type for special-purpose logical processing methods. Some examples are indexed files and global array files.

Indexed files are actually two contiguous files. One file acts as an ordered "map" of a second file containing the target data. The index

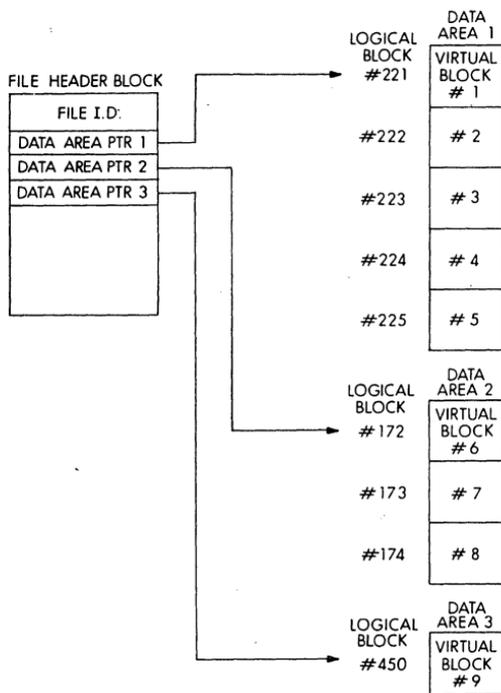


Figure 2-8 Mapped File Structure (Non-contiguous file)

portion or "map" contains either an ordered list of key data selected from the target data records or pointers to data records in the second file, or both. The target data records can be processed in the order of the index portion, or the target data records can be selected by searching through the index portion for the key data identifying the records. These methods of logically processing the target data are called indexed sequential access and random access by key, respectively.

Global array files display a special form of linked file structure. The arrays themselves are a logical tree-structured organization consisting of one or more subscripted levels of elements. All elements on a particular subscripting level are stored in a single chain of linked blocks. At the end of each block in the chain is a pointer to the next block in the chain. The levels of the array (all the block chains) are linked together through pointers in the first block of each chain. This file structure ensures that the time it takes to access any element of the array is minimal.

### 2.3.6 Directories and Directory Access Techniques

Just as file structure and access methods are required to locate records

within files, directory structures and directory access techniques are required to locate files within volumes.

A directory is a system-maintained structure used to organize a volume into files. It allows the user to locate files without specifying the physical addresses of the files. It is a direct access method applied to the volume to locate files.

RT-11 supports the simplest kind of file directory. When floppy disk medium is initialized for use, the system creates a directory on the device. Each time a file is created, an entry is made in the directory that identifies the name of the file, its location on the device and its length. When access to the file is requested thereafter, the system examines the directory to find out where the file is actually located. The system can access the file quickly without having to examine the entire device.

RT-11 also permits non-block replaceable media such magnetic tape to be given a file structure. This media has no directory because a directory could not be updated and replaced. Instead, each file is preceded by one or more header records which contain the directory information such as the file's name. The operating system can locate a file by scanning the volume and reading each file header until the correct one is found. The file can then be processed by a sequential access method.

### **2.3.7 File Naming**

The most common way users communicate their desire to process data is through file specifications. A file specification uniquely identifies and locates any logical collection of data which is on-line to a computer system.

A compiler, for example, needs to know the name and location of the source program file that it is to compile; it also needs to know the name that the user wants to use for the output object program and listing files it produces. Most PDP-11 operating systems share the same basic format for input and output file specifications.

In the RT-11 system, a file specification consists of the name of the device on which the file resides, a filename, and a filename extension in the following format:

dev:filnam.ext

The colon is part of the device name, separating it from the filename on the right. The period is part of the filename extension, separating it from the filename on the left.

PDP-11 operating systems use the same device names for the devices they can access. A device name consists of a 2-letter mnemonic and, for multiple devices of the same kind, a one-digit number indicating the device unit number. For example, the name "DX1:" is used to identify the RXV11 disk drive number 1.

In the RT-11 system, a filename is a 1- to 6-character alphanumeric name designated by the user. For example, "SYMBOL", "RL12", and "NORT4" are examples of filenames.

A filename extension is a 1- to 3-character alphanumeric name preceded by a period. The extension can either be assigned by the user or, if unspecified, assigned by the system. The extension generally indicates the format of a file. System-assigned and recognized extensions make it easy for the user and the system to distinguish between different forms of a file. For example, a file having the extension ".FOR" is recognized by the FORTRAN compiler as a source program written in FORTRAN. A file with the extension ".OBJ" is recognized by the Linker as an object program, a legal input file. When in the process of compiling and linking a FORTRAN program, the user has only to specify a filename to the compiler and Linker. The FORTRAN compiler will compile the file whose extension is ".FOR" and produce a file with the same filename whose extension is ".OBJ". The Linker will link the file whose extension is ".OBJ".

In most cases, the user does not have to issue a complete file specification. The PDP-11 operating systems use default values when a portion of a file specification is not supplied. The filename extension defaults, as indicated previously, depend on the kind of operation being performed.

The device name, if omitted, is normally assumed to be the system device. For example, the file specification "FILE.DAT" is equivalent to the specification "DX0:FILE.DAT," if the system device is RXV11 drive 0. If the drive number is omitted, the number is assumed to be 0. For example "DX:" is equivalent to "DX0:"; it signifies RXV11 drive 0.

In addition to relying on defaults in the file specification, the user can also put an asterisk in place of a filename, filename extension, account number or version number to indicate a class of files. The asterisk convention, also called the wildcard convention, is commonly used in PDP-11 operating systems when performing the same operation on related files. For example, the file specification "DX1:[2,]PROG.\*" refers to all files on DX1: under account [2,1] with a filename PROG and any extension. The file specification "DX:[\*,\*]FILE.SAV" refers to the files under all accounts on RXV11 drive unit 0 named FILE.SAV. The file specification "DX:\*.OBJ" refers to all files on drive unit 0 that have the extension .OBJ.

## 2.4 USER INTERFACES

A user interface refers to both the software that passes information between an operator and a system and the "language" that a system and an operator use to communicate. In the latter sense, a user interface consists of commands and messages. Commands are the instructions that the user types on a terminal keyboard (or gives to a batch processor) to tell the system what to do. Messages are the text that a system prints on a terminal (or line printer) that tells the operator what is going on, for example, prompting messages, announcements and error messages. This section discusses commands, the portion of the user interface that tells the system what to do, and prompting messages, the messages the system prints when it is ready to receive commands.

There are basically four types of commands used in PDP-11 operating systems:

- monitor or command language commands—used to request services from the system as a whole
- I/O commands—used to direct any kind of I/O operation (often a part of monitor commands)
- special terminal commands—these use keys on a terminal for special functions
- system program commands—commands used in system programs that perform operations relevant only for the individual program

Since system program commands are relevant only for individual system programs, and not operating systems in general, this section discusses monitor and command language commands, I/O commands and special terminal commands only.

#### **2.4.1 Special Terminal Commands**

Special terminal commands are a set of keys or key combinations that, when typed on a terminal, are used to perform special functions. For example, a user normally types the carriage return key at the end of an input command string to send the command to the system, which responds immediately by performing a carriage return and line feed on the terminal. The key labelled RUBOUT or DELETE is used to delete the last character typed on the input line.

The most significant special terminal commands are those used with the key labelled CTRL (control). When the CTRL key is held down (like the shift key) and another key is typed, a control character is sent to the system to indicate that an operation is to be performed.

For example, a line currently being entered (whether as part of a command or as text) will be ignored by the system by typing a CTRL/U combination (produced by holding down the CTRL key and typing a U key). The user can then enter a new input line. The CTRL/U function is the same as typing successive RUBOUT keys to the beginning of a line. CTRL/U is standard on PDP-11 operating systems.

Another example is the CTRL/O function. If, during the printing of a long message or a listing on the terminal, the user types a CTRL/O, the teleprinter output will stop. The program printing the output, however, will still continue. The user can type a CTRL/O again to resume output. CTRL/O is a standard function on PDP-11 operating systems.

#### **2.4.2 I/O Commands**

As mentioned in section 2.3.7, users communicate their intentions to process data files by issuing I/O commands consisting of at least one file specification. Normally, the I/O commands used in a system are standard throughout that system; in addition, most PDP-11 operating systems share the same basic I/O command string format.

For example, in RT-11 systems, the monitor includes a Command String Interpreter routine that parses and validates I/O command strings. The Command String Interpreter routine is used both by the monitor and the system programs to obtain a definition from the user of the input file or

files to be processed and a definition of the output file or files to be created. User-written programs can also call the Command String Interpreter to obtain I/O specifications from the operator at a terminal.

A standard I/O command string consists basically of one or more input and/or output file specifications. An I/O command string uses the following general format:

filespec=filespec

where "filespec" is a file specification (see section 2.3.7) and the equal sign (=) represents a character (usually equal sign or less-than sign) that separates an input file specification on the right from an output file specification on the left. If there is more than one input file specification or output file specification, they are separated from each other by commas. For example, if there are two output file specifications and three input file specifications:

filespec,filespec= filespec,filespec,filespec

If the program requesting an I/O command string does not need either an input or output file specification, the equal sign (or less than sign) is not present in the I/O command string.

As an example, the user can run the RT-11 operating system's Linker system utility to link one or more object program files and produce an executable program file and a load map. The I/O command issued to the Linker could be:

\*DX:RESTOR.SAV,DX1:RESTOR.MAP=DX:RESTOR.OBJ/B:500

Where:

- |                |  |
|----------------|--|
| *              | Is the prompting character printed by the Linker program indicating that it wants an I/O command string. After it is printed, the user types the remaining characters on the line. |
| DX:RESTOR.SAV  | Is the name of the executable program file to be created. It will be stored on the diskette mounted on drive unit 0.   |
| DX1:RESTOR.MAP | Is the name of the load map file to be created. It will be stored on the diskette mounted on drive unit 1.   |
| DX:RESTOR.OBJ  | Is the name of the object module (input file) to be used to create RESTOR.SAV.   |
| /B:500         | Is a command string switch indicating that the RESTOR.SAV program is to be linked with its starting address at location 500.   |

Command string switches are simply ways of appending qualifying information to an I/O command string. The switches used vary from program to program. They are not usually required in an I/O command string, since most programs assume default values for any switch.

### 2.4.3 Monitor and Command Language Commands

The primary system/user interface is provided in PDP-11 operating systems by either monitor software or special command language interface programs that run under the monitor. The monitor software and command languages allow the user to request the system to set system parameters, load and run programs, and control program execution.

An input command line consists of the command name (an English word that describes the operation to be performed) followed by a space and a command argument. For example, the command to run a program is the word "RUN" followed by the name of the file containing the program. If the command name is long, it can usually be abbreviated. For example, the command to set the system's date to August 27, 1975 could be "DATE 27-AUG-75." The system could also accept "DA 27-AUG-75." A command input line is normally terminated by typing the carriage return key on the console keyboard, although in some systems the key labelled ALTMODE is also used. Typing the carriage return key (or ALTMODE key) tells the system that the command line is ready to be processed.

In the RT-11S system, a monitor component called the Keyboard Monitor prints a period (.) on the left column of the system's console to indicate that the monitor is ready to accept commands. The user enters a command string on the same line following the period, and terminates the command string by typing the carriage return key.

In the RSX-11 system, a command interface called the Monitor Console Routine (MCR) allows the user to perform system level operations. When MCR is activated, it prints the characters "MCR>" on the terminal. The user enters a command on the same line as the prompt, and terminates the line with a carriage return or an ALTMODE. If the line is terminated with a carriage return, MCR prints a prompt and is ready to receive another command. If the line is terminated with an ALTMODE, MCR does not reactivate. To reactivate MCR at a terminal, the user types a CTRL/C.

There are two kinds of commands that MCR accepts: general user commands and privileged user commands. General user commands provide system information, run programs, and mount and dismount devices. Privileged user commands control system operation and set system parameters.

To run a system utility, the user can type the utility's name in response to an MCR prompt. When the utility is loaded, it prints a prompt to request a command string. The user can then enter a command string. When it completes the operation, the user can enter another command or type CTRL/Z to terminate the program. For example, to run the PIP utility program:

```
MCR>PIP
PIP>command string
PIP>↑Z
MCR>
```

If the user wants to issue only one command to the utility, the user can type the command string on the same line with the MCR request to run the utility. For example:

```
MCR>PIP command string  
MCR>
```

## 2.5 PROGRAMMED SYSTEM SERVICES

All PDP-11 operating systems provide access to their services through requests that programs or tasks can issue during execution.

The RT-11 system provides a variety of programmed requests. There are programmed requests that perform file manipulation, data transfer and other system services such as loading device handlers, setting a mark time for asynchronous routines, suspending a program, and calling the Command String Interpreter. Monitor services are requested through macro instructions in assembly language programs, or through calls to the system library in FORTRAN programs. The basis of the programmed requests in RT-11 are the EMT (Emulator Trap) instructions. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT instruction and executes the operation requested. When the operation is performed, the monitor returns control to the program.

## 2.6 SYSTEM UTILITIES

PDP-11 operating systems provide, in general, three kinds of system utility programs: program development utilities, file management utilities, and special system management utilities.

Most PDP-11 operating systems include the following kinds of program development utilities:

- |             |   |
|-------------|---|
| Text Editor | An editor is used for on-line interactive creating and editing of source programs or data files. An editor uses several sets of commands that search for character strings, insert, move or delete characters or lines, and insert, move, delete or append whole buffers of data. Although a text editor is designed for interactive use, it can also usually be run under a batch processor if the operating system supports batch processing. |
| Assembler   | An assembler accepts a source program written in PDP-11 symbolic machine language and produces an object module as output.  |
| Linker      | A linker is a program that accepts relocatable object programs created by an assembler or compiler and produces an executable program module. Some linkers provide facilities for overlaid program segments to enable a large program to execute in a small memory area.  |
| Librarian   | A librarian is a program that enables a programmer to create, update, modify, list and maintain library files. A library file is an object module (or modules) that is used several times in a program, used by more than   |

one program, or routines that are related and simply gathered together to incorporate easily into a program.

**Debugger**

A debugger is a program which enables a user to troubleshoot program errors dynamically through a terminal keyboard. It is normally linked with a program and runs as part of the program.

Some of the file management utilities available on many operating systems include:

**PIP**

The Peripheral Interchange Program (PIP) is a general-purpose file utility package for both the general user and programmer and the system manager. PIP normally handles all files with the operating system's standard data formats. In general, the program transfers data files from any device in the system to any other device in the system. PIP can also delete or rename any existing file. Some operating systems include special file management operations in the PIP utility, such as directory listings, device initialization and formatting, and account creation.

**FILEX**

The File Exchange program is a special-purpose file transfer utility similar in operation to PIP. It provides the ability to copy files stored in one kind of format to another format. This enables a user to create data on one system in a special format and then transfer the data to a device in a format that another system can read.

**DUMP**

DUMP displays all or selected portions of a file on a terminal or line printer. In general, DUMP enables the user to inspect the file in any of three modes: ASCII, byte, and octal. In ASCII mode, the contents of each byte is printed as an ASCII character. In byte mode, the contents of each byte is printed as an octal value. In octal mode, the contents of each word is printed as an octal value.

**VERIFY**

In general, a VERIFY program checks the readability and validity of data on a file-structured device.

Most system management utilities included in an operating system are dependent on the function the operating system serves.



## LANGUAGE PROCESSORS

### 3.1 LANGUAGE TRANSLATION SYSTEMS

A programming language is a system of symbols and syntax which can be used to describe a procedure that a computer can execute. A language processor is a program that translates one programming language into another. A language processor reads a program written in a language easily understood by people and translates it into a program written in the binary language of a digital computer. The program which the processor reads is called the source program. The program which the processor writes is called the object program.

#### Assemblers

An assembler is a language processor written for a particular digital computer. The source language it translates is called assembly language. There is a one-to-one correspondence between most of the mnemonics used as the assembly language operators and the binary instructions of the computer. Some exceptions are macro calls and assembler directives.

During the language translation process, an assembler performs a number of error checking operations. When an error is detected, the assembler notes the error and attempts to continue processing. At the end of processing, the assembler produces an error listing showing all the occurrences of errors, with substantial messages to the programmer. In addition to an error listing, the programmer can obtain an assembly listing in any of several formats and a symbol table listing. In addition, some assemblers can provide a Cross Reference listing for all symbols used in the program.

Most assemblers produce an object program by making one or more passes over the source program (reading the original source code several times). The resultant object program is in relocatable binary format. That is, the first instruction appears to be located in the first word of processor memory. Since in most cases the program is not to be loaded into the bottom of memory, the object program must be linked to the proper memory addresses before it can be executed.

The linking program is provided as a standard program development utility with an operating system. Figure 3-1 illustrates the fundamental steps in producing an executable program from assembly source code.

#### Compilers

A compiler is a language processor written to translate a higher-level language whose structure, syntax and symbols are independent of any particular machine. The higher-level language operators most often do not correspond directly to binary instructions. It is the compiler's job to provide algorithms for their translation.

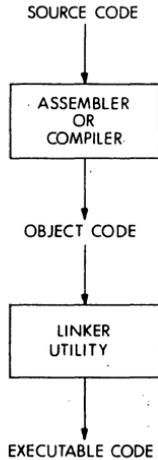


Figure 3-1 Fundamental Assembly or Compilation Procedure

Most compilers do not translate the source code until the entire source program is read at least once. The translation of the source code into object code takes place during several passes over the source code or, if only one pass over the original source code is made, during several phases of the compilation process. This allows the compiler to examine the code it produces as a whole to eliminate unnecessary instructions (code optimization). In addition, the compiler can perform many levels of error checking and it can produce several kinds of compilation listings, including source code listings, code generation listings, and diagnostics.

An incremental compiler (also called an interpreter) is a compiler that immediately translates source statements into an internal format. Each source statement is translated (and therefore can be executed) before the following statement is translated. Although this method of source translation does not enable possible object code optimization, it allows the compiler to provide program development services not possible in multi-pass or multi-phase compilers. For example, a syntax error detected in a source statement can be reported to the programmer immediately, and the programmer can correct the statement before proceeding.

One significant difference between a general compiler and an incremental compiler are the characteristics of the resulting object program. The object code produced by the former type of compiler requires a separate step of linking before it can be executed, as shown in Figure 3-1. This approach enables the programmer to combine several object programs into one executable program. This provides several advantages:

#### Modularity

A source program may be too large to be compiled successfully as a single unit but, if divided into modular sections, can be compiled as

several separate units. The separate sections can be combined at the object level to produce the resultant program. In addition, programs that are extremely complex can be divided into several sections so that they can be easily manipulated, debugged or modified. A change in one module of the program will only require recompilation of that section.

#### Assembly Language Routines

The compiler's object code can be combined with the object code produced by the operating system's assembler. Algorithms which are most easily written in assembly language, such as user-defined I/O processing, can be incorporated into a program written primarily in a higher-level language.

#### Library Routines

Libraries of commonly-used routines and functions written in either assembly or the higher-level language can be maintained in object format. These routines can be selectively included in the resultant program by the linking utility. This not only eliminates repetitive source coding and associated errors, it also decreases the size of the source and object programs.

The object code produced by an incremental compiler does not require an intermediate step of linking before it can be executed. The incremental compiler actually serves two purposes: it translates the source code into object code and it provides the environment in which to execute the object code. That is, the steps of source code translation, linking and execution are all provided by the translator. Figure 3-2 illustrates this type of translator operation.

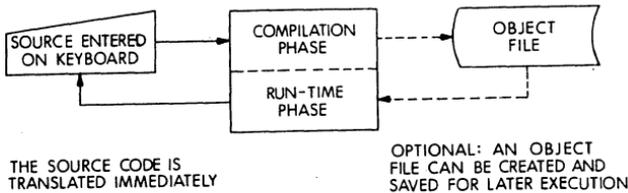


Figure 3-2 Fundamental Incremental Compiler Operation

#### Program Development Facilities

A complete language translation system requires facilities for creating and editing source programs, to link object programs into executable programs, and to debug programs. Most PDP-11 operating systems provide an editor utility for source program creation and editing, and a librarian utility for library file creation. Operating systems also provide a linker utility to link and combine object modules produced by multi-pass compilers and assemblers. Finally, operating systems also include debugging utilities.

Some of these facilities may or may not be incorporated into the language translator program itself. For example, an incremental compiler may include an editing facility as part of the language translation code.

This allows the programmer to edit the program interactively as it is being compiled and executed.

### **Libraries and Object Time Systems**

Also included in most language translation systems is a library of the most commonly-used functions and routines. The system library is generally a part of the language processor's Object Time System (OTS).

A multi-pass or multi-phase compiler does not usually generate all of the machine language code required by the program at run-time. Common sequences of code required by the program can be maintained in the OTS file. The compiler then flags the places where the desired sequences are needed. The linker utility, during its pass over the object program, selects those sequences from the OTS file and incorporates them into the executable program module.

An incremental compiler may also have an OTS. In this case, however, the OTS is generally part of the run-time code of the translator. When the object code is executed by the incremental compiler's run-time code, the OTS is used to provide common library code sequences.

## **3.2 PDP-11 ASSEMBLERS AND THE FORTRAN COMPILERS**

FORTRAN IV is available on the operating systems described in this handbook.

The MACRO assembler and FORTRAN IV compiler display the same external operating characteristics. In general, they accept source code from any valid input device and produce an object file on any valid file-structured device. If the input device is a file-structured device, the assembler or compiler can accept several source files. If desired, an assembly or compilation listing can also be produced as output, either as a file or on a line printer or terminal. MACRO can also generate a symbol table listing if requested. RT-11 MACRO can generate a Cross Reference Listing (CREF) if desired.

As shown in Figure 3-3, there are several methods for creating sources. A source program can be punched on cards if a card reader is available, or it can, in some cases, be entered directly on the terminal. The common method is to create a file on a file-structured device. The file can be created from paper tape, using the PIP file transfer utility to copy it onto disk. The file can also be created on a terminal, using the operating system's editor utility to store it on disk.

In addition to source program files, the MACRO assembler accepts source library files as input. The operating system provides a system library for MACRO containing the macro definitions for the system's monitor calls or executive directives. The assembler selects those macro definitions required by the source program from the system library file.

In RSX-11S systems, the MACRO assembler can also accept a user-created macro library as input. The sources for the user-defined macro libraries are created in the same manner as normal source programs.

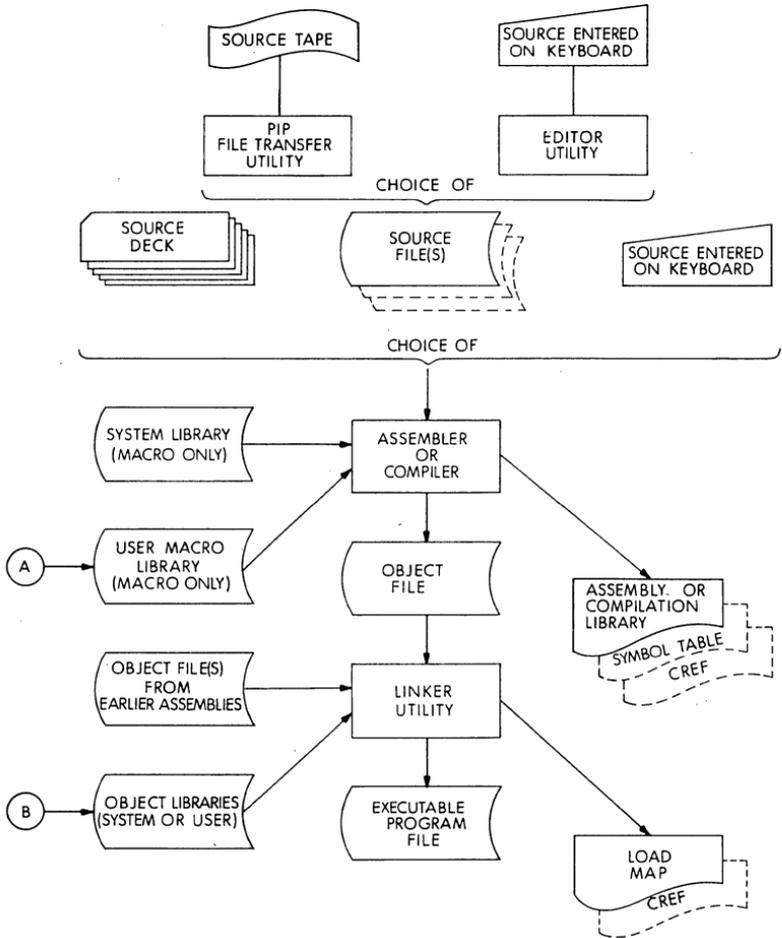


Figure 3-3 Building an Executable User Program Written in MACRO or FORTRAN

The operating system's librarian utility program is used to create the library files. Figure 3-4 illustrates this procedure.

Once the assembler or compiler produces an object file, the object file can be linked by the linker utility. The linker can accept several object files as input. In addition, when linking object files produced a FORTRAN compiler, the linker accepts the FORTRAN system object library for the given compiler as input. The linker automatically selects the required routines from the library.

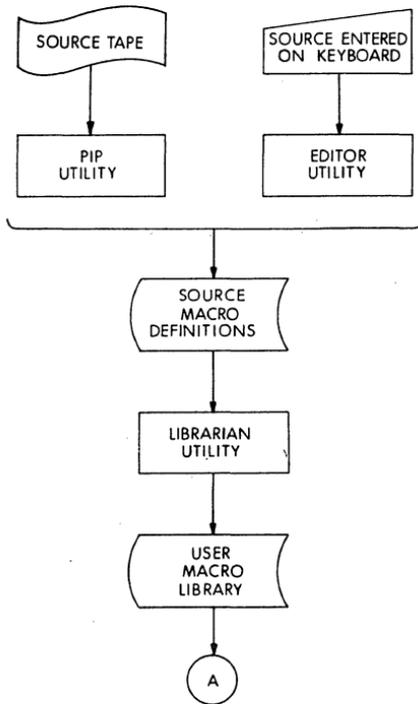


Figure 3-4 Building User Macro Libraries

Users can also create their own object library files. The source code is created in the same manner as normal source programs. The librarian utility is used to build the library file. Figure 3-5 illustrates the procedure.

PDP-11 assemblers and compilers differ in their internal operation, as shown in Figure 3-6. The MACRO assembler is a two-pass assembler. It makes a first pass over the source input to collect the symbol references, expand macros and produce preliminary object code. A second pass is made to resolve symbol references and produce the completed object code and listings.

The FORTRAN IV compiler is a multiple-phase compiler. Instead of making multiple passes over the source program, it reads the source program once and manipulates the source code in memory. The compiler operates in multiple phases. An overlay is read into memory for each phase of the compilation process. This method enables the compiler to compile relatively large programs very quickly.

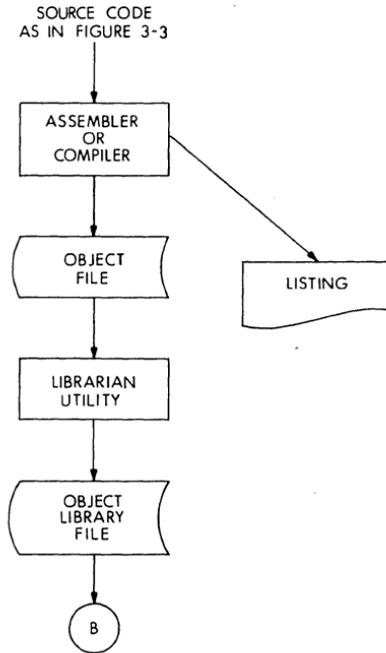


Figure 3-5 Building User Object Libraries from Sources Written in MACRO or FORTRAN

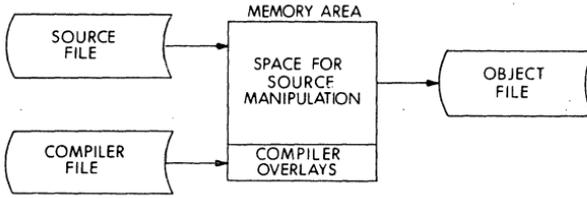
### 3.3 INCREMENTAL COMPILERS

The BASIC language processed by incremental compiler are:

Two BASIC language compilers are available: single-user BASIC and multi-user BASIC. Both are available for use on RT-11 operating systems.

The BASIC language processors can accept source input from a terminal or from a file generated using an editor utility, as illustrated in Figure 3.7. The most common method of creating a source program is by giving the source statements directly to the compiler through an interactive terminal. For this reason, the BASIC language processors include an editing facility. This allows the programmer to create, test, and modify the source program interactively.

MULTI-PHASE COMPILATION



MULTI-PASS COMPILATION

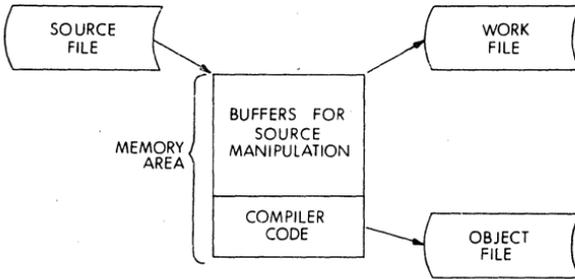


Figure 3-6 Compilation Methods

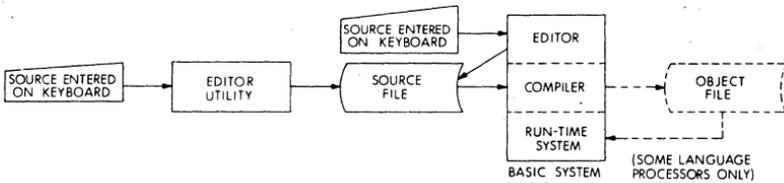


Figure 3-7 BASIC Language Processor Operation

## FOREGROUND/BACKGROUND OPERATING SYSTEM RT-11

### 4.1 OPERATING SYSTEM FUNCTIONS AND FEATURES

RT-11 is a single-user disk operating system designed for interactive program development and on-line applications. In a foreground/background environment, the user can run a real-time application in the foreground while interactively developing a program or running a batch stream in the background. RT-11 includes two compatible monitors and a variety of program development and system utilities. As options, RT-11 supports FOCAL, FORTRAN IV and BASIC languages. BASIC is offered in a single-user or multi-user version.

RT-11 runs on LSI-11, PDP-11/03 and PDP-11V03 systems with between 8K and 28K words of memory. In addition, RT-11 requires a console terminal and RXV11 floppy disk drive. RT-11 also supports a high-speed printer (LAV11).

RT-11 is a particularly easy-to-use system, and yet it provides a sophisticated set of programming tools, particularly for the lab application environment. The system is completely device-independent and configuration-independent. Programs can be directed to use a specific peripheral at run-time. The system can run unmodified on a variety of memory configurations.

The I/O system has been designed so that device handlers are files on the system device. Users can incorporate device drivers for special devices through simple file-oriented interfaces.

**Table 4-1 RT-11 System Summary**

|                   |   |
|-------------------|---|
| System type       | Single-user, real-time application foreground, program development or batch job background.   |
| Memory size range | MINIMUM: 8K words for single job monitor<br>(12K with BATCH)<br>16K words for Foreground/<br>Background Monitor<br>MAXIMUM: 28K words |

**Table 4-1 RT-11 System Summary (Cont.)**

|  |  |
|--|--|
| Additional CPU hardware supported                        | KEV11 Extended Instruction Set*,<br>Floating Instruction Set*<br>* Under BASIC or FORTRAN and FOCAL  |
| Minimum peripherals                                      | Console terminal; RXV11 floppy disk drive.   |
| Additional peripherals                                   | LAV11 high-speed printer   |
| Monitor size   | Single Job Monitor: 2K words<br>Foreground/Background Monitor: 4K words  |
| Maximum space available for programs and device handlers | Under Single Job Monitor: 26K words<br>Under Foreground/Background Monitor 24K words   |
| System programs  | EDIT text editor<br>MACRO assembler (requires 12K system)<br>EXPAND macro expander<br>ASEMBL assembler<br>LINK linker<br>LIBR librarian<br>ODT on-line debugger<br>PATCH code patch utility<br>PATCHO object code patch utility<br>PIP peripheral interchange program<br>FILEX file exchange utility<br>SRCCOM text comparison utility<br>DUMP file dump utility<br>LAB APPLICATION-11 library (optional)<br>BATCH batch processor |
| High-level languages                                     | FOCAL<br>Single-user BASIC or multi-user MU BASIC<br>FORTRAN IV  |

## 4.2 MONITOR ORGANIZATION

The RT-11 operating system actually provides two monitors: The single job (SJ) Monitor and the Foreground/Background (F/B) Monitor. The F/B Monitor allows two programs to operate: A foreground program and a background program. The real-time or on-line function is accomplished in the foreground which has priority on system resources. Functions which do not have critical response-time requirements (e.g., program development or batch operations) are accomplished in the background, which operates whenever the foreground is not busy. Both foreground and background have access to all the monitor services. The language processors and utilities are confined to the background. Although they operate independently, Foreground and Background can communicate through disk files and/or system message queuing facilities. If F/B operation is not required, the SJ Monitor—a totally compatible subset, which requires less memory and less overhead—can be booted from the system device instead of the F/B Monitor.

### 4.2.1 Monitor Components

To keep system overhead low, the RT-11 Monitor is designed in a modular fashion so that only those portions actually in use at a given time are resident in memory. The Monitor is made up of four major modules plus the device handlers available under the system. As shown in Figure 4-1, only the resident monitor (RMON) is permanently in memory. The keyboard monitor (KMON), the user service routine (USR), the command string interpreter (CSI), and the device handlers are loaded into memory only as needed.

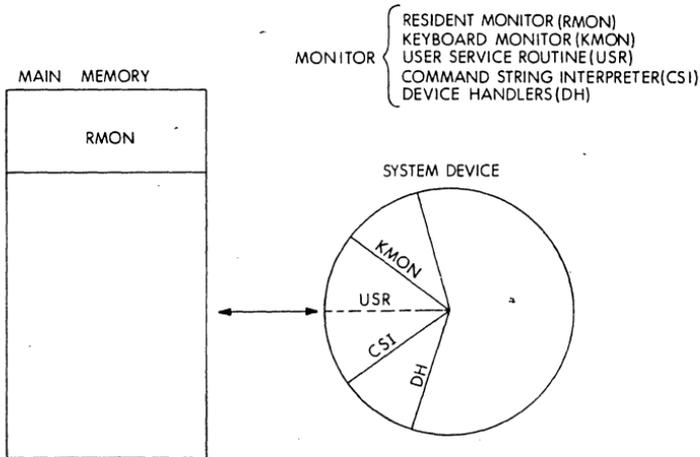


Figure 4-1 RT-11 MONITOR

#### Resident Monitor (RMON)

The resident monitor is the only permanently memory-resident part of RT-11. The programmed requests for all services of RT-11 are handled by RMON. RMON also contains the console terminal service, error processor, system device handler, and system tables.

#### Keyboard Monitor (KMON)

The keyboard monitor provides communication between the user at the console and the RT-11 system. Monitor commands allows the user to assign logical names to devices; run programs, load device handlers, and control F/B operations. A dot at the left margin of the console terminal page indicates that the Keyboard Monitor is in memory and is waiting for a user command.

#### User Service Routine (USR)

The User Service Routine provides support for the RT-11 file structure. It loads device handlers, opens files for read or write operations, deletes and renames files, and creates new files. The Command String Interpreter (CSI) is part of the USR and can be accessed by any program to interpret a character string as a command to perform a set of USR operations.

### Device Handlers (DH)

As shown below, RT-11 device handlers are, for the most part, short, easily implemented routines. The device handlers are merely files on the system device. This means they may be created by the normal editing and assembling process. Interfacing them to the monitor then requires only the modification of five to six parameters in the monitor. Only the device handlers with an asterisk are presently usable with LSI-11, PDP-11/03, and PDP11V03 peripherals.

#### HANDLER

|                                       |      |
|---------------------------------------|------|
| Cartridge Disk (RK11)                 | 113  |
| DECtape (TC11)                        | 106  |
| Fixed Head Disk (RF11)                | 92   |
| Cassette (TA11)                       | 996  |
| Magtape (TM11)                        | 1120 |
| *Line Printer (LP11) (use with LAV11) | 99   |
| Card Reader (CR11)                    | 363  |
| *Console Terminal (LA36)              | 140  |
| *Paper Tape Reader                    | 62   |
| *Paper Tape Punch                     | 56   |
| *Floppy Disk (RX11) (use with RXV11)  | 216  |
| Disk Pack (RP02)                      | 142  |
| Fixed Head Disk (RS03/4)              | 78   |

#### 4.2.2 General Memory Layout and Component Sizes

When the RT-11 system is first bootstrapped from the system device, memory is arranged as shown in the left diagram of Figure 4-2 (this is the case for either the Single-Job or Foreground/Background Monitor, since no foreground job exists yet). Under the F/B Monitor the background job is the RT-11 module KMON.

When device handlers are loaded into memory, USR and KMON are moved down, as shown in the center diagram of Figure 4-2.

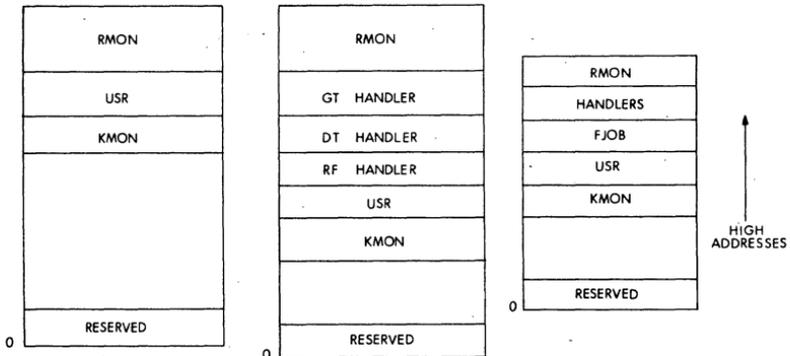


Figure 4-2 RT-11 System Memory Maps

RT-11 maintains a free memory list to manage memory. Thus, when a handler is unloaded, the space the handler occupied is returned to the free memory list and is reclaimed by the background.

When an RT-11 foreground job is initiated, room is created for the foreground job to be loaded by decreasing the amount of space available to the background job, as shown in the right diagram of Figure 4-2.

Following are the approximate sizes (in decimal number of words) of the components for RT-11.

|      | <b>F/B</b> | <b>Single-job</b> |
|------|------------|-------------------|
| RMON | 4000       | 2000              |
| USR  | 2050       | 2050              |
| KMON | 1550       | 1550              |

In the F/B system, the background area must always be large enough to hold KMON and USR (3.5K words). The following list indicates the total space available for the loaded device handlers, the foreground job, and the display handler. Note that the low memory area from 0-477 is never used for executable programs. (These sizes also allow room for the 4K RMON).

| <b>MACHINE SIZE (WORDS)</b> | <b>FOREGROUND SPACE AVAILABLE (WORDS)</b> |
|-----------------------------|---|
| 16K                         | 8.5K                                      |
| 24K                         | 16.5K                                     |
| 28K                         | 20.5K                                     |

With the Single-Job Monitor, RMON requires only 2K. The following list shows the amount of space available to users with the Single-Job Monitor:

| <b>MACHINE SIZE (WORDS)</b> | <b>PROGRAM SPACE AVAILABLE (WORDS)</b> |
|-----------------------------|--|
| 8K                          | 6K                                     |
| 16K                         | 14K                                    |
| 24K                         | 22K                                    |
| 28K                         | 26K                                    |

When a background job or single-job program is initiated, RT-11 allows the program to be loaded in over the KMON and USR if it exceeds the free memory available. While the program is running, the KMON and USR are not resident. If the program issues a request that the USR must service, a portion of the program is swapped out to make room for the USR. The USR is swapped in and remains resident until the request is serviced. When the request is serviced, the portion of the program that was previously swapped out is swapped back in.

If the program terminates or if an operator types a CTRL/C to interrupt the program, both the KMON and USR are loaded into memory. That portion of the program that occupies the KMON/USR space is swapped out; it is held until the operator either continues the program or re-initializes the environment.

If a background or single-job program does not exceed the free memory available when it is loaded, the KMON and USR remain resident during its execution and are not reloaded when the program terminates. Swapping, if it does occur, is invisible to the user.

Typical memory layouts for both the Single-Job and Foreground/Background systems are shown in Figure 4-3.

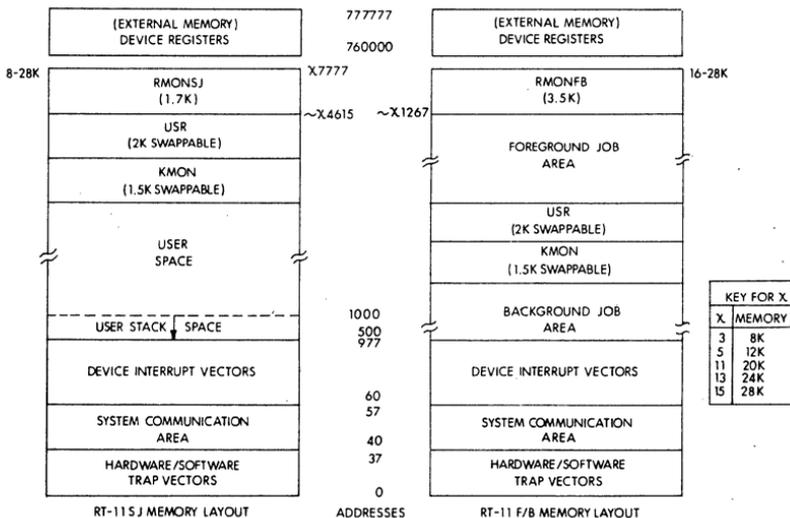


Figure 4-3 Typical Memory Layouts

### 4.2.3 I/O System Design and Operation

All I/O is handled in block format directly to and from the user area. There is no intermediate buffering by the monitor to slow down throughput. Since most programs deal with data on a character-by-character basis, it was deemed superfluous to provide line-oriented handling in the monitor.

All I/O is processed by a queue manager. RMON sends I/O orders to the queue manager which directs the device handlers to carry out the order. If the device handler is busy, the order waits until the device is available and the order can be completed. If parallel processing is indicated, the order is placed in the I/O queue for later processing and control is returned to the requesting program. There is one queue for each device. In F/B operation, the foreground request is always given the highest priority because foreground requests are queued ahead of background requests as directed by the initial read/write order. The queue manager may initiate a completion routine when the device handler indicates completion of the I/O order.

There are three modes of I/O operation:

#### **SYNCHRONOUS I/O**

In this mode, the I/O transfer is initiated and control does not return to the requesting program until the transfer is complete or an error is detected. This allows processing to proceed knowing that the data transfer has been completed.

#### **ASYNCHRONOUS I/O**

In this mode, the transfer is started (the system will place the request in a queue automatically if the device is already in use) and control is returned immediately to the requesting program so that processing may continue. When the user program needs assurance that the I/O operation has been completed, it issues a WAIT request which returns control only when the requested operation has terminated. This option allows overlap of an I/O process with program execution and with other I/O processes. The processes are usually asynchronous, but can be synchronized when necessary.

#### **EVENT-DRIVEN I/O**

This mode utilizes the full flexibility of the PDP-11 and RT-11: The I/O transfer is initiated (automatically queued if the device is already in use) and control is returned to the requesting program. When the I/O operation has been completed, the user program is interrupted and control is passed to a "completion routine" (specified when the transfer was originally requested). When the completion routine exits, control is returned to the interrupted program at the point of interrupt. The completion function allows fully overlapped, asynchronous I/O, greatly simplifying the real-time task.

#### **4.2.4 Batch Processing**

RT-11 BATCH is a complete job control subsystem which provides batch mode processing of user jobs. In the foreground/background environment, BATCH processes job streams in the background partition, allowing real-time or other user jobs to run in the foreground. The BATCH run-time support package (which is resident when BATCH is running) requires only 1K words of memory. RT-11 BATCH may be used in either single-job monitor configurations of 12K or more of memory, or in any foreground/background configuration.

RT-11 BATCH also has the unique capability of being interrupted. For example, a three-hour batch stream might be half done when an urgent need for a specific job arises. A simple command may be typed to RT-11, causing BATCH to give control to the keyboard monitor at a convenient breaking point. Commands may be entered and jobs may be run (even a batch job), after which a command may be given to continue running the interrupted batch stream to its normal completion.

The BATCH job control language is discussed in paragraph 4.4.3.

#### **4.2.5 Switching Between Single-Job and Foreground/Background**

The RT-11 user can use either the Single-Job or the Foreground/Background Monitor; the process of switching between the two requires no special programming considerations or elaborate operator procedures. For an application that requires frequent switching between the F/B and Single-Job monitors, the basic procedure is:

1. Store both monitors on the same device, one with the active monitor name, the other with an inactive name.
2. When the switch is desired, the operator runs the PIP system utility, preserves the running monitor by renaming it to an inactive name and renames the desired monitor to the active monitor name. The operator then uses two PIP commands to copy the bootstrap and reboot the system.

### 4.3 SYSTEM CONVENTIONS

The following sections describe the RT-11 system conventions for devices and device names, filenames and extensions, data formats and file structures.

#### 4.3.1 Physical Device Names

Devices are referenced by means of a standard two-character device name. Devices can also be assigned logical names. A logical name takes precedence over a physical name and thus provides device independence. With this feature a program that is coded to use a specific device does not need to be rewritten if the device is unavailable.

#### 4.3.2 Filenames and Extensions

Files are referred symbolically by a name of one to six alphanumeric characters followed, optionally, by a period and an extension of up to three alphanumeric characters. (Excess characters in a filename may cause an error message.) The extension to a filename generally indicates the format of a file. If an extension is not specified for an input or output file, most system programs assign appropriate default extensions. Table 4-2 lists the standard extensions used in RT-11.

**Table 4-2 Filename Extensions**

| <b>EXTENSION</b> | <b>MEANING</b>  |
|------------------|---|
| .BAD             | Files with bad (unreadable) blocks; this extension can be assigned by the user whenever bad areas occur on a device. The .BAD extension makes the file permanent in that area, preventing other files from using it and consequently becoming unreadable. |
| .BAK             | Editor backup file.   |
| .BAS             | BASIC source file (BASIC input).  |
| .BAT             | BATCH command file.   |
| .CTL             | BATCH control file generated by the BATCH compiler.   |
| .CTT             | BATCH internal temporary file.  |
| .DAT             | BASIC or FORTRAN data file.   |
| .DIR             | Directory listing file.   |
| .DMP             | DUMP output file.   |
| .FOR             | FORTRAN IV source file (FORTRAN input).   |
| .LDA             | Absolute binary file (optional Linker output).  |
| .LLD             | Library listing file.   |
| .LOG             | BATCH Log file.   |

**Table 4-2 Filename Extension (Cont.)**

|      |   |
|------|---|
| .LST | Listing file (MACRO or FORTRAN output).   |
| .MAC | MACRO or EXPAND source file (MACRO, EXPAND, SRCCOM input).  |
| .MAP | Map file (Linker output).   |
| .OBJ | Relocatable binary file (MACRO, ASEMBL, FORTRAN IV output, Linker input, LIBR input and output).                          |
| .PAL | Output file of EXPAND (the MACRO expander program), input file of ASEMBL.   |
| .REL | Foreground job relocatable image (Linker output, default for monitor FRUN Command).                                       |
| .SAV | Memory image or SAVE file; default for R, RUN, SAVE and GET Keyboard Monitor commands; also default for output of Linker. |
| .SOU | Temporary source file generated by BATCH.   |
| .SYS | System files and handlers.  |

### **4.3.3 Data Formats**

The RT-11 system makes use of five types of data formats: ASCII, object, memory image, relocatable image, and load image.

Files in ASCII format conform to the American National Standard Code for Information Interchange, in which each character is represented by a 7-bit code. Files in ASCII format include program source files created by the Editor, listing and map files created by various system programs, and data files consisting of alphanumeric characters.

Files in object format consist of data and PDP-11 machine language code. Object files are those output by the assembler or FORTRAN compiler and used as input to the Linker.

The Linker can output files in memory image format (.SAV), relocatable image format (.REL), or load image format (.LDA).

A memory image file (.SAV) is a "picture" of what memory will look like when a program is loaded. The file itself requires the same number of disk blocks as the corresponding number of 256-word memory blocks.

A relocatable image file (.REL) is one which can be run in the foreground. It differs from a memory image file in that the file is linked as though its bottom address were 0. When the program is called (using the monitor FRUN command), the file is relocated as it is loaded into memory. (A memory image file requires no such relocation.)

A load image (or .LDA) file may be produced for compatibility with the PDP-11 Paper Tape System and is loaded by the absolute binary loader. .LDA files can be loaded and executed in stand-alone environments without relocation.

### **4.3.4 File Structure**

RT-11 uses a "contiguous" file structure. This type of structure implies

that every file on the device is made up of a contiguous group of physical blocks. Thus, a file that is 9 blocks long occupies 9 contiguous blocks on the device. This file structure minimizes the number of disk accesses needed to transfer data and hence increases access speed.

A contiguous area on a device can be in one of the following categories:

1. Permanent file. This is a file which has been closed on a device. Any named files which appear in a PIP directory listing are permanent files.
2. Tentative file. Any file which has been created but not closed is a tentative file entry. When the file is closed, the tentative entry becomes a permanent file. If a permanent file already exists under the same name, the old file is deleted. If a file is never closed, the tentative file is treated like an empty entry.
3. Empty entry. When disk space is unused or a permanent file is deleted, an empty entry is created. Empty entries appear in a complete device directory listing as <UNUSED> N, where N is the decimal length of the empty area.

Since a contiguous structure does not automatically reclaim unused disk space, RT-11 PIP has an option which allows the user to collect all empty areas so that they occur at the end of a device.

#### **4.4 COMMANDS**

The operator controls and directs system operation through three different interfaces. The operator communicates directly with the Monitor using keyboard commands and special function keys, communicates indirectly with the Monitor or user programs by issuing an I/O command string. The following sections describe the commands and command formats.

##### **4.4.1 Keyboard Communication**

Keyboard commands and special function keys allow the operator to communicate with the RT-11 Monitor to allocate system resources, manipulate memory images, start programs, and use foreground/background services. The keyboard commands are interpreted by KMON. Note that in a Foreground/Background environment, KMON is always run as the background job. The special function keys are interpreted by the console terminal device handler. They can be used to communicate with any system or user program through the console terminal.

Table 4-3 lists the Monitor keyboard commands. The first set of commands are general commands used in either a Single-Job or Foreground/Background environment. The second set of commands are used in either environment to control single-job or background program operation. The last set of commands are only in a Foreground/Background system to control foreground program operation.

The special function of certain terminal keys used for communication with KMON are listed in Table 4-4. It is important to note that console input and output under F/B are independent functions; input can be typed to one job while output is printed by another. The operator may be in the process of typing input to one job when the other job is ready to print on the terminal, in which case the job which has output inter-

rupts the operator and prints the output on the terminal. Input control is not redirected to this job, however, unless a CTRL/B or CTRL/F is explicitly issued. If input is typed to one job while the other has output control, echo of the input is suppressed until the job accepting input regains output control. At this point, all accumulated input is echoed. If the foreground job and background job are both ready to print output at the same time, the foreground job has priority.

**Table 4-3 Monitor Keyboard Commands**

**General Commands**

|         |  |
|---------|--|
| DATE    | Enters or reports the system date. The specified date is assigned to newly-created files, new device directory entries, and listing output until a new DATE command is issued.   |
| TIME    | Resets or reports the current time of day kept by the system.  |
| ASSIGN  | Assigns or deassigns a user-specified 1-3 character logical name to a physical device. This is useful, for example, to redirect I/O to another device when a program to be executed refers to a device that is not available. It can also be used to assign FORTRAN logical units to device names.   |
| SET     | Used to change device handler characteristics and certain system configuration parameters. The user can set characteristics such as line printer or console line width, "device not ready" error handling, cassette read verification, card reader card codes, etc.  |
| LOAD    | Makes a device handler memory-resident for foreground and/or background jobs. Execution is faster when a handler is resident but memory area must be allocated for the handler. Any device handler to be used by a foreground job must be loaded before it can be used. A foreground or background job can own a device exclusively, or the jobs can share a device.   |
| UNLOAD  | Makes handlers that were previously loaded non-resident, freeing the memory they were using. UNLOAD clears ownership for all units of an indicated device type. Any memory freed is returned to a free memory list and eventually reclaimed for the background job. A special function of this command is to remove a terminated foreground job and reclaim memory, since the space occupied by the foreground job is not automatically returned to the free memory list when it finishes. |
| BASE    | Sets a relocation base which is added to the address specified in subsequent Examine or Deposit commands to obtain the address of the location to be referenced. This command is useful when referencing linked modules; the base address can be set to the address where the module is loaded.  |
| EXAMINE | Prints the contents of a specified location(s) in octal on the console.  |
| DEPOSIT | Deposits a specified value at a given location.  |

**Table 4-3 Monitor Keyboard Commands (Cont.)**

|       |   |
|-------|---|
| SAVE  | Writes a specified user memory area to a named file and device in save image format. This command allows the user to modify a background or single job program using the Examine and Deposit commands after it has been loaded into memory, and then save the modified program. |
| CLOSE | Causes currently open output files in the background job to become permanent files. The command is used after a CTRL/C is issued to abort a background job and the user wants to preserve any new files that job had open prior to its termination.                             |

**Background or Single-Job Program Control Commands**

|            |   |
|------------|---|
| GET        | Loads a specified memory image file (background program) into memory. It is normally used to load a background program into memory for modification or debugging with ODT or the Base, Examine, Deposit and START commands. Multiple GET commands can be used to combine programs.  |
| START      | Begins execution of the program currently in memory (loaded using the GET command) at a specified address. START does not clear or reset memory areas.  |
| INITIALIZE | Resets background system tables and cleans up the background area. It makes non-resident any device handlers used by the background program that were not made permanently resident (using the LOAD command) and purges the background I/O channels. The command can be used prior to running a background program, or when the accumulated results of previously-issued GET commands are to be discarded.  |
| RUN        | Loads a specified memory image file (background program) into memory and starts execution at its start address. It is equivalent to a GET command followed by a START command with no start address specified.  |
| R          | The command is identical to the RUN command except that the file specified must reside on the system device.  |
| REENTER    | Starts a program at its reentry address (start address minus two). It is generally used to avoid reloading the same program for repetitive execution, or to restart a program interrupted by a CTRL/C.  |
| FRUN       | Used to initiate foreground jobs. Normally, FRUN loads the relocatable image file and starts it. The user can optionally specify additional space to be allocated for the job over and above the actual program size or specify additional stack space. In addition, the user can request that the program be loaded only. The user must then explicitly start the program using the RSUME command. If ODT is used with the foreground job, this feature allows the user to examine or modify the program before starting it. |

**Table 4-3 Monitor Keyboard Commands (Cont.)**

|         |  |
|---------|--|
| SUSPEND | Stops the execution of a foreground job. The user can either resume the job, run another foreground job, or unload the stopped foreground job to release the memory. |
| RSUME   | Resumes execution of the foreground job where it was suspended.  |

**Table 4-4 Special Function Commands**

|        |   |
|--------|---|
| CTRL/A | Display next page of output; used only after CTRL/S (used only with GT ON).   |
| CTRL/B | Under the F/B Monitor echoes B> on the terminal (unless output is already coming from the background job) and causes all keyboard input to be directed to the background job. At least one line of output will be taken from the background job (the foreground job has priority, and control will revert to it if it has output). All typed input will be directed to the background job until control is redirected to the foreground job (via CTRL/F).   |
| CTRL/C | Interrupts current program execution, and returns control to the Keyboard Monitor. Note that under the F/B Monitor, the job which is currently receiving input will be the job that is stopped (determined by whether a CTRL/F or CTRL/B was most recently typed). To ensure that the command is directed to the proper job, type CTRL/B or CTRL/F before typing CTRL/C. If a program is waiting for terminal input or is using the console terminal device handler for input, typing a single CTRL/C interrupts execution and returns control to the monitor command level; otherwise, two CTRL/C's must be typed in order to interrupt execution. |
| CTRL/E | Display all I/O on the screen and console terminal simultaneously (with GT ON command only).  |
| CTRL/F | Under the F/B Monitor echoes F> on the terminal and instructs that all keyboard input be directed to the foreground job and all output be taken from the foreground job. If no foreground job exists, F? is printed and control is directed to the background job. Otherwise, control remains with the foreground job until redirected to the background job (via CTRL/B) or until the foreground job terminates.   |
| CTRL/O | Inhibit printing on console terminal.   |
| CTRL/Q | Resume console output; used only after CTRL/S.  |
| CTRL/S | Temporarily suspended terminal output until CTRL/A or CTRL/Q.   |
| CTRL/U | Delete current line being entered.  |
| CTRL/X | Delete entire command string (EDIT only).   |
| CTRL/Z | End-of-file for terminal input.   |

#### 4.4.2 Entering I/O Information Using the CSI

Once either monitor has been loaded and a system program (or any program which uses the Command String Interpreter) has been started, the Command String Interpreter prints an asterisk at the left margin. In response to the asterisk, a command string is entered providing information about devices, filenames and extensions, and switch options. The general format of this command line is:

\*OUTPUT = INPUT/SWITCH

The = sign is a delimiter which separates the output and input fields; the < sign may be used in place of the = sign.

OUTPUT is entered in the format:

dev:filnam.ext[n], . . . dev:filnam.ext[n]

INPUT as:

dev:filnam.ext, . . . dev:filnam.ext

Command string switches vary with the system or user program.

#### 4.4.3 BATCH Job Control Language

The RT-11 BATCH job control language provides complete system control capabilities through easy-to-learn commands:

##### Job Control

|            |                                    |
|------------|------------------------------------|
| \$JOB      | Begin a job.                       |
| \$SEQUENCE | Assign an identification to a job. |
| \$EOJ      | End a job.                         |

##### Data Control

|        |  |
|--------|--|
| \$DATA | Delimit the beginning of data in the input stream. |
| \$EOD  | Signal end of data in input stream.                |

##### Operator Communications

|           |  |
|-----------|--|
| \$MESSAGE | Write a message to the console terminal. |
|-----------|--|

##### Device Associations

|            |   |
|------------|---|
| \$MOUNT    | Allows the operator to mount a requested volume and to associate the volume with a logical device name. |
| \$DISMOUNT | Disassociates a logic device name from a physical unit.   |

##### Functional Commands

|             |  |
|-------------|--|
| \$COPY      | Copies input files to output files.                      |
| \$CREATE    | Create a file with the data records in the input stream. |
| \$DELETE    | Delete specified files.                                  |
| \$DIRECTORY | Make directory listings.                                 |

|                            |   |
|----------------------------|---|
| <b>\$LINK</b>              | Create an executable program from object modules and libraries. |
| <b>\$PRINT</b>             | Print copies of specified files.                                |
| <b>\$RUN</b>               | Initiate execution of a specified program.                      |
| <b>Language Processors</b> |   |
| <b>\$BASIC</b>             | Execute BASIC.  |
| <b>\$FORTRAN</b>           | Compile FORTRAN source programs.                                |
| <b>\$MACRO</b>             | Assemble MACRO source programs.                                 |
| <b>Additional Commands</b> |   |
| <b>\$RT11</b>              | Accept RT-11 Keyboard Monitor commands directly.                |
| <b>\$LIBRARY</b>           | Create a library list to be used in Link operations.            |
| <b>\$CALL</b>              | CALL another BATCH command file as a subroutine.                |
| <b>\$CHAIN</b>             | Transfer control to another BATCH command file.                 |

All of the commands shown have several options each, allowing BATCH commands to specify many different operations. The following is an example of a runnable BATCH stream which will compile and execute a simple FORTRAN program:

```

$JOB
$FORTRAN/RUN
$EOJ

```

RT-11 BATCH also has a programmable mode. In this mode, BATCH supports such features as labels, variables, conditional branches, and substitute arguments.

#### **4.5 MONITOR PROGRAMMED REQUESTS**

The RT-11 monitor provides a complete set of services for user programs. Monitor services include calls used for program initialization, control of system operating characteristics, interrogation of system state and resources, command interpretation, file operations, I/O transfers, program termination, and interrupt servicing. Some features which are available only to the F/B user include:

1. Mark Time—This facility allows user programs to set clock timers to run for specified amounts of time. When the timer runs out, a routine specified by the user is entered. There may be as many mark time requests as desired, providing system queue space is reserved.
2. Timed Wait—This feature allows the user program to “sleep” until the specified time increment elapses. Typically, a program may need to sample data every few seconds or even minutes. While the program is idle, the other job can run. The timed wait accomplishes this; when the time has elapsed, the issuing job is again runnable.
3. Send Data/Receive—It is possible, under RT-11 F/B, to have the foreground and background programs communicate with one another.

This is accomplished with the send/receive data functions. Using this facility, one program sends messages (or data) in variable size blocks to the other job. This can be used, for example, to pass data from a foreground collection program directly to a background analysis program.

For users of FORTRAN, the system subroutine library (SYSLIB) provides a set of functions for alphanumeric string manipulation as well as most monitor functions.

Communication with the monitor is accomplished through the EMT instruction or FORTRAN CALLS to the system library. The low-order byte of the EMT instruction is used as an operation code in the range 360(8) through 375(8). Those operations which use codes 360(8) through 374(8) either require no arguments or use Register 0 to pass arguments to or return arguments from the monitor. Operations that require more than one argument use EMT 375, and R0 is used to point to an argument list in memory. The argument list includes an operation code, the channel for that operation, and any other arguments necessary.

For example, the macro call to read 1000(8) words into a buffer from block 7 of file 3

```
.READW #AREA, #3, #BUFFER#, #1000, #7
```

is expanded into code which points R0 to the argument block, fills the argument block with the specified arguments and executes the instruction EMT 375.

This structure is fully re-entrant, and allows users to construct programmed requests either statically or dynamically from parameters specified at run-time. The same routine can process many monitor services, simply loading R0 with argument pointers from appropriate queues. FORTRAN users can issue monitor calls through function calls to SYSLIB, the FORTRAN system interface library. These calls execute FORTRAN object time system (OTS) routines which actually issue the appropriate EMT instructions to be executed with the arguments specified. For example, the previous READ operation can be accomplished in a FORTRAN program via a CALL READW statement.

Most system resource control and interrogation is done through the programmed requests, but some communication is accomplished using two core areas, the system communication area and the monitor "fixed-offsets." The job communication area is located in locations 40(8)-57(8), and contains parameters which describe and control execution of the particular job running at the time (it is context-switched in F/B). Included in this area is the Job Status Word, starting address of the job, USR swapping address, and other parameters.

The second memory communication area is the "fixed-offset area," whose items are accessed by a fixed address offset from the start of the resident monitor. It contains system constants used to control monitor operation. The user program can interrogate these constants to determine characteristics of the operating environment while the job is running.

|       |                   |   |                        |
|-------|-------------------|---|------------------------|
| AREA: | READ<br>CODE      | 3 | ; FUNCTION AND CHANNEL |
|       | BUFFER            |   | ; BUFFER ADDRESS       |
|       | 1000 <sub>8</sub> |   | ; WORD COUNT           |
|       | 7                 |   | ; BLOCK #              |
|       | 0                 |   | ; USE WAIT MODE(READW) |

#### 4.5.1 Summary of Programmed Requests

There are three types of services which the monitor makes available to the user through programmed requests. These are:

1. Requests for File Manipulation.
2. Requests for Data Transfer.
3. Requests for Miscellaneous Services

Table 4-5 summarizes the programmed requests in each of these categories alphabetically. Those marked with an asterisk function only in a F/B environment; they are ignored under the Single-Job Monitor.

**Table 4-5 Summary of Programmed Requests**

##### File Manipulation Requests

| MNEMONIC    | PURPOSE  |
|-------------|--|
| *.CHCOPY    | Establishes a link and allows one job to access another job's channel.                           |
| .CLOSE      | Closes the specified channel.  |
| .DELETE     | Deletes the file from the specified device.  |
| .ENTER      | Creates a new file for output.   |
| .LOOKUP     | Opens an existing file for input and/or output via the specified channel.                        |
| .RENAME     | Changes the name of the indicated file to a new name.  |
| .REOPEN     | Restores the parameters stored via a SAVESTATUS request and reopens the channel for I/O.         |
| .SAVESTATUS | Saves the status parameters of an open file in user memory and frees the channel for future use. |

##### Data Transfer Requests

| MNEMONIC | PURPOSE  |
|----------|--|
| *.RCVD   | Receives data. Allows a job to read messages or data sent by another job in an F/B environment. The three modes correspond to the READ, .READC, and READW modes described below. |
| *.RCVDC  |  |
| *.RCVDW  |  |

**Table 4-5 Summary of Programmed Requests (Cont.)**

|         |  |
|---------|--|
| .READ   | Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. No special action is taken upon completion of I/O.  |
| .READC  | Transfers data via the specified channel to a memory buffer and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the read, control transfers asynchronously to the routine specified in the .READC request. |
| .READW  | Transfers data via the specified channel to a memory buffer and returns control to the user program only after the transfer is complete.   |
| *.SDAT  | Allows the user to send messages or data to the other job in an F/B environment. The three modes correspond to the .WRITE, .WRITEC, and WRITEW modes.  |
| *.SDATC |  |
| *.SDATW |  |
| .TTYIN  | Transfers one character from the keyboard buffer to R0.  |
| .TTINR  |  |
| .TTYOUT | Transfers one character from R0 to the terminal output buffer.   |
| .TTOUTR |  |
| .WRITE  | Transfers data via the specified channel to a device and returns control to the user program when the transfers request is entered in the I/O queue. No special action is taken upon completion of the I/O.  |
| .WRITC  | Transfers data via the specified channel to a device and returns control to the user program when the transfer request is entered in the I/O queue. Upon completion of the write, control transfers asynchronously to the routine specified in the .WRITC request.       |
| .WRITW  | Transfers data via the specified channel to a device and returns control to the user program only after the transfer is complete.  |

**Miscellaneous Requests**

| MNEMONIC | PURPOSE   |
|----------|---|
| .CDFN    | Defines additional channels for doing I/O.  |
| .CHAIN   | Chains to another program (in the background job only).                                 |
| *.CMKT   | Cancels an unexpired mark time request.   |
| *.CNTXSW | Requests that the indicated memory locations be part of the F/B context switch process. |
| .CSIGEN  | Calls the Command String Interpreter (CSI) in general mode.                             |
| .CSISPC  | Calls the CSI in special mode.  |
| *.CSTAT  | Returns the status of the channel indicated.  |
| .DATE    | Moves the current date information into R0.   |

**Table 4-5 Summary of Programmed Requests (Cont.)**

|           |  |
|-----------|--|
| *.DEVICE  | Allows user to turn off device interrupt enable in F/B upon program termination.   |
| .DSTATUS  | Returns the status of a particular device.   |
| .EXIT     | Exits the user program and returns control to the Keyboard Monitor.  |
| .FETCH    | Loads device handlers into memory.   |
| .GTIM     | Gets time of day.  |
| .GTJB     | Gets parameters of this job.   |
| .HERR     | Specifies termination of the job on fatal errors.  |
| .HRESET   | Terminates I/O transfers and does a .SRESET operations.  |
| .INTEN    | Notifies monitor that an interrupt has occurred and to switch to "system state," and sets the processor priority to the correct value.               |
| .LOCK     | Makes the monitor User Service Routines (USR) permanently resident until .EXIT or .UNLOCK is executed. The user program is swapped out if necessary. |
| *.MRKT    | Marks time; i.e., sets asynchronous routine to occur after a specified interval.   |
| *.MWAIT   | Waits for messages to be processed.  |
| .PRINT    | Outputs an ASCII string to the terminal.   |
| *.PROTECT | Requests that a vector or vectors in the area from 0-476 be given exclusively to this job.   |
| .PURGE    | Clears out a channel.  |
| .QSET     | Expands the size of the monitor I/O queue.   |
| .RCTRLO   | Enables output to the terminal.  |
| .REGDEF   | Defines the PDP-11 general registers.  |
| .RELEASES | Removes device handlers from memory.   |
| *.RSUM    | Causes the main line of the job to be resumed where it was suspended with .SPND.   |
| .SERR     | Inhibits most fatal errors from causing the job to be aborted.   |
| .SETTOP   | Specifies the highest memory location to be used by the user program.  |
| .SFPA     | Sets user interrupt for floating point processor exceptions.   |
| .SPFUN    | Performs special functions on magtape and cassette units.  |
| *.SPND    | Causes the running job to be suspended.  |
| .SRESET   | Resets all channels and releases the device handlers from memory.  |

**Table 4-5 Summary of Programmed Requests (Cont.)**

|         |  |
|---------|--|
| .SYNCH  | Causes the rest of the interrupt service routine to run as a completion routine.               |
| *.TLOCK | Indicates if the USR is currently being used by another job and performs a .LOCK if available. |
| .TRPSET | Sets a user intercept for traps to locations 4 to 10.  |
| *.TWAIT | Suspends the running job for a specified amount of time.                                       |
| .UNLOCK | Releases USR if a LOCK was done. The user program is swapped in if required.                   |
| .WAIT   | Waits for completion of all I/O on a specified channel.  |

#### **4.5.2 Program Environment Control**

A user program typically issues several programmed requests to control the operating environment in which it will run. These include memory use, I/O access, device control, and error processing.

The memory needs of a program are specified to the monitor by the .SETTOP request. When loaded, a program occupies the memory specified by its image created at link item. To obtain more memory, a .SETTOP request is executed, with R0 containing the highest address desired. The monitor returns the highest address available (resident handlers or foreground jobs may prevent all the memory desired from being available), and adjusts its core usage algorithms accordingly. If the memory requirements of the running program permit it, the monitor will retain the file system in resident memory (reducing swapping). If not, it will automatically swap the file system with part of the user program (invisible to that program). The .SETTOP request, then, is the main mechanism by which a user program can determine how much memory is available and can control monitor swapping characteristics.

If a program needs so much memory that the USR (file system) must swap, the swapping will automatically occur (invisible to the program) whenever a USR call is made. If a program knows what file operations are necessary, however, and these operations can be consolidated and performed in localized areas, the efficiency of the system can be enhanced in the following manner: request the USR to be swapped in, have it remain resident while a series of USR operations are performed one after the other, then swap the USR back out when the sequence is done. Three requests are provided for the control of USR swapping: LOCK causes the USR to be made resident for a series of file operations; UNLOCK causes it to swap again. F/B programs can use .TLOCK to make the USR resident, but only if the USR is not occupied servicing the other job's file requests. This check can prevent a job from becoming blocked while the USR, which is a serial, synchronous resource, is processing a request. When a .TLOCK succeeds, the USR is ready to perform an operation immediately.

I/O in RT-11 is accomplished via "channels." A "channel" is a logical link between a file or a device and a program doing I/O on that file or device. When a file is opened, a unique channel number is associated

with it, and all operations on that file are identified via the channel number. The only time a file or device name is used is when the file is opened. RT-11 provides 16(10) channels as part of the resident monitor; that is, up to 16(10) files can be active at any given time. More channels can be activated (up to 255(10)) with the .CDFN request. This request sets aside memory inside the job area to provide storage necessary to accommodate status information for the extra channels. Once the .CDFN request has been executed, as many channels as space allows can be active simultaneously.

The Foreground/Background monitor context-switches critical items such as the machine registers and the job status area. The job can add memory locations to the list of items to be context-switched with the CNTXW request. Typical use of CNTXW is to include a special user device or arithmetic unit in normal context switches.

Special devices can be stopped and interrupt-disabled should the job have to be unexpectedly aborted. The .DEVICE request allows the user to specify a set of device control register addresses and a mask of bits to be set in that register on job exit. When a job is terminated (either normally, by error condition, or operator command), the specified bits are set in the specified locations. The normal use of this feature is to disable any special devices that may be in use by the running program.

Special devices also require vector locations in the area 0-500(8). Since RT-11 normally loads the unused vector space as part of a program load, the user specifies the use of a vector with the .PROTECT request. This will cause RT-11 to leave this vector intact, and prevent other jobs from obtaining the vector at the same time.

During the course of program execution, errors can occur which normally indicate that a program cannot continue, and the job is stopped by the monitor. Typical examples of such errors include directory I/O errors, monitor I/O errors on the system device, or I/O requests to non-existent devices. Some programs cannot afford, however, to allow the monitor to abort the job because of such errors; a typical example is RT-11 multi-user BASIC, which cannot be aborted because of a directory I/O error affecting only one of its users. For such applications, a pair of requests is provided, .HERR and .SERR. A .HERR request (the normal default) indicates that severe errors are to be handled by the monitor and result in job abortion; a .SERR request causes the monitor to return most errors to the user program for appropriate action.

Each of the pending I/O, message, or timer requests must be queued in one of the monitor queues. Since queue length is variable from one job to another, memory for queue elements desired is set aside with the .QSET request. If no .QSET request is executed by the user program, the monitor uses queue elements set aside in the resident monitor. If only one element is available, all operations will be synchronous (any request issued when the queues are full automatically waits for an element to free up). To expand the size of the I/O queues, a .QSET is executed by the user program. The .QSET declares where in memory the additional queues will go and how many elements they are to contain.

Finally, in addition to using .HERR and .SERR to process I/O errors, the user program can specify that it wishes to handle traps to locations 4(8) and 10(8) (processor error traps). Normally these fatal errors are reported by the monitor and the job is aborted. A .TRPSET request, however, will specify the address of a user routine that is to be entered when a trap to 4(8) to 10(8) occurs. If this request is issued, the monitor takes no action on these errors: it merely calls the specified routines.

All the previous initialization and control requests allow the user complete control over the characteristics of his operating environment. The monitor offers a variety of services to make it easy to write simple programs; sophisticated programmers, however, have full control over all aspects of monitor operation, and can place the monitor in the subservient role an operating system should play.

#### **4.5.3 Resource and System Interrogation**

Many programs are statically written. The devices to be used are known at programming time and the code is written accordingly. Other programs depend on user commands to control their operation. They must interrogate the system to find out specific details about a device or file it may be using.

The date can be obtained with a .DATE request, and then printed on a report or entered as a data record in a file. The time-of-day can also be determined with a .GTIM request for much the same purposes. Information about whether the job is running in the foreground or background, as well as information on the memory limits of the job and the address of its channel areas, can be obtained with a .GTJB request.

The specifics of a given file (what block it starts at, its length, what device it is on) can be obtained with a .CSTATUS request, and the specifics of a device (what controller it is, whether or not it is file structured, its block length, etc.) can be obtained with a .DSTATUS request.

These requests are most often used by programs which alter their behavior if they are using a particular device or are running in the foreground.

#### **4.5.4 Command Interpretation**

Two of the most powerful requests are those for the system Command String Interpreter (CSI). The CSI, which is part of the USR, will process standard RT-11 command strings in the form

```
*DEV:OUTPUT/SWITCH = DEV:INPUT/SWITCH
```

These are the same command strings used by RT-11 system programs.

The most commonly used CSI request is .CSIGEN. When called via .CSIGEN, the CSI will obtain a command string from the user at the terminal (or process a string in memory if desired). The CSI will then analyze the string for correct syntax (if incorrect, an error will be reported to the user and another command will be automatically solicited). It will then load the specified device handlers into memory, open the specified files, and return to the calling programs with switch information and all I/O channels active and ready for I/O. This means that with

one simple macro call

```
..CSIGEN #DEFEXT, #DEVSPC, #0
```

a program can obtain a command, create or locate the name files, open the appropriate channels, and be ready to perform I/O without ever having to worry about file names, devices, or the like. Many RT-11 programs never directly open files or manipulate devices; a simple call to .CSIGEN sets up I/O and starts the program. With one macro call, a language processor such as FORTRAN is ready to do I/O from the source file and output to the listing and binary files. All user-specified switches are available to control the language processor's operation.

Some programs desire to do their own file and device manipulation, but do not wish to provide their own command processor. For these needs .CSISPC is provided. When .CSISPC is used, the CSI will obtain a command string, syntactically analyze it into tabular form, and pass the tables on to the user program for appropriate action. The calling program is saved the chore of analyzing the command, but retains complete control over any device or file activity. As an example of how powerful these CSI calls are, consider the following routine. It copies, in only nine statements, a user-specified file on any input device to a file on any output device. The file names and devices for input and output are specified by the user during program execution.

```
START: .CSIGEN #DSPACE, #DEFEXT, #0
LOOP:  .READW #AREA, #3, #BUFFER, #400, BLOCK
      BCS    DONE
      .WRITW #AREA, #0, #BUFFER, #400, BLOCK
      BCS    HERR
      INC    BLOCK
      BR     LOOP
DONE:  .EXIT
HERR:  .
```

#### 4.5.5 File Operations

The basis of all RT-11 I/O are the device handlers. Device handlers are the monitor's device-specific routines which actually receive an I/O command. The system device and console terminal handlers are part of the resident monitor and require no further attention. All other device handlers are loaded into the user area with a .FETCH request prior to any request which might access that device.

Once the handler has been loaded, existing files can be located and opened for access with a .LOOKUP request. New files are created with an .ENTER request, which allows allocation of a specific-size disk space for the file. The option also exists to request that the monitor automatically allocate as much space as possible. When file operations are completed, a .CLOSE is done to make a new file permanent in the directory, or a .PURGE may be performed to free the channel (close the file)

without involving any directory operations. Existing files can be renamed with a .RENAME request, or deleted with a .DELETE. Once the I/O on a device is finished, a .RELEAS command will remove the device handler from memory and free the space it occupied for other use. Two other requests add to the flexibility of file operations. Once an existing file has been opened with a .LOOKUP request, it can be made temporarily inactive with a .SAVESTATUS, which "remembers" the current status of the file and frees the channel for use with another file. When the time comes to access the file again, it can be re-activated on any free channel with a .REOPEN request, and I/O can continue on that channel. This feature is used for two purposes; it enables more files to be open than there are channels to keep them active, and it increases system swapping efficiency. The user keeps more files open than there are channels by performing a .SAVESTATUS on those which do not have to be active, shuffling files between active and inactive status as demand dictates. USR efficiency can be increased by locking the USR into core, opening all the files a job needs at once, .SAVESTATUSing them as they are opened, releasing the USR from core, and .REOPENing the files one at a time as they are needed. Because .REOPEN does not require any I/O, all USR swapping and directory motion for a job can be isolated in non-real-time initialization code, and many files can be efficiently manipulated at once.

#### 4.5.6 Input/Output

RT-11 I/O operates in three modes: synchronous, asynchronous, and event-driven or completion I/O. Synchronous I/O is the simplest and consists of .READW and .WRITW requests. These requests do not return control to the user program until the specified operation is complete. When a program resumes after one of these requests, it can process the buffer, as the specified operation has been completed. Asynchronous I/O is accomplished with .READ and .WRITE requests. These requests cause the I/O command to be queued. Control is returned immediately to the calling program. It is up to the program, then, to perform a .WAIT on that channel before operating on the buffer. It is asynchronous I/O which is most commonly used for double-buffering, with a typical algorithm of

```

START:      .READ          Buffer 1
LOOP:      .READ          Buffer 2
           .WAIT          Buffer 1
           process        Buffer 1
           .READ          Buffer 1
           .WAIT          Buffer 2
           process        Buffer 2
           GO TO LOOP

```

Event-driven or completion I/O is the mode most akin to the I/O structure of the processor itself. Initiated with a .READC or .WRITC, completion I/O requests specify an extra parameter: the address of a user-written service routine to be entered when the operation is complete. The request is queued and control returns immediately to the calling program. When the operation is completed, the user program is "interrupted," and the completion routine is entered. Completion routines

perform whatever operation is appropriate to the completion of an I/O request (they can even issue an I/O request of their own), and return to the monitor, which resumes the main program where it was left off.

Completion routines are one of the most powerful features of RT-11. They are most frequently used as "event detectors," which keep I/O going independent of main processing or respond with service to a specific external stimulus. They are a software analog to the hardware interrupt structure; they require no processor time until the completion occurs, at which point they become the highest priority code in the job.

Consider as an example of completion I/O a simple "spooler" which prints a disk file on the line printer (LAV11).

Main Routine

Read disk with completion routine DC (start spooling)

(processing)

Disk Completion Routine

DC: Write line printer with completion routine LP

RETURN

Line Printer Routine

LP: Read disk with completion routine DC

RETURN

The main program starts the processing by reading the first buffer load from the disk, then goes to a compute-bound job. When the disk-read completes, the disk completion routine is entered. A request is issued to print the buffer on the line printer, then the disk returns to the interrupted program. When the line printer-write completes, the line printer completion routine is entered and a request is issued for the next disk block to be read. This loop of completion routine issuing a request which results in activation of another completion routine continues until the entire file is printed. Meanwhile, the main program continues its processing, unaware of the interrupt-driven activity except that it has slightly less CPU time available.

RT-11 provides an additional I/O capability for the console terminal. Although buffers can be read or written to the terminal in the same manner as any other device, an alternative mode of I/O permits the terminal to provide character-by-character I/O more in keeping with the nature of the device itself. A .TTYIN request will obtain a character from

the console; a .TTYOUT will print one on the console. Whole lines can be output with a .PRINT request. Real-time programs can issue .TTINR and .TTOUR requests, which return an indication that a character is not available or the output buffer is full, rather than waiting for their availability. The program can then resume real-time operation and try again later. A .RCTRLO request will force the terminal output to be reactivated should the user have typed CTRL/O to suppress it, assuring that urgent messages will be printed. The console terminal handler that services these requests also makes all the special function commands available for input and output to user programs.

Finally, RT-11 provides .SPFUN, a request for performing special functions on unique devices such as magnetic tape. .SPFUN requests are passed to the handler, and are used for such things as rewind or space-forward operations on magnetic tape devices.

#### **4.5.7 Interjob Communications**

The foreground/background monitor provides a mechanism for sending and receiving messages that is analogous to normal I/O. The .SDAT and .RCVD requests also have three modes (synchronous, asynchronous, and event-driven) which allow transfer of buffers between the two jobs as if I/O were being done. The sending job treats .SDAT requests as if they were writes, and the receiving job views .RCVD as a read. Receiving jobs can be "activated" when messages are sent via .RCVDC completion routines, and sending jobs via .SDATC completion routines. .MWAIT is provided as a synchronization tool for message requests, similar to .WAIT for normal I/O.

It is a common practice for one job in F/B to wish to read or write data in a file opened by the other job (the background may be processing data collected by the foreground, for example). This is accomplished via the .CHCOPY request, which allows the user to obtain channel information from the other job and use that channel information to control a read or write request. Thus a background job processing foreground data might get a message from the foreground stating that the information is available on a given channel. It would then perform a .CHCOPY on that channel and read from the file which was opened by the foreground.

#### **4.5.8 Timer Support**

Completion routines are also the mechanism used to provide timer support in the F/B monitor. With the .MRKT request, the user specifies the address of a routine that is to be entered after a specified number of clock ticks. Similar to an I/O completion routine, .MRKT routines are asynchronous and independent of the main program. After the specified time elapses, the main program is interrupted, the timer completion routine executes, and when done, returns control to the interrupted program.

Pending .MRKT requests—as many as the queue can hold—are numbered for identification purposes. Pending timer requests can be cancelled with a .CMKT request. .MRKT requests are normally used as a scheduling tool; multi-function jobs can "schedule" subjobs on the basis of clock "events," detected by timer completion routines.

RT-11 allows a job to suspend itself for a specified time interval with a .WAIT request. .WAIT allows a compute-bound job to relinquish slices of time to the rest of the system, permitting other components to run.

#### **4.5.9 Program Termination or Suspension**

Many jobs come to an execution point where there is no further processing necessary until an external event occurs. In the F/B environment, such a job can issue a .SPND request, which suspends the execution of that job until it later executes a .RSUM request in a completion routine. While the foreground is suspended, the background will be running. When the desired external event occurs, it is often detected by a completion routine, which executes a .RSUM to continue the job where it was suspended.

When a job is ready to terminate or reaches a serious error condition, it can reset the system with .SRESET and .HRESET directives. .SRESET is a soft reset: the monitor data base is re-initialized, but queued I/O is allowed to run to completion. .HRESET is a hard reset: all I/O is stopped by a RESET instruction in the single job monitor or by calls to the handlers in F/B.

For actual termination, a job can return to the keyboard monitor with .EXIT, or initiate the execution of another program with a .CHAIN request. MACRO, for example, chains to CREF when finished to provide for the Cross REFERENCE listings. Files can remain open across a .CHAIN, and information is passed in memory to the chained job, so that it can adjust processing accordingly.

#### **4.5.10 Interrupt Service**

RT-11 does not require hardware for memory management or I/O management, so two program requests are used in interrupt service routines to provide necessary links to the monitor I/O system. .INTEN is the first command in every interrupt routine; it causes the system to use the system stack for interrupt service and allows the scheduler to make note of the interrupt. If device service is all the routine does, .INTEN is the only call it need make. If the interrupt routine is to do any other program requests, however (such as .READ or .WRITE), it must first force a context switch with a .SYNCH call. .SYNCH causes the remainder of the interrupt routine to be scheduled as a completion routine. When the .SYNCH is finished the completion routine can execute programmed requests, initiate I/O, resume the mainline code or schedule a subjob.

### **4.6 SYSTEM PROGRAMS**

As a comprehensive program development and operating system, RT-11 provides an impressive set of system programs to assist in user program development and debugging. These include a text editor, macro expander, language translators (MACRO, ASEMBL, FORTRAN, BASIC, FOCAL), linker, librarian, and debugging utilities (ODT, PATCH, PATCHO). In addition, the system programs include several file utility programs. This section describes the standard system programs. Paragraph 4.7 describes the languages.

#### 4.6.1 EDIT Interactive Editor

The RT-11 editor, EDIT, is an interactive, character-oriented text editor with all the advantages of character searches and string manipulation that a character-oriented editor offers.

EDIT has essentially three modes: character commands for character-level operations, line-oriented commands for those operations suited to lines, and scope mode, which allows users of VT11 displays (not available for use on LSI-11, PDP-11/03, and PDP-11V03 systems) to interactively edit text with the aid of the screen. Features of EDIT above and beyond normal text editors include iteration brackets, which allow repetition of portions of the command as often as desired, command macros, which allow specification of a series of operations to be used over and over again as a discrete command, and the ability to save and move blocks of text.

#### 4.6.2 LINK Linker

The outstanding feature of the RT-11 linker is its transparent overlay scheme. The linker provides standard capabilities such as automatic library searches, the ability to output core image, foreground-relocatable or absolute binary program files, and a variety of link-time library manipulation capabilities.

Overlays in RT-11, however, require no assembly language instructions or macro calls. The overlay structure of a program is specified at link time and can be altered simply by altering the linker command strings (no source changes involved). It is only necessary to follow a few simple conventions as routines are written; (the restrictions amount to good modular programming rules). The routines are then combined in any number of possible overlay structures.

For example, assume main program A, subroutines B and C which are called by A, and subroutines D and E, called by subroutine B.

The user can generate a program image which includes all five modules if abundant memory is available with the command:

```
*PROG = A,B,C,D,E
```

As memory gets more restricted, D and E could share an overlay region by relinking with the command string.

```
*PROG = A,B,C/C  
*D/O:1/C  
*E/O:1
```

Finally, the program could be made as small as possible by overlaying B and C as well as D and E with command strings

```
*PROG = A/C  
*B/O:1/C  
*C/O:1/C  
*D/O:2/C  
*E/O:2
```

#### 4.6.3 LIBR Librarian

The RT-11 Librarian (LIBR) allows the user to create, update, modify, list, and maintain library files.

LIBR provides the user with the capability of maintaining libraries composed of commonly-used functions and routines.

Each library is a file containing a library header, library directory (or entry point table), and one or more object modules. The object modules in a library file may be routines which are repeatedly used in a program, routines which are used by more than one program, or routines which are related and simply gathered together for ease in usage. The contents of the library file are determined by the user's needs. An example of a typical library file is the FORTRAN library provided with the FORTRAN package, which contains all the mathematical functions needed for normal usage.

A program requests a library module through a subprogram call to a global entry point. If this global entry point (subprogram name) is defined in another module passed to the Linker, the request is said to be satisfied. When a library is passed to the Linker, it is searched for entry points to match unsatisfied requests. The modules which satisfy requests are linked in; all other modules in the library are ignored.

#### **4.6.4 ODT On-Line Debugger**

RT-11 On-Line Debugging Technique (ODT) is a system program that aids in debugging assembled and linked object programs. From the keyboard, the user interacts with ODT and the object program to:

- Examine and modify the contents of any location in main memory, the registers, the processor status register, or ODT internal registers.
- Run all or any portion of an object program using the breakpoint feature.
- Search the object program for specific bit patterns.
- Search the object program for words which reference a specific word.
- Calculate offsets for relative addresses.
- Fill a single word, block of words, byte or block of bytes with a designated value.

The assembly listing of the program to be debugged should be readily available when ODT is being used. Minor corrections to the program can be made on-line during the debugging session, and the program may then be run under control of ODT to verify any changes made.

It is possible to use ODT to debug programs written as either background or foreground jobs. In the background or under the Single-Job Monitor, ODT can be linked with the program.

To debug a program in the foreground area, it is recommended that ODT be run in the background while the program to be debugged is in the foreground.

#### **4.6.5 PATCH Code Patch Utility**

The PATCH utility program is used to make code modifications to memory image (.SAV) files, including overlay-structured and monitor files. PATCH, like ODT, can be used to interrogate, and then to change, words or bytes in the file.

#### **4.6.6 PATCHO Object Patch Utility**

The RT-11 PATCHO program is used to correct and update object modules (files output by the assemblers or by the FORTRAN compiler). It is particularly useful when making corrections to routines that are in .OBJ format for which the source files are not available. PATCHO cannot be used to patch libraries built by LIBR, but it can be used to patch the .OBJ modules from which a library is built.

#### **4.6.7 PIP Peripheral Interchange Program Utility**

A very commonly used RT-11 system program is PIP, the file transfer and maintenance program. PIP is the "system utility"; besides providing file-oriented operations, PIP offers several functions which are used in system building or control. PIP functions include the ability to copy files individually or in groups; operations which can extend, delete, or rename files; commands which list device directories; commands to initialize devices with directories or system bootstraps; the ability to bootstrap any supported system device; and the ability to scan a disk for bad blocks. Finally, one of the more important functions provided by PIP is the ability to consolidate the free space on a device into one area, making more efficient use of RT-11's contiguous file structure.

#### **4.6.8 SRCCOM Source Compare Utility**

The RT-11 Source Compare program (SRCCOM) is used to compare two ASCII files and to output any differences to a specified output device. It is particularly useful when the two files are different versions of a single program, in which case SRCCOM prints all the editing changes which transpired between the two versions.

#### **4.6.9 FILEX File Exchange Utility**

FILEX is a general file transfer program used to convert files among file-formatted devices for various operating systems. Files are transferred as 16-bit binary data. No processing is done on the data itself except that which is necessary to convert between various data representations.

#### **4.6.10 DUMP File Dump Utility**

RT-11 DUMP prints the contents of all or any part of a file on the console or the line printer. DUMP can print the file contents in any one of four selected formats: octal words, octal bytes, ASCII characters, or RAD50 characters.

### **4.7 LANGUAGES**

RT-11 supports a wide variety of programming languages. The user can select any one of four languages to solve a particular application problem. Users can implement entire problem solutions using FORTRAN IV, BASIC or FOCAL, or can combine any of these languages with MACRO assembly language modules.

#### **4.7.1 MACRO Assembler**

The RT-11 MACRO assembler is a powerful, general-purpose macro assembler, the primary implementation tool for assembler language programs. MACRO is a two-pass assembler requiring an RT-11 system configuration (or background partition) of 12K or more words. Some notable features of MACRO are:

- program control of assembly functions
- device and filename specifications for input and output files
- error listing on command output device
- alphabetized, formatted symbol table listing
- relocatable object modules
- global symbols declaration for linking among object modules
- conditional assembly directives
- program sectioning directives
- instruction repetition directives
- macro definition directives for user-defined macros
- comprehensive set of system macros
- extensive listing control, including cross-reference listing; can be specified in the command string or in source program

MACRO also allows an ASCII file or macros, called a macro library, to be edited and used. Normally, DIGITAL's system macro library is used to support macro calls to the monitor; these can be conveniently combined with user-defined macro libraries with a simple editing operation.

#### **4.7.2 EXPAND Macro Expander and ASEMBL Assembler**

Because the MACRO assembler requires 12K words of memory to operate, a pair of programs is provided which gives a subset of macro capabilities to 8K users. EXPAND is a one-pass macro expansion utility that takes a macro source file as input, expands the macro calls into simple assembler source statements, and outputs a macro-free assembly language program as an ASCII file. This source file is then input to the two-pass assembler, ASEMBL, which runs in 8K. The effect is that EXPAND and ASEMBL combine to provide a considerable subset of MACRO's capability; enough of a subset to handle the system macro library and user macros which use a subset of the macro language.

#### **4.7.3 FORTRAN**

FORTRAN/RT-11 is an extended, optimizing FORTRAN IV system which operates on any RT-11 system. The FORTRAN IV compiler processes source programs extremely rapidly; typical 300-line programs compile in less than 25 seconds.

Extensive optimizations such as common subexpression elimination, array vectoring, and "peephole" local code sequence tailoring, decrease the size and increase the speed of object programs. The compiler produces the object code directly, without using temporary files, and does not require an intermediate assembly step, thus speeding program development time.

The FORTRAN IV system is designed to minimize the size of executable programs. The entire system (including the compiler and optimization capabilities) is completely functional in the minimum 8K RT-11 environment. The optional KEV11 EIS/FIS can be used to further improve system performance.

FORTRAN programs may be developed under RT-11 and output in absolute binary format for execution on a satellite machine with minimum peripherals. Only a device such as a paper tape reader is required for program loading.

Using SYSLIB, the RT-11 FORTRAN system subroutine library, all features and services of the RT-11 monitor are available to the FORTRAN programmer without the need for assembly language coding. FORTRAN programs may schedule subroutines to be executed when an external event occurs (the receipt of a message from the other job, the completion of an I/O transfer, the lapsing of a specified time interval); perform all types of monitor-level input/output and system informational calls; handle interrupts with a FORTRAN subprogram.

SYSLIB also contains extensive string manipulation routines in addition to routines for calling the monitor functions previously described. These routines create strings in LOGICAL\*1 arrays, and allows their manipulation.

|        |   |
|--------|---|
| CONCAT | to concatenate two strings together.  |
| GETSTR | to input a string.  |
| INDEX  | for locating substring X in string Y.                                       |
| INSERT | for inserting one string into another.                                      |
| LEN    | for determining the length of a string.                                     |
| PUTSTR | will output a string.   |
| REPEAT | will repeat strings.  |
| SCOMP  | for string comparison and sorting.  |
| SCOPY  | will copy a string.   |
| STRPAD | to pad a string with blanks to a specified length.                          |
| SUBSTR | extracts a substring from a larger string.                                  |
| TRANSL | will replace one string with another after directed character modification. |
| TRIM   | to remove trailing blanks.  |
| VERIFY | to test whether characters in one string appear in another.                 |

The real power of these functions is that they operate on variable-length strings. The strings can be manipulated fully without knowledge of their length, adding a new dimension to FORTRAN capabilities.

#### 4.7.4 FOCAL

FOCAL is an easy-to-learn interactive language. The RT-11 implementation is as an interpreter, which provides both stored program and immediate mode operations. Commands may be abbreviated to a single letter. Alphanumeric symbol names are provided, with up to six characters carried in the symbol table, only two of which are significant to FOCAL. FOCAL utilizes the same floating point package as does FORTRAN/RT-11, so all arithmetic options are supported; a double-precision version of FOCAL supports up to 17 digits of accuracy.

Any peripheral supported under RT-11 is available to the FOCAL user. The LIBRARY command allows the user to access any RT-11 file-structured device. Programs may be saved, loaded, started and/or CHAIN'd. Data may be save/accessed in sequential files and/or virtual files. (Files may be treated by the FOCAL user as a virtual array; data types include floating, double-precision floating, signed or unsigned integers, or byte data.)

Other features include scheduling up to eight asynchronous tasks from the clock; processing interrupts in the FOCAL language; user-controlled error processing, and the facility for one or more user-written assembly language functions.

#### 4.7.5 Single-User BASIC

BASIC/RT-11 is best suited for interactive applications. Machine resources (memory and execution time) are expended with resulting savings in programming time. BASIC/RT-11 has several features which add to its usefulness as a development tool. It is implemented as an incremental compiler; source statements are translated into a more compact, easily executed code and stored directly in memory. When the RUN command is issued, this easily interpreted internal language is executed as a program. When the LIST or SAVE commands are used, the internal form is translated back to the original ASCII for output. Like all Dartmouth-compatible BASIC implementations, statements are entered directly to BASIC one at a time (or the editor can be used), then executed as a program. Statements can also be executed immediately simply by typing commands without statement numbers. This immediate mode can be used for debugging, development, or even as a calculator. Immediate mode is a valuable debugging tool, because statements like

```
PRINT V1,Q
```

or

```
LET V1=3
```

allow the user to investigate the state of various program variables, change them, or insert statements in the middle of program execution.

BASIC/RT-11 supports two types of files, sequential and virtual. Sequential files are ASCII files of string or numeric data that are accessed sequentially. To get to item N, items 1 through N-1 have to be read first. Virtual files behave like standard arrays; string or numeric elements are accessed by subscript as if they were memory-resident. If the desired element is in the buffer, no I/O takes place; if not, the disk block containing the desired element is read, invisible to the program. Virtual files have a 256-word "cache;" access of elements near each other minimizes overhead, while I/O overhead increases as elements become more random.

Note that virtual files in immediate mode can be used to manipulate or "edit" a data base. If, for example, a file of 2000 data points is taken from A/D and is to be examined, commands like

```
OPEN "DATA" AS FILE VF1(2000)
```

and

```
PRINT VF1(137)
```

allow points to be selectively examined, and commands like

```
LET VF1(137)=0
```

are used to change them.

Beside numeric data, BASIC allows variable-length alphanumeric string variables. These string variables can also be used in both types of files.

For interface of user-written and special-purpose functions, BASIC/RT-11 supports the CALL statement. CALL allows the user to invoke a function by name and pass it any number of arguments.

In addition to these features, BASIC/RT-11 programs execute much faster than traditional interpreters, although programs written under FORTRAN execute considerably faster.

#### **4.7.6 Multi-User BASIC**

MU BASIC/RT-11 is an extended version of BASIC/RT-11 that allows an RT-11 system to support up to eight interactive BASIC users on single-job systems with 24K words of memory, or up to four users on single-job systems with 16K words of memory. Under the foreground/background monitor, up to four BASIC terminals can be supported on 28K word systems.

All terminal I/O is performed by the MU BASIC language processor. Users are optionally provided with accounts and passwords for file protection.

MU BASIC extends the BASIC/RT-11 language with additional statements as well as commands and functions. In particular, some additional statement features are:

|             |   |
|-------------|---|
| COMMON      | Allows a program to pass information to a chained program.  |
| PRINT-USING | Provides extensive output formatting capabilities, including the ability to print exponential numbers and dollar amounts with asterisk-fill protection. |
| ON-GOTO     | Transfers control to one of several lines of the program based on one or more conditions.   |
| ON-GOSUB    | Transfers control to a subroutine based on the condition specified in the statement.  |

# REAL-TIME MULTIPROGRAMMING RSX-11S

## 5.1 FUNCTIONS AND FEATURES

RSX-11S is a small execute-only operating system for dedicated application environments that can be run on LSI-11, PDP-11/03 and PDP-11V03 systems.

RSX-11S requires a host RSX-11M system for program development and system generation. Tasks can be written in MACRO or FORTRAN IV, assembled or compiled and subsequently linked on the host system, and then transported to an RSX-11S system for execution. The minimum RSX-11S system includes an Executive (with incorporated device drivers) and a special FCS that contains no support for file-structured devices. The user can also add a subset of RSX-11M's MCR services if the hardware configuration includes a terminal. If on-line task loading is desired, the user can include an On-line Task Loader (OTL) utility. If the user wants to save a system image for subsequent re-booting, the user can include the System Image Preservation (SIP) utility.

Since RSX-11S is a memory-only system, it does not support a file system, non-resident tasks, task checkpointing, dynamic memory allocation or program development. It does, however, support data storage on all devices supported by RSX-11M. Its purpose is to provide a run-time environment for the execution of tasks on a small system with a very modest complement of peripherals.

The minimum configuration for an RSX-11S system is a PDP-11 processor (including the LSI-11) with at least 8K words of memory and one of the following load devices: paper tape reader, paper tape reader/punch, or RXV11 floppy disk. At least 16K words are required for on-line task loading or the execution of tasks written in FORTRAN IV.

Table 5-1 summarizes the components of RSX-11.

## 5.2 RSX-11S OPERATING SYSTEM CONCEPTS

The RSX-11S operating system is designed to provide a resource-sharing environment ideal for multiple real-time activities. The basic facilities that RSX-11S provides for handling multiple requests for services while maintaining real-time response to each request are:

- multiprogramming
- priority scheduling
- contingency exist
- power-fail shutdown and auto-restart

The basic unit of work, which these operating system facilities service, is called the task. A task consists of one or more programs written in a source language such as MACRO or FORTRAN, assembled or compiled into an object format, and then built into a task image by the linker utility called the Task Builder. In addition to the normal linkage functions of combining object modules or creating overlays, the Task Builder sets up the basic task attributes that determine the task's resource requirements and relationship to other tasks in the system. The significant task

**Table 5-1 RSX-11S System Summary**

|                                   |   |
|-----------------------------------|---|
| System type                       | Execute-only real-time applications system; requires RSX-11M system for generation and program development  |
| CPU's supported                   | LSI-11 processors (including PDP-11/03 and PDP-11V03)<br>PDP-11/04, 05, 10<br>PDP-11/35, 40, 45<br>PDP-11/70  |
| Memory ranges                     | Minimums:<br>8K words without on-line task loading option<br>16K words for one-line task loading or execution of tasks written in FORTRAN<br>Maximum:<br>124K words on all but PDP-11/70, LSI-11, PDP-11/03, and PDP-11V03<br>1024K words on a PDP-11/70<br>28K words on LSI-11, PDP-11/03, and PDP-11V03 |
| Additional hardware supported CPU | Same as RSX-11M   |
| Minimum peripherals               | One of the following load devices:<br>Paper tape reader<br>Paper tape reader/punch<br>RXV11 floppy disk system  |
| System utilities                  | OTL On-line Task Loader<br>SIP System Image Preservation Program<br>MCR Subset Console Interface  |

attributes that affect a task's operation in a real-time multiprogramming environment are:

- partition—the section of memory where the task will reside when it executes
- priority—the task's relationship to other tasks competing for system resources

Once a task is built, it can be installed in the system and executed. Task installation simply registers a task's attributes with the system. The task is not in memory, nor is it in competition for system resources. An installed task can be put in active competition for system resources by the operator or by another active task in the system.

When an installed task is activated, the system will allocate necessary resources, bring the task into memory for execution, and place it in competition with other active tasks. Task installation is the basis for efficient task operation. An installed task uses very little memory resources; yet, when the task is needed to service a real-time event, it can be introduced into the system quickly since its basic parameters are already known to the system.

Tasks can also share code and data among themselves through the Shareable Global Areas facility. A Shareable Global Area (SGA) is made accessible to the system and to tasks by installing the SGA and the task which intends to use it.

The following paragraphs describe how the RSX-11's real-time facilities handle task execution.

### **Multiprogramming**

Multiprogramming is the concurrent execution of two or more tasks residing in memory. In a single processor, only one task can have control of the CPU at a time. When that task does not need CPU time (for example, when it is waiting for input from a terminal) another task that needs CPU time can execute. The multiprogramming of tasks is accomplished by logically dividing available memory into a number of named partitions. Tasks are built to execute out of a specific partition, and all partitions in the system can operate in parallel. Partitions are user controlled allowing the user to handle the allocation of memory for the execution of tasks. A user controlled partition is allocated to only one task at a time. The user has complete control over system activity in this type of partition. As a result, it provides an ideal environment for a real-time task's execution.

In RSX-11M or RSX-11S systems, a user controlled partition can be subdivided into as many as seven non-overlapping subpartitions. The subpartitions occupy the identical physical memory occupied by the main partition. Tasks built to execute in the subpartitions can execute in parallel. Tasks can not, however be resident in a main partition and its subpartitions simultaneously. If a main partition is occupied, the subpartitions can not be. All subpartitions can have tasks residing in them; therefore, up to seven potentially parallel task executions can exist within a pre-empted user controlled main partition. The goal of subpartitioning is to reclaim large memory areas when a task requiring a main partition is no longer active.

When installing tasks into a partition, a task is linked to be installed and run in a partition with a specific base address. It can not run in any partition whose base address is not the same.

### **Priority Scheduling**

Task scheduling in RSX-11S is primarily event-driven, in contrast to systems which use a time slice mechanism for determining a task's eligibility to execute. The basis of event-driven task scheduling is the software priority assigned to each active task. A task's default priority is set when the task is built. It can be altered once it is installed (though not active) by an MCR command from the console.

Tasks are run at a software priority level ranging from a low of 1 to a high of 250. The Executive grants central processor resources to the highest priority task capable of execution. That task retains control of the central processor until it declares a significant event.

A significant event occurs when a task issues a system directive that implicitly or explicitly suspends a task's execution, or when an external interrupt occurs that can affect a task's execution. For example, a task can issue a directive that indicates it wants to wait until an I/O operation is complete before continuing execution; a significant event is declared when the I/O operation is complete. A special system directive also exists that allows a task to stimulate the event-driven task scheduling mechanism explicitly.

When a significant event is declared, the Executive interrupts the executing task and searches for a task capable of executing. The highest priority task that has all the resources it needs to run, and can make use of the resources it needs, will be the task that gains control of the CPU.

Event flags are associated with significant events. When a significant event occurs, the event flag indicates the specific cause of the interrupt.

There are 64 event flags. Flags 1 through 32 are local to the task, while event flags 33 through 64 are common to all tasks. A task can set, clear, test, and wait for any event flag or combination of event flags to achieve efficient synchronization between itself and other tasks in the system.

For example, upon completion of I/O requests, a device handler normally sets a requestor-indicated event flag and declares a significant event. If a requesting task instructs the system that it cannot run until an event flag is set (signalling task I/O completion), other eligible tasks of lower priority may run. In the scan of the active task list, a task that is awaiting I/O completion is bypassed until a significant event is declared through the setting of an event flag upon task I/O completion.

Although event-driven scheduling is the primary RSX-11 task scheduling mechanism, it is not the only mechanism available. As an option during system generation, RSX-11S allows the user to supplement event-driven task scheduling with time-sliced or time-based scheduling. The time-slice scheduling is based on a priority range specified by the user during system generation. All tasks that have priorities within the specified range are scheduled using a time-slice algorithm. Tasks with higher or lower priorities than the specified range receive service in an event-driven manner. As a whole, the task range also receives service

in an event-driven manner, but CPU time among the tasks within the range is shared.

### **Contingency Exits (System Traps)**

Subroutines automatically entered as the result of an unanticipated synchronous condition (for example, an attempt to execute an illegal instruction) or as the result of an asynchronous condition anticipated or unanticipated (for example an I/O termination) are called contingency exit or system trap routines.

System traps are another means of governing task execution. While significant events have a system-wide scope, traps are local to a task. Traps interrupt the sequence of instruction execution in the task and cause control to be transferred to a prespecified point in the program. In this way, system traps provide the ability to service certain conditions without continuously testing for their existence.

When a task plans to use the system trap facility, it must contain a trap service routine. This routine is automatically entered when the trap occurs using the task's normal priority and privilege. If a service routine is not supplied, the action taken by the Executive is dependent upon the type of trap.

There are two types of system traps: Synchronous System Traps (SST's) and Asynchronous System Traps (AST's).

SST's provide a means of servicing fault conditions within a task. These conditions, which are internal to a task and are not significant events, occur synchronously with respect to task execution. In these cases, if an SST service routine is not included in the task, the task's execution is aborted.

AST's commonly occur as the result of a significant event and thus occur asynchronously with respect to a task's execution. A task does not have direct or complete control over when AST's occur. A characteristic of AST's is that they are for information purposes, such as signifying an I/O completion that a task wants to know about immediately. If an AST service routine is not provided, a trap does not occur and task execution is not interrupted.

It should be emphasized that SST's are only initiated by the Executive; no further action is taken. That is, they appear to the Executive just like normal task execution. The Executive, having initiated an SST, cannot determine that the task is in the SST service routine. Thus, an SST service routine can be interrupted by another SST or an AST. SST's can be nested.

SST's are caused by activities internal to the task, while AST's occur as a result of an external event. The Executive keeps track of all AST's, queues them first-in, first-out, and is aware that a task is executing an AST.

### **Power Failure Restart**

Power failure restart is the ability of a system to smooth out intermittent short-term power fluctuations with no apparent loss of service and without losing data, all the while maintaining logical consistency within the

system itself and the application tasks. Power failure restart can be implemented in systems using non-volatile memory (PROM or core), or systems using battery backup (user-implemented) power for processor and volatile memory. Power failure affects absolute response time and peak capacity differently from the facilities previously discussed, since it applies to the aggregate system performance rather than increasing performance when the system is actually in operation. A system is not performing when it is shut down, and if the Executive can reduce the shutdown periods due to power failure restart, aggregate performance is increased.

RSX-11S systems perform the phases of power failure restart listed below:

1. When power begins to fail, the processor traps to the Executive which stores all register contents.
2. When power is restored, the Executive again receives control and restores the previously preserved state of the system.
3. The Executive then informs any tasks that have requested power failure restart notification through the Asynchronous System Trap mechanism that a power failure has occurred. These tasks can then, if required, make restorations of state they deem necessary.
4. The Executive schedules all device drivers that were active at the time the power failure occurred at their powerfail entry point. Drivers have the option of always being scheduled on power recovery, or only when the driver has an outstanding I/O order.

These drivers can then, if required, make those restorations of state (for example, repeating I/O requests) they deem necessary.

The RSX-11S family's approach is quite efficient because the repeating of I/O is placed nearest the source most likely to know how to make the restoration.

### **5.3 SYSTEM ORGANIZATION AND GENERATION**

The following paragraphs discuss the basic design elements of the RSX-11S operating system. Total system structure is essentially dependent on the decisions that the user makes during system generation. The user defines the system organization and chooses the Executive services appropriate for the particular applications environment.

There are three basic functional uses for which memory is allocated. The amount of memory allocated to each function is specified by the user during system generation. The three functional memory spaces are for:

- the RSX-11S Executive
- partition space for tasks and shared global areas
- the system communication areas, including system lists and tables

RSX-11S systems are designed to provide the most efficient use of system resources during system operation. To be useful to a wide range of applications and still obtain maximum system performance for a given

operating environment, RSX-11S systems require the user to become more involved in system generation.

System generation for RSX-11S systems provide the user with absolute control over system features and capabilities. Users concerned about size can eliminate the Executive services that are not essential to a particular application. In addition, I/O device drivers must be included in the Executive during system generation. Their inclusion in the Executive serves two purposes: it enhances overall system operation and it greatly decreases I/O driver size. System operation is enhanced because interrupt response, processing speed, and system throughput are increased. The size of the individual I/O drivers is decreased for two reasons. They do not have to contain code to initialize themselves, since they are initialized during system generation. Furthermore, the Executive can perform many operations common to all drivers; the drivers do not have to contain duplicate code required for independent operation.

For RSX-11S system generation, the user has to define the system communication area parameters, specify the sizes and base addresses of the partitions, and select the Executive services needed for the particular application. System generation is performed in two phases: the first phase defines the hardware configurations and software options; the second phase builds the complete system. Some system generation parameters can be changed on-line, for example, partition configuration. If Executive services are to be changed, however, the user must regenerate the system.

RSX-11S requires an RSX-11M system for system generation and program development. An RSX-11S system is generated from the RSX-11M system using the standard system generation process. The maximum hardware and software configuration is the same as that of an RSX-11M system with the exceptions of file system support, non-resident tasks, task checkpointing, and dynamic memory allocation and available peripherals.

The basic software building blocks for an RSX-11S system are:

1. The generatable features of the RSX-11M Executive (2.5 to 4K), including a special File Control Services (FCS) that contains no support for directory devices
2. All RSX-11M I/O device drivers
3. Subset MCR (2K)
4. Online Task Loader (2.5K)
5. System Image Preservation Program (1.5K)

The minimum software system is an Executive. The smallest Executive that can be generated requires 2.5K words of memory. Services that are omitted from the 2.5K Executive include:

- Address checking
- Asynchronous System Traps

- I/O rundown
- Task termination and device-not-ready notification
- External MCR functions (user-written functions)
- Install, Request, and Remove-on-exit support
- SEND, RECEIVE, GET TASK PARAMETERS,\* GET SENSE SWITCHES and GET PARTITION PARAMETERS directives
- Parity Memory support
- Network support
- All I/O drivers

Although omitted from the minimum Executive, these features can be generated into an RSX-11S system at the cost of memory.

The minimum RSX-11S software system must include the Executive and I/O device drivers. For example, two to four I/O device drivers could be added to the minimum Executive at the cost of an additional 1.5K words of memory. In an 8K word system, approximately 4K words would be available to application tasks.

If operator communication is required, subset MCR can be included in a system at a cost of 2K of memory. In an 8K system this still leaves approximately 2K for application tasks.

The On-line Task Loader (OTL) can be included in an RSX-11S system if on-line loading of tasks is desired, OTL is capable of loading tasks from the following media:

- Paper Tape Reader
- Floppy Disk (RXV11)

Tasks are created on a host RSX-11M system, transferred to the load medium using RSX-11M's File Exchange Utility (FLX), and then loaded into a running RSX-11S system using OTL. The minimum size for OTL is 2.5K words. In 2.5K words, however, OTL supports only one load device. On-line task loading requires a 16K word system, since approximately 8.5K words will be required for system software (2.5K Executive, 2K MCR, 1.5K device drivers, and 2.5K OTL).

The System Image Preservation Program (SIP) is an on-line utility task that provides the capability to save the image of a running system onto a load device-medium in bootstrappable format. Permissible load devices are the same as for OTL. The saved system can subsequently be restored by bootstrapping it from the load device medium. The minimum size for SIP is 1.5K words. In 1.5K words, it can support only one load device.

The standard RSX-11M File Control Services (FCS) record I/O package contains a large amount of code to support file-structured devices, RSX-11S contains no file support and therefore this code is unnecessary. The special version of FCS provided with RSX-11S is the standard FCS without the file support code. This provides a significant reduction in size while maintaining complete transparency.

## 5.4 SYSTEM CONVENTIONS

To simplify operations, RSX-11 systems observe certain conventions with respect to devices and operator commands.

### 5.4.1 Devices

The RSX-11S system supports a variety of peripheral devices. They are referred to by a 2-letter name and an optional 1- or 2-digit unit number followed by a colon. For example, TT12: represents user terminal number 12. Peripheral devices can be referred to by mnemonics, by pseudo-device names, or, in task references, by logical unit numbers. In addition, RSX-11M systems support logical device name assignments.

Pseudo-device names are associated with normal device mnemonics assigned by the system manager. They permit the system manager to dynamically determine the physical devices that will send or receive information. The following are pseudo-devices:

- SY: System device; indicates the device on which the system disk is mounted.
- TI: Terminal interface; indicates the terminal with which a particular task is associated. Each terminal has a unique TI. The TI of each task is assigned to the requesting terminal.
- CL: Console log; indicates the device normally used for the listing of files. The CL device is normally redirected to the high-speed printer (LAV11).
- CO: Console output; indicates the device by which the system can communicate with the system manager. The CO device is normally redirected to the system console.

Logical unit numbers (LUN's) provide the mechanism for programs to maintain device independence. The logical unit numbers used in a program can be assigned by means of device mnemonics to any available peripheral device that performs the desired function. LUN's can be assigned by the programmer at task build time, or by the task itself at run-time. Because the system provides default LUN assignments, it is not always necessary to assign a LUN to a task. Furthermore, LUN's can be changed by an MCR function for any installed, inactive, non-fixed task.

### 5.4.2 MCR Operator Commands and Terminal Control

The Monitor Console Routine (MCR) is the terminal interface between the user and the RSX-11S operating system. In the system, terminals can have either of two functions: command or slave. The system does not accept any unsolicited input from a slave terminal; its I/O is normally under task control. A command terminal is used to activate MCR and interface with the system using MCR system commands.

MCR's system commands enable the general user to perform the following functions:

- gain access to the system
- initiate and terminate execution of system or user programs

In addition, the privileged user can perform the following additional functions:

- adjust, modify and control the system environment

The privileged MCR user has complete control over the system's operation. The general user has no protection from the operations directed by the privileged user.

The RSX-11 systems include four different kinds of MCR commands: initialization commands, informational commands, task control commands, and system maintenance commands. Table 5-2 lists the RSX-11S MCR commands, and indicates their system functions.

**Table 5-2 RSX-11 MCR Commands**

### **Initialization Command**

| Command | Function   |
|---------|--|
| TIME    | Lists the time and date maintained in the system clock calendar. A privileged user can change the time and date. |

### **Informational Commands**

| Command          | Function  |
|------------------|---|
| ACTIVE TASK LIST | Lists the active tasks in the system, indicating the tasks' current status, for example, task suspended, waiting for I/O, etc.  |
| TASK LIST        | Lists a description of each task installed in the system, including task name, version number, default partition name, priority and size. (RSX-11D, this is an option of the SYSTEM command.) |

### **Task Control Commands**

| Command  | Function   |
|----------|--|
| REDIRECT | Redirects all I/O requests from one physical device to another.  |
| RUN      | Initiates the execution of an installed task. An installed task can be started immediately, started a specified time from when the command is issued, started a specified time from the next time unit, or started at an absolute time of day. A special option of the RUN command allows the user to run a task not installed; when issued, the task is installed, run and removed on exit. In all cases except the latter, the user can specify a reschedule interval for the task. In addition, RSX-11D provides a RUN command option to run a program only if there is available memory. |
| CANCEL   | Cancels any pending periodic rescheduling for a task.  |
| ABORT    | Terminates execution of a specified task.  |
| RESUME   | Continues execution of a previously suspended task.  |
| REMOVE   | Removes a task name from the System Task Directory (opposite of INSTALL). Under RSX-11D, this command is also used to remove a Shareable Global Area (SGA) from the Global Common Directory.   |

**Table 5-2 RSX-11 MCR Commands (Cont.)**

**System Maintenance Command**

| Command | Function  |
|---------|---|
| OPEN    | Allows the privileged user to examine or modify a word in memory. |

In addition to the MCR commands available to control system execution, RSX-11 system provide the following special terminal control characters:

|        |  |
|--------|--|
| CTRL/C | Activates MCR at a terminal. The system types the prompt "MCR>". Note that, unlike most other PDP-11 systems, the RSX-11 family does not use CTRL/C to affect execution of any currently running tasks other than MCR. |
| CTRL/Z | Logical end-of-file; when typed in response to a prompt from most utility programs, CTRL/Z causes the program to exit.   |
| CTRL/I | Causes a horizontal tab.   |
| CTRL/U | Cancels the current input line.  |
| CTRL/O | Enables or disables output to the terminal.  |
| CTRL/S | Temporarily suspends output to the terminal. This feature enables users with high-speed terminals to fill the display screen, stop output with a CTRL/S and then continue with a CTRL/Q.                               |
| CTRL/Q | Resumes printing of characters on the terminal from the point at which printing was interrupted using CTRL/S.  |

**5.5 SYSTEM DIRECTIVES**

System directives are instructions to RSX-11S to perform functions for an executing task. System directives allow tasks to perform the following:

- Schedule other tasks,
- Communicate with other tasks,
- Measure time intervals,
- Perform I/O functions,
- Suspend execution,
- Exit

Directives are generated by MACRO programs via macro calls and are supported for FORTRAN by library routines supplied by DIGITAL.

Directives are implemented solely through the PDP-11's EMT 377 instruction. By using only EMT 377, programs using EMT 0 through EMT 376 can be run via a non-RSX system trap. Any EMT other than EMT 377 traps to a task-contained service routine that can be written to simulate another environment to whatever degree is desired.

By using macro calls, instead of executing the directive, the programmer need only reassemble a program if changes are made in the directive specifications, rather than being required to edit the source code.

Listed below are the RSX-11S Executive directives.

## Task Execution Control Directives

|         |   |
|---------|---|
| REQUEST | Instruct the executive to request immediate execution of an indicated task. Memory partition, priority, and UIC may be specified.   |
| RUN     | Instruct the executive to schedule an indicated task's execution at a time specified in terms of delta time from issuance. Periodic rescheduling, memory partition, priority, and UIC may be specified. |
| CANCEL  | Instruct the executive to cancel scheduled requests for an indicated task's execution. Normally, all scheduling for the indicated task is cancelled.  |
| SUSPEND | Instruct the executive to suspend execution of the issuing task until explicitly resumed via a RESUME directive.  |
| EXIT    | Instruct the executive that the issuing task has completed its execution. Unless the exiting task is fixed, its memory is freed for use by other tasks.   |
| ABORT   | Instruct the executive to terminate the execution of an indicated task.   |

## Informational Directives

|                          |   |
|--------------------------|---|
| GET TASK PARAMETERS      | Instruct the executive to return parameters related to the issuing task.  |
| GET PARTITION PARAMETERS | Instruct the executive to return parameters related to an indicated memory partition.                                     |
| GET LUN DATA             | Instruct the executive to return information regarding an indicated logical unit to the issuing task.                     |
| GET TIME PARAMETERS      | Instruct the executive to return current time parameters (year, month, day, hour, minute, second, tick and ticks/second). |
| GET SENSE SWITCHES       | Instruct the executive to return the polarities of the sixteen console switches.  |

## Event-associated Directives

|                           |  |
|---------------------------|--|
| DECLARE SIGNIFICANT EVENT | Instruct the executive to test for a new "highest priority task capable of execution" and, if found, interrupt the currently executing task and start the execution of the new highest priority task. An event flag to be set may also be specified. |
| SET EVENT FLAG            | Instruct the executive to set an indicated event flag and return the previous polarity of the indicated flag (without a declaration of a significant event).   |

|                                |  |
|--------------------------------|--|
| CLEAR EVENT FLAG               | Instruct the executive to clear an indicated event flag and return the previous polarity of the indicated flag.  |
| READ ALL FLAGS                 | Instruct the executive to return the polarities of all event flags.  |
| WAIT FOR SINGLE EVENT FLAG     | Instruct the executive to suspend the execution of the issuing task until an event flag of an indicated set of flags is set.                                 |
| WAIT FOR LOGICAL OR OF FLAGS   | Suspend execution of the issuing task until any indicated event flag in one of five groups is set.   |
| WAIT FOR ANY SIGNIFICANT EVENT | Instruct the executive to suspend the execution of the issuing task until the next significant event.  |
| EXIT IF                        | Instruct the executive to cause the issuing task to exit, if, and only if, an indicated event flag is clear, i.e., to continue execution if the flag is set. |

#### Trap-associated Directives

|                            |   |
|----------------------------|---|
| MARK TIME                  | Instruct the executive to set an indicated event flag after a specified period of time, and/or cause an AST (Asynchronous System Trap) after the specified time has elapsed.                        |
| CANCEL MARK TIME REQUESTS  | Instruct the executive to cancel MARK TIME requests that have been made by the issuing task.  |
| DISABLE AST RECOGNITION    | Instruct the executive to inhibit AST (Asynchronous System Trap) recognition for the issuing task; the AST's are queued, and only their recognition is inhibited.                                   |
| ENABLE AST RECOGNITION     | Instruct the executive to enable AST recognition for the issuing task.  |
| SPECIFY POWER RECOVERY AST | Tell the system whether or not power recovery AST's are desired for the issuing task. If desired, this directive indicates where control is to be transferred when the AST occurs.                  |
| SPECIFY FPP EXECUTION AST  | Instruct the executive that the issuing task contains an AST service routine whose execution is to interrupt the task's execution whenever a floating point processor exception (error) occurs.     |
| SPECIFY RECEIVE AST        | Instruct the executive that the issuing task contains an AST service routine whose execution is to interrupt the task's execution whenever data is sent to the issuing task via the SEND directive. |

|  |  |
|--|--|
| AST EXIT                                   | Instruct the executive that the issuing task is exiting from an AST service routine.   |
| SPECIFY SST VECTORS                        | Instruct the executive that the issuing task contains a table of addresses of service routines to be executed upon task trap or fault conditions (viz., memory protect violation).   |
| SPECIFY SST VECTOR TABLE FOR DEBUGGING AID | Specifies the virtual address of a table of synchronous system trap service routine entry points for use by ODT or other debugging aids.   |
| <b>I/O Related Directives</b>              |  |
| QUEUE I/O                                  | Instruct the executive to queue an I/O request for the issuing task. This request is queued by priority for a logical unit which is assigned to a physical unit via various mechanisms. An event flag, an AST, and an in-task status block may be specified as an I/O completion indication. |
| ASSIGN LUN                                 | Instruct the executive to assign a logical unit number (LUN) of the issuing task to an indicated physical unit.  |
| SEND DATA                                  | Instruct the executive to send a block of data to an indicated task.   |
| RECEIVE DATA                               | Instruct the executive that the issuing task wishes to receive data that has been sent from another task via the SEND directive.   |
| RECEIVE DATA OR EXIT                       | Instruct the executive to attempt to receive data for the issuing task. If no data is received, the issuing task is allowed to exit.   |
| GET.MCR COMMAND                            | Instruct the executive to transfer an MCR (console) command line to the issuing task.  |

## 5.6 LANGUAGES

RSX-11 programs can be written in MACRO or FORTRAN IV. Programs written in MACRO or FORTRAN are translated into object code by the MACRO assembler or FORTRAN IV compiler, and then are task built into executable images by the TKB Task Builder.

### 5.6.1 MACRO

RSX-11 MACRO assembler allows the user to take full advantage of the PDP-11 instruction set and the RSX-11 system directives. During assembly, MACRO detects errors and produces output indicating the types of errors found, thus simplifying the process of locating program errors.

### 5.6.2 FORTRAN IV

RSX-11 FORTRAN is an ANSI standard FORTRAN IV language accompanied by an object time system (OTS). FORTRAN programs can be compiled into object code which can be linked, optionally with other object programs (either FORTRAN or MACRO), into executable task images. The OTS is an object library of commonly used FORTRAN routines, including math, error handling, process I/O, laboratory peripheral routines, and Executive directives.

## MACRO

**6.1 FUNCTIONS AND FEATURES**

PDP-11 MACRO processes source programs written in the MACRO assembly language and produces a relocatable object module and optional assembly listing. MACRO is included with the RT-11 operating system.

MACRO provides the following features:

- relocatable object modules
- global symbols for linking separately-assembled object programs
- device and filename specifications for input and output files
- user-defined macros
- comprehensive system macro library
- program sectioning directives
- conditional assembly directives
- assembly and listing control functions at program and command string levels
- alphabetized, formatted symbol table listing
- default error listing on command output device

The MACRO assembler included in the RT-11 system also features:

- a Cross Reference Table (CREF) symbol listing

**6.2 LANGUAGE**

A MACRO source program is composed of a sequence of source coding lines. Each source line contains a single assembly language statement followed by a statement terminator, such as a carriage return. The assembler processes source statements sequentially, generating binary machine instructions and data words or performing assembly-time operations (such as macro expansions) for each statement.

A statement can contain up to four fields which are identified by order of appearance and by specified terminating characters. The general format of a MACRO assembly language statement is:

```
label:   operator   operand(s)   ;comments
```

The label and comment fields are optional. The operator and operand fields are interdependent; either can be omitted depending upon the contents of the other. Some statements have one operand, for example:

```
CLR     R0
```

while others have two:

```
MOV     #344,R2
```

A label is a unique user-defined symbol which is assigned the current location counter and entered into the user-defined symbol table. A label is a symbolic means of referring to a specific location within a program. The value of the label can be either absolute (fixed in memory independently of the position of the program) or relocatable (not fixed in memory), depending on whether the location counter value is currently absolute or relocatable.

An operator field follows the label field, if present, and can contain a macro call, a PDP-11 instruction mnemonic, or an assembler directive. When the operator is a macro call, the assembler inserts the appropriate code during assembly to expand the macro. When the operator is an instruction mnemonic, it specifies the instruction to be generated and the action to be performed on any operands which follow. When the operator is an assembler directive, it specifies a certain function or action to be performed during assembly.

An operand is that part of the statement manipulated by the operator. Operands can be expressions, numbers, symbolic arguments, or macro arguments.

The comment field can contain any ASCII text characters. Comments do not affect assembly processing or program execution, but are useful in source listings for later analysis, documentation or debugging purposes.

### **6.2.1 Symbols and Symbol Definitions**

Three types of symbols can be defined for use within MACRO source programs: permanent symbols, user-defined symbols and macro symbols. Correspondingly, MACRO maintains three types of symbol tables: the Permanent Symbol Table (PST), the User Symbol Table (UST) and the Macro Symbol Table (MST).

Permanent symbols consist of the PDP-11 instruction mnemonics and MACRO directives. The PST contains all the permanent symbols automatically recognized by MACRO and is part of the assembler itself. Since these symbols are permanent, they do not have to be defined by the user in the source program.

User-defined symbols are those used as labels or defined by direct assignment. Macro symbols are those symbols used as macro names. The UST and MST are constructed during assembly by adding the symbols to the UST or MST as they are encountered.

The value of a symbol depends on its use in the program. A symbol in the operator field can be a macro symbol, a user-defined symbol or a permanent symbol. To determine the value of the symbol, the assembler searches the three symbol tables in the following order: MST, UST and PST.

A symbol used in the operand field can be either a user-defined symbol or a permanent symbol. To determine the value of the symbol, the assembler searches the User Symbol Table and the Permanent Symbol Table in that order.

These search orders allow redefinition of Permanent Symbol Table entries as user-defined or macro symbols. The same name can be assigned to both a macro and a label.

User-defined symbols are either internal or external (global) to a source program module. An internal symbol definition is limited to the module in which it appears. A global symbol can be defined in one source program module and referenced within another.

Internal symbols are temporary definitions which are resolved by the assembler. Global symbols are preserved in the object module and are not resolved until the object modules are linked into an executable program. With some exceptions, all user-defined symbols are internal unless explicitly defined as being global.

When a label is given to a program statement, a symbol table entry is made and the value of the current location counter is assigned to it.

A direct assignment statement associates a symbol with a value. When a direct assignment is first used to define a symbol, that symbol is entered into the User-defined Symbol Table, and the specified value is associated with it. The general format for a direct assignment is:

symbol = expression

Expressions are combinations of terms that are joined together by binary operators and that reduce to a 16-bit value. Binary operators are, for example, addition, subtraction, multiplication, division, logical AND, and logical inclusive OR.

The expression in a direct assignment statement can itself be a reference to another symbol. In this way, a symbol can be redefined in a subsequent direct assignment statement if the symbol definition contains a reference to a subsequently-defined symbol. Only one level of forward referencing is allowed. The following example illustrates an illegal forward reference:

```
X = Y (Legal forward reference)
Y = Z (Illegal forward reference)
Z = 1
```

Although one level of forward referencing is allowed for local symbols, a global symbol defined in a direct assignment statement must not contain a forward reference. The global assignment expression ( $=$ ) must not itself contain an undefined reference to another symbol.

Local symbols are specially-formatted internal symbols used as labels within a given range of source code, called a local symbol block. Local symbols are of the form  $n\$$ , where  $n$  is a decimal integer between 1 and 65535, inclusive. Examples of local symbols are: 1\$, 27\$, 59\$, 104\$.

A local symbol block can be delimited in one of three ways:

- The range of a local symbol block usually consists of those statements between two normally-defined labels.

- The range of a local symbol block is normally terminated upon entering a new program section, as defined by a program section directive.
- The range of a local symbol block can be explicitly defined by the use of the `.ENABL` and `.DSABL` directives.

Local symbols provide a convenient means of generating labels to be referenced only within a local symbol block. The use of local symbols reduces the possibility of entry point symbols with multiple definitions appearing within a program. A local symbol, then, is not referenced from other source program modules or even from outside its local symbol block. Thus, local symbols of the same name can appear in other local symbol blocks without conflict.

### 6.2.2 Directives

A program statement can contain one of three different operators: a macro call, a PDP-11 instruction mnemonic, or an assembler directive. MACRO includes directives for:

- listing control
- function specification
- data storage
- radix and numeric usage declarations
- location counter control
- program termination
- program boundaries information
- program sectioning
- global symbol definition
- conditional assembly
- macro definition
- macro attributes
- macro message control
- repeat block definition
- macro libraries

Table 6-1 lists the MACRO directives.

**Table 6-1 Assembly and Macro Directives**

---

#### Listing Control Directives

---

|                     |  |
|---------------------|--|
| <code>.LIST</code>  | Control the listing of source lines, sequence numbers, current location counter field, generated binary code, source code, comments, macro expansions, table of contents, symbol table, etc. |
| <code>.NLIST</code> |  |
| <code>.TITLE</code> | Assigns a name to the object module and provides the header of each page in the assembly listing.  |

**Table 6-1 Assembly and Macro Directives (Cont.)**

|        |   |
|--------|---|
| .SBTTL | Identifies an element to be included in the assembly listing table of contents. |
| .IDENT | Provides an additional label for the object module.                             |
| .PAGE  | Ejects a page in the assembly listing. Same as issuing a form feed.             |

---

Function Directives

---

|        |  |
|--------|--|
| .ENABL | Enables or disables the following function control options:  |
| .DSABL | produce absolute binary output, assemble all relative addresses as absolute addresses (useful during debugging), ignore card column sequence numbers, truncate or round floating-point values, accept lower case input, permit a local symbol block to cross program boundaries (useful for multiple-entrant routines), inhibit binary output. |

---

Data Storage Directives

---

|        |  |
|--------|--|
| .BLKB  | Reserves a byte- or word-aligned block of storage in the object program.   |
| .BLKW  |  |
| .BYTE  | Stores a binary value in a byte in the object module. Used to generate successive bytes of data.   |
| .WORD  | Stores a binary value in a word in the object module. Used to generate successive words of data.   |
| '      | Stores the ASCII code(s) for the given character(s) following the apostrophe or quote in a byte or word. Used to generate text characters in the source code.  |
| ''     |  |
| .ASCII | Translates a character string into its equivalent 7-bit ASCII values and stores them in the object module.   |
| .ASCIZ | Translates a character string into its equivalent 7-bit ASCII values and stores them in the object module appending a zero byte to the string. This enables the program to identify the end of the string by searching for a null character (zero byte). |
| .RAD50 | Allows three ASCII characters to be packed into one word (Radix-50 format); using this directive, any 6-character symbol can be stored in two consecutive words.   |
| .FLT2  | Stores a floating-point number in 2-word floating-point format.  |
| .FLT4  | Stores a floating-point number in 4-word floating-point format.  |

**Table 6-1 Assembly and Macro Directives (Cont.)**

---

**Radix and Numeric Control Operators**

---

|        |  |
|--------|--|
| .RADIX | Declare any one of the following radices to apply to succeeding numbers in the source program: 2, 4, 8 or 10.          |
| ↑D     | Declare a (temporary) decimal, octal or binary radix for the number following the control operator.                    |
| ↑O     |  |
| ↑B     |  |
| ↑C     | Declare that the number following the control operator is to be one's complemented as it is evaluated during assembly. |
| ↑F     | Declare that the number following the control operator is to be interpreted as a 1-word floating point argument.       |

---

**Location Counter Control Directives**

---

|       |   |
|-------|---|
| .EVEN | Ensures that the current location counter contains an even value by adding one if the current value is odd. |
| .ODD  | Ensures that the current location counter contains an odd value by adding one if the current value is even. |

---

**Terminating Directives**

---

|      |   |
|------|---|
| .END | Indicates the logical end of source input and, optionally, specifies the entry point, starting or transfer address. |
| .EOT | End of input tape. Ignored by the Assembler (included for compatibility with PAL-11 assemblers).                    |

---

**Program Boundaries Directive**

---

|        |  |
|--------|--|
| .LIMIT | Reserves two words in the object module which, during linking, are used to store the address of the bottom of the program and the address of the first free word following the program image. This enables the program to determine its upper and lower address boundaries during execution. |
|--------|--|

---

**Program Sectioning Directives**

---

|        |  |
|--------|--|
| .ASECT | Begin or continue an absolute program section.   |
| .CSECT | Begin or continue a relocatable program section. |

---

**Symbol Control Directive**

---

|        |   |
|--------|---|
| .GLOBL | Defines (and thus provides linkage to) symbols not otherwise defined as global symbols within a module. |
|--------|---|

---

**Conditional Assembly Directives**

---

|     |  |
|-----|--|
| .IF | If the condition specified in the argument is met, include |
|-----|--|

**Table 6-1 Assembly and Macro Directives (Cont.)**

the following block of code in the assembly. Condition testing can be based on the value of an expression, the existence of a definition for a symbol, or the value of a macro-type argument.

|       |  |
|-------|--|
| .ENDC | Identifies the end of the conditional assembly block.  |
| .IFF  | The code following this subconditional directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, providing that the condition tested upon entering the conditional assembly block is false.     |
| .IFT  | The code following this subconditional directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, providing that the condition tested upon entering the conditional assembly block is true.      |
| .IFTF | The code following this subconditional directive, and continuing up to the next occurrence of a subconditional directive or to the end of the conditional assembly block, is to be included in the program, regardless of the result of the condition tested upon entering the conditional assembly block. |
| .IIF  | Assemble this line of code if the condition specified on the line is met.  |

---

**Macro Definition Directives**

---

|        |  |
|--------|--|
| .MACRO | Identifies the beginning of a macro definition.                          |
| .ENDM  | Identifies the end of a macro definition.                                |
| .MEXIT | Terminates a macro expansion before the end of the macro is encountered. |

---

**Macro Attribute Directives**

---

|        |   |
|--------|---|
| .NARG  | Determines the number of arguments in the macro call currently being expanded.  |
| .NCHR  | Determines the number of characters in a specified character string. It is useful in calculating the length of macro arguments. |
| .NTYPE | Determines the addressing mode of a specified macro argument.   |

---

**Macro Message Control Directives**

---

|        |  |
|--------|--|
| .ERROR | Sends a message to the listing file during assembly pass 2. A common use of this directive is to provide a diagnostic announcement of a rejected or erroneous macro call or to alert the user to the existence of an illegal set of conditions |
|--------|--|

**Table 6-1 Assembly and Macro Directives (Cont.)**

specified in a conditional assembly.  
.PRINT Identical to the .ERROR directive, except that it is not flagged in the assembly with an error code.

---

**Macro Repeat Block Directives**

---

.IRP Replaces a dummy argument with successive real arguments specified in an argument string.  
.IRPC Replaces a dummy argument with each successive character of the specified string.  
.REPT Duplicates a block of code a number of times in-line with other source code.

---

**Macro Library Directive**

---

.MCALL Includes in the assembly macro definitions which are taken from system or user macro library files.

**LISTING CONTROL DIRECTIVES**

Several listing control directives are provided in MACRO to control the content, format and pagination of all listing output generated during assembly. Facilities also exist for creating object module names and other identification information in the listing output.

The listing control options can also be specified at assembly-time through switch options included in the listing file specification in the command string issued to the MACRO assembler. The use of these switch options overrides all corresponding listing control directives in the source program.

When no listing file is specified, any errors encountered in the source program are printed on the terminal from which MACRO was initiated.

**FUNCTION DIRECTIVES**

Several function control options are provided by MACRO through the .ENABL and .DSABL directives. These directives are included in a source program to invoke or inhibit certain MACRO functions and operations incident to the assembly process itself. They include the ability to:

- Produce absolute binary output.
- Assemble all relative addresses as absolute addresses. This function is useful during the debugging phase of program development.
- Cause source columns 73 and greater (to the end of the line) to be treated as comment
- Truncate or round floating point literals.
- Accept lower case ASCII input instead of converting it to upper case.

- Enable a local symbol block to cross program section boundaries. A local symbol block is normally established by encountering a new symbolic label or a program section directive in the source program. By enabling a local symbol block to cross program section boundaries, a local symbol block can be established which is not terminated until another symbolic label or program section directive is encountered following a disable local symbol block function directive. Local symbols cannot, however, be defined in a program section other than that which was in effect when the block was entered. The basic function of this directive in regard to program sections is limited to those instances where it is desirable to leave a program section temporarily to store data, followed by a return to the original program section.
- Inhibit binary output.

#### CONDITIONAL ASSEMBLY DIRECTIVES

Conditional assembly directives enable the programmer to include or exclude blocks of source code during the assembly process, based on the evaluation of stated condition tests within the body of the program. This capability allows several variations of a program to be generated from the same source.

The user can define a conditional assembly block of code, and within that block, issue subconditional directives. Subconditional directives within conditional assembly blocks are used to indicate:

- The assembly of an alternate body of code when the condition of the block tests false.
- The assembly of a non-contiguous body of code within the conditional assembly block, depending on the result of the conditional test on entering the block.
- The unconditional assembly of a body of code within a conditional assembly block.

Conditional assembly directives can be nested. MACRO permits a nesting depth of 16 conditional assembly levels.

#### MACRO DEFINITIONS AND REPEAT BLOCKS

In assembly-language programming, it is often convenient and desirable to generate a recurring coding sequence by invoking a single statement within the program. In order to do this, the desired coding sequence is first established with dummy arguments as a macro definition. Once a macro has been defined, a single statement calling the macro by name with a list of real arguments (replacing the corresponding dummy arguments in the macro definition) generates the desired coding sequence or macro expansion.

Macros can be nested; that is, the definition of one macro can include a call to another. The depth of nesting allowed is dependent on the amount of memory used by the source program being assembled.

A label is often required in an expanded macro. Normally, a label can be explicitly specified as an argument with each macro call. Care must be taken, however, in issuing subsequent calls to the same macro in

order to avoid specifying a duplicate label as a real argument. This concern is eliminated through a feature of MACRO which creates a unique symbol where a label is required in an expanded macro.

MACRO can automatically create unique local symbols. This automatic facility is invoked on each call of a macro whose definition contains a dummy argument preceded by the question mark (?) character, if a real argument of the macro call is either null or missing. If the real argument is specified in the macro call, however, MACRO does not generate a local symbol and normal argument replacement occurs.

An indefinite repeat block is a structure that is very similar to a macro definition. Such a structure is essentially a macro definition that has only one dummy argument. At each expansion of the indefinite repeat range, this dummy argument is replaced with successive elements of a specified real argument list. An indefinite repeat block directive and its associated repeat range are coded in-line within the source program. This type of macro definition does not require calling the macro by name, as required in the expansion of conventional macros described above.

An indefinite repeat block can appear within or outside of another macro definition, indefinite repeat block, or repeat block.

#### MACRO CALLS AND STRUCTURED MACRO LIBRARIES

All macro definitions must occur prior to their references within the user program. MACRO provides a selection mechanism for the programmer to indicate in advance those system macro definitions required in the program. (System macros include the monitor programmed requests or executive directives available with each operating system.)

The .MCALL directive is used to specify the names of all the macro definitions not defined in the current program but which are used in the program. When this directive is encountered, MACRO searches the system macro library file to find the requested definition.

Each library file contains an index of the macro definitions it contains. When an .MCALL directive is encountered in the source program, MACRO searches the user macro library for the named macro definitions and, if necessary, continues the search with the system macro library. Because each macro library contains an index of all of its entries, MACRO searches only the index in each library to find where the macro definition is stored.

### 6.3 ASSEMBLER OPERATION

The MACRO Assembler assembles one or more ASCII sources containing MACRO statements into a single relocatable binary object program. MACRO can accept source data from any input device, such as a floppy disk or paper tape reader. The sources to be included in a single assembly are listed in the command string from left to right in the order in which they are to be assembled. The last statement in the last source specified is normally the .END statement.

Assembler output consists of the binary object file and an optional assembly listing followed by the symbol table listing. Using the MACRO

available under RT-11, cross reference (CREF) listings can also be produced.

MACRO is a two-pass assembler. During assembly pass one, MACRO locates and reads all required macros from libraries, builds symbol tables and program section tables for the program, and performs a rudimentary assembly of each source statement. During assembly pass two, MACRO completes the assembly, writes out an object file, and generates an assembly and symbol table listing for the program.

At the end of assembly pass one, MACRO determines whether a given global symbol is defined in the current program modules or whether it is to be treated as an external symbol. In general, all undefined global symbols appearing in a given program must be defined by the end of assembly pass one.

The object module MACRO produces must be processed by the operating system's linker utility program (called the Linker or Task Builder) to create an executable program. The linker joins separately-assembled object modules into a single load module (task image). The linker fixes (makes absolute) the values of the external or relocatable symbols in the object module.

To enable the linker to fix the value of an expression, MACRO passes it certain directives and parameters. In the case of the relocatable expressions in the object module, the linker adds the base of the associated relocatable program section to the value of the relocatable expression provided by MACRO. In the case of external expression values, the linker determines the value of the external term in the expression (since the external expression must be defined in at least one of the other object modules being linked together) and then adds it to the absolute portion of the external expression, as provided by MACRO.

In summary, an executable program image can be constructed from one or more source modules, which can be assembled either separately or together. The resultant object module(s) must be linked together using the linker utility. Figure 6-1 illustrates the processing steps required to produce an executable program from several sources stored as files.

#### PROGRAM SECTIONING

The MACRO program sectioning directives are used to declare names for program sections and to establish certain program section attributes. These program section attributes are used when the program is linked into an executable load module or task.

A program can consist of an absolute program section, an unnamed relocatable program section, and up to 254 named relocatable program sections. The absolute program section serves to "link" the program with fixed memory locations such as interrupt vectors and the peripheral device register addresses.

The relocatable program sections are also called control sections, since they normally contain instructions. The unnamed control section is internal to each object module. That is, every object module can have an

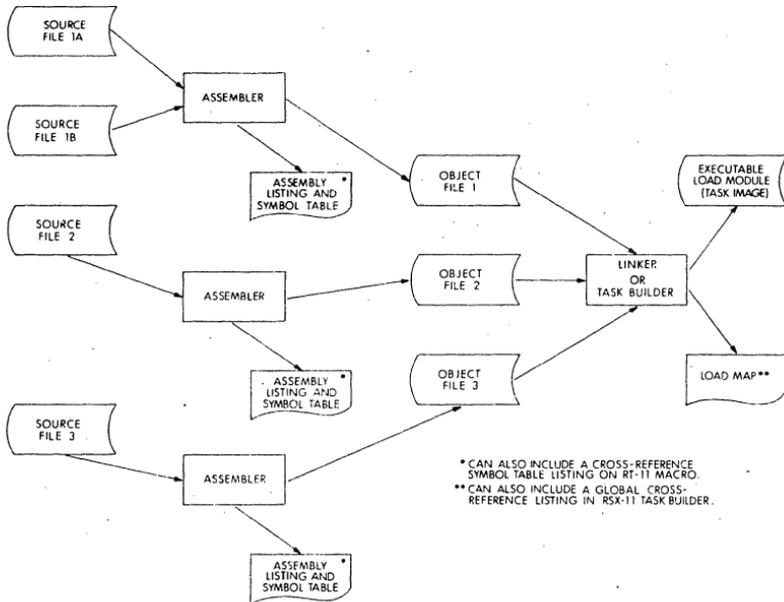


Figure 6-1 MACRO Assembly Procedure

unnamed control section but the linker treats each control section independently. Each is assigned an absolute address such that it occupies an exclusive area of memory. Named control sections, on the other hand, are treated globally, in the same manner as FORTRAN COMMON.\* If different object modules have control sections with the same name, they are all assigned the same absolute load address and the size of the area reserved for loading of the section is the size of the largest. Thus, named control sections allow for the sharing of data and/or instructions among object modules.

The assembler maintains separate location counters for each section. The first occurrence of a program section directive assumes that the current location counter is set at relocatable zero. The scope of this directive then extends until a directive declaring a different program section is specified. For example:

```

.CSECT                                ;start the unnamed relocatable section
A:  0                                  ;assembled at relocatable 0,
B:  0                                  ;relocatable 2 and
C:  0                                  ;relocatable 4
ST: CLR      A-                        ;assemble code at
      CLR      B                        ;relocatable address
      CLR      C                        ;6 through 21
.ASECT                                ;start the absolute section
  
```

\*If declared with the .PSECT directive (see below), they must have the attributes global and overlaid.

```

.=4                ;assemble code at
.WORD .+2, HALT   ;absolute locations 4 through 7
.CSECT            ;resume the unnamed relocatable section
INC              A ;assemble code at
BR              ST ;relocatable 22 through 27
.END

```

By maintaining separate location counters for each program section, MACRO allows the user to write statements which are not physically contiguous within the program, but which can be loaded contiguously following assembly.

MACRO under RT-11 includes two program sectioning directives: .CSECT and .ASECT. The .CSECT directive is used to define the named and unnamed relocatable program sections. The .ASECT directive is used to identify the portions of the absolute program section.

The .PSECT directive allows the user to exercise absolute control over the memory allocation of a program at task-build time, since any program attributes established through this directive are passed to the Task Builder. For example, if a programmer is writing programs for a multi-user environment, a program section containing pure code (instructions only) or a program section containing impure code (data only) can be explicitly declared through the .PSECT directive. Furthermore, these program sections can be explicitly declared as read-only code, qualifying them for use as protected, reentrant programs. In addition, program sections exhibiting the global attribute can be explicitly allocated in a task's overlay structure by the user at task-build time. The advantages gained through sectioning programs in this manner therefore relate primarily to control of memory allocation and program modularity.

The .PSECT directive allows the user to define the following program section attributes:

#### Access

Two types of access can be permitted to the program section: read-only or read/write.

#### Contents

A program section can contain either instructions or data. This attribute allows the Task Builder to differentiate global symbols that are program entry-point instructions from those that are data values.

#### Scope

The scope of the program section can be global or local. In building single-segment programs, the scope of the program has no meaning, because the total memory allocation for the program will go into the root segment of the task. The global or local attribute is significant only in the case of overlays. If an object module contains a local program section, then the storage allocation for that module will occur within the segment in which the module resides. Many modules can reference this same program section, and the memory allocation for each module is either concatenated or overlaid within the segment, depending on the argument of the program section defining its allocation requirements (see

below). If an object module contains a global program section, the memory area allocations to this program section are collected across segment boundaries, and the allocation of memory for that section will go into the segment nearest the root in which the first memory allocation to this program section appeared.

#### Relocatability

A program section can be absolute or relocatable. When a program section is declared to be absolute, the program section requires no relocation. The program section is assembled and loaded, starting at absolute virtual address 0. When the program section is declared to be relocatable, the Task Builder calculates a relocation bias and adds to it all references within the program section.

#### Allocation Requirements

The program section can be concatenated or overlaid. When concatenated, all memory allocations to the program section are to be concatenated with other references to this same program section in order to determine the total memory allocation requirements for this program section. When overlaid, all memory allocations to the program section are to be overlaid. Thus, the total allocation requirement for the program section is equal to the largest individual allocation request for this program section.

### 6.4 ASSEMBLER ENVIRONMENTS

MACRO requires an RT-11 system configuration (or background partition) of 12K words or more. If more than 12K words are available, MACRO will use the additional memory to increase the amount of symbol table space possible.

RT-11 MACRO provides a system macro library containing the expanded code for all the RT-11 Monitor's programmed requests. Refer to the RT-11 chapter in Section II of this handbook for a list of the RT-11 programmed requests.

Under the RT-11 operating system, a smaller version of MACRO, called ASEMBL, is available for users with minimum system configurations. ASEMBL has the same features as MACRO with the following exceptions:

- macro directives (.MACRO, .MCALL, .ENDM, .IRP, etc.) are not recognized
- DATE is not printed in listings
- wide line-printer output is not available
- there is no lower case mode
- there is no enable/disable punch directive
- there are no floating point directives
- there are no local symbols or local symbol blocks
- CREF is not available

Most of the macro definition features not available in ASEMBL are supported by EXPAND. EXPAND is an RT-11 system program which processes the macro references in a macro assembly language source file.

EXPAND simply copies its input files to its output file unless it encounters any of the following directives: .MCALL, .MACRO, .name, and .ENDM. The .MCALL directive instructs EXPAND to search the system macro library to find the macro names listed in the directive, and store their definition in internal tables. The .MACRO directive instructs EXPAND to copy a macro definition from the user's input file to its internal tables. The .name directive, if "name" is the name of a macro, instructs EXPAND to replace the macro call with the definition stored in its internal tables. The .ENDM directive terminates the macro definition when encountered while EXPAND stores a macro definition.



## FORTRAN IV

**7.1 FUNCTIONS AND FEATURES**

The FORTRAN IV compiler and Object Time System is available as an optional language processing system for the RT-11 operating system. The FORTRAN compiler accepts source programs written in the FORTRAN IV language and produces an object file which must be linked prior to execution. The PDP-11 FORTRAN IV language conforms to the specifications for the American National Standard FORTRAN X3.9-1966. The following are enhancements to the American National Standard:

- Array Subscripts—any arithmetic expression can be used as an array subscript. If the value of the expression is not an integer, it is converted to integer format.
- Array Dimensions—arrays can have up to seven dimensions.
- Character Literals—character strings bounded by apostrophes can be used in place of Hollerith constants.
- Mixed-mode Expressions—mixed-mode expressions can contain any data type, including complex and byte.
- End of line comments—any FORTRAN statement can be followed, in the same line, by a comment that begins with an exclamation point.
- Debugging Statements—statements that are included in a program for debugging purposes can be so designated by the letter D in column 1. Those statements are compiled only when the associated compiler command string option switch is set. They are treated as comments otherwise.
- Read/Write End-of-file or Error Condition Transfer—the specifications  $END = n$  and  $ERR = n$  (where  $n$  is a statement number) can be included in any READ or WRITE statement to transfer control to the specified statement upon detection of an end-of-file or error condition. The  $ERR = n$  option is also permitted in the ENCODE and DECODE statements, allowing program control of data format errors.
- General Expressions in I/O lists—general expressions are permitted in I/O lists of WRITE, TYPE, and PRINT statements.
- General Expression DO and GO TO Parameters—general expressions are permitted for the initial value, increment, and limit parameters in the DO statement, and as the control parameter in the computed GO TO statement.
- DO Increment Parameter—the value of the DO statement increment parameter can be negative.

- **Optional Statement Label List**—the statement label list in an assigned GO TO is optional.
- **Override Field Width Specifications**—undersized input data fields can contain external separators to override the FORMAT field width specifications for those fields (called "short field termination"), permitting free-format input from terminals.
- **Default FORMAT Widths**—the FORTRAN IV programmer may specify input or output formatting by type and default width, and precision values will be supplied.

- **Additional I/O Statements:**

Device-oriented I/O

ACCEPT

TYPE

PRINT

Memory-to-memory formatting

ENCODE

DECODE

Unformatted direct access I/O

DEFINE FILE

READ (u'r)    u = logical unit number

WRITE (u'r)   r = record number

FIND (u'r)

The unformatted direct access I/O facility allows the FORTRAN programmer to read and write files written in any format.

- **Logical Operations on INTEGER Data**—the logical operators .AND., .OR., .NOT., .XOR., and .EQV. may be applied to integer data to perform bit masking and manipulation.
- **Additional Data Type**—the byte data type (keyword LOGICAL\*1 or BYTE) is useful for storing small integer values as well as for storing and manipulating character information. It enables the programmer to save space when manipulating small integer values without affecting their characteristics, because they are treated internally as integer values. Since the arithmetic and masking operations are legal for the byte data type, it is also possible to manipulate and modify character data.
- **IMPLICIT Declaration**—the IMPLICIT statement has been added to redefine the implied data type of symbolic names.
- **Fewer Syntactic Restrictions**—FORTRAN IV has no statement ordering requirements, allowing declarations to appear anywhere within the source program. Terminal format input (using the tab character to delimit fields) eases program preparation.

The FORTRAN IV compiler performs well in small environments. On an RT-11 system with as little as 8K words of memory, FORTRAN IV can compile programs containing as many as 450 lines. On an RT-11 system with 28K words, FORTRAN IV can compile programs containing as many as 2200 lines.

Despite its small size requirements and high compilation rate, FORTRAN IV provides a high level of automatic object program optimization. The compiler performs redundant expression elimination, constant expression

folding, branch structure optimization, and several types of subscripting optimizations.

## 7.2 LANGUAGE

A FORTRAN program consists of FORTRAN statements and optional comments. There are two kinds of statements: executable and non-executable. Executable statements describe the action of the program. Non-executable statements describe the data arrangement and characteristics, and provide editing and data conversion information.

There are assignment statements, control statements, I/O statements, FORMAT statements and specification statements. FORMAT and specification statements are non-executable. Table 7-2 summarizes the FORTRAN IV language components.

Table 7-2 FORTRAN IV Language Summary

### Expression Operators

| TYPE       | OPERATOR | OPERATES ON   |   |
|------------|----------|---|---|
| Arithmetic | **       | exponentiation  | arithmetic or logical constants, variables and expressions  |
|            | *        | multiplication  |   |
|            | /        | division  |   |
|            | +, -     | addition, subtraction, unary plus and minus   |   |
| Relational | .GT.     | greater than  | arithmetic or logical constants, variables and expressions (all relational operators have equal priority) |
|            | .GE.     | greater than or equal to  |   |
|            | .LT.     | less than   |   |
|            | .LE.     | less than or equal to   |   |
|            | .EQ.     | equal to  |   |
|            | .NQ.     | not equal to  |   |
| Logical    | .NOT.    | .NOT.A is true if and only if A is false  | logical or integer constants, variables and expressions   |
|            | .AND.    | A.AND.B is true if and only if A and B are both true                                  |   |
|            | .OR.     | A.OR.B is true if and only if A or B or both are true                                 |   |
|            | .EQV.    | A.EQV.B is true if and only if either A and B are both true or A and B are both false |   |
|            | .XOR.    | A.XOR.B is true if and only if A is true and B is false or B is true and A is false   |   |

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

**Assignment Statements**

|                     |  |
|---------------------|--|
| variable=expression | Arithmetic/Logical Assignment:<br>The value of the arithmetic or logical expression is assigned to the variable.   |
| ASSIGN-TO           | The ASSIGN statement is used to associate a statement label with an integer variable. The variable can then be used as a transfer destination in a subsequent assigned GO TO statement in the same program unit. |

**Control Statements**

|          |               |  |
|----------|---------------|--|
| GO TO    | Unconditional | Transfers control to the same statement every time it is executed.   |
|          | Computed      | Permits a choice of transfer destinations, based on a value of an expression within the statement.   |
|          | Assigned      | Transfers control to a statement label that is represented by a variable. Because the relationship between the variable and a specific statement label must be established by an ASSIGN statement, the transfer destination can be changed, depending upon which ASSIGN statement was most recently executed.    |
| IF       | Arithmetic    | Transfers control to a statement depending on the value of an arithmetic expression. Used for conditional control transfers.   |
|          | Logical       | Executes a statement if the test of a logical expression is true.  |
| DO       |               | Causes the statements in its range to be repeatedly executed a specified number of times. The range of the DO begins with the statement following the DO and ends with a specified terminal statement. The number of iterations is determined by the values for the initial, terminal, and increment parameters. |
| CONTINUE |               | Causes no processing. Passes control to the next executable statement. Used primarily as the terminal statement of a DO loop when that loop would otherwise end with a GO TO, arithmetic IF, or other prohibited control statement.  |
| CALL     |               | Calls a SUBROUTINE subprogram and passes it actual arguments to replace the dummy arguments in the subprogram.   |

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

|        |  |   |
|--------|--|---|
| RETURN |  | Returns control from a subprogram to the calling program unit.  |
| PAUSE  |  | Prints a message (if specified) on the terminal and suspends execution until the user responds.   |
| STOP   |  | Terminates program execution and prints a message (if specified) on the terminal.   |
| END    |  | Marks the end of a program unit. In a main program, if control reaches the END statement, a CALL EXIT is implicitly executed. In a subprogram, a RETURN statement is implicitly executed. |

**Input/Output Statements**

|       |               |   |
|-------|---------------|---|
| READ  | Formatted     | Reads at least one logical record from the specified device according to the given format specifications, and assigns values to the elements in a list.   |
|       | Unformatted   | Reads one logical record from the specified device, assigning the input values to the variables in a list.  |
|       | Direct Access | Reads from the specified logical record and assigns the input values to the variables in a list.  |
|       | Error Control | Optional elements in the READ statement allow control transfer on error conditions. If an end-of-file condition is detected and the END option is specified, execution continues at a given statement. If a recoverable I/O error occurs and the ERR option is specified, execution continues at a given statement. |
| WRITE | Formatted     | Writes one or more logical records containing the values of the variables in a list onto the specified device in the given format.  |
|       | Unformatted   | Writes one logical record containing the values of the variables in the list onto the specified device.   |
|       | Direct Access | Writes one logical record containing the values of the variables in the list into the specified record of the given device.   |
|       | Error Control | Optional elements in the WRITE statement allow control transfer on error conditions. If an end-of-file condition is   |

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

|             |   |
|-------------|---|
|             | detected and the END option is specified, control transfers to the given statement. If an I/O error occurs and ERR is specified, execution continues at the given statement.                                    |
| ACCEPT      | Reads input from a given logical unit (normally associated with a terminal keyboard).   |
| TYPE        | Identical to a formatted WRITE except that output is directed to a logical unit normally associated with the terminal printer.  |
| PRINT       | Same as a TYPE statement, except that output is directed to a logical unit normally associated with the line printer.   |
| DEFINE FILE | Defines the record structure of a disk or DECTape direct access file: the logical unit number, the number of fixed-length records in the file, the length of single record, and the pointer to the next record. |
| REWIND      | The given logical unit is repositioned to the beginning of the currently open file.   |
| BACKSPACE   | The currently open file on the given logical unit is backspaced one record.   |
| END FILE    | An end-of-file record is written on the file open on the given logical unit.  |
| FIND        | Positions the direct access file on the given logical unit to the specified record and sets the associated pointer.   |
| ENCODE      | Changes the elements in the given list of variables into ASCII format. The ERR option allows control transfer to a given statement if an error condition is detected.   |
| DECODE      | Changes the elements in the given list of variables in ASCII format into internal binary format. The ERR option allows control transfer to a given statement if an error is detected.                           |

**Format Statements**

|        |   |
|--------|---|
| FORMAT | Describes the format in which one or more records are to be transmitted. The format descriptors include integer and octal, logical, |
|--------|---|

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

real, double precision, complex, literal and editing. Real, double precision and complex formats can be scaled.

**Specification Statements**

**IMPLICIT**

Overrides the implied data type of symbolic names, in which all names that begin with the letters I, J, K, L, M, or N are presumed to be INTEGER values, and all names beginning with any other letter are assumed to be REAL values, unless otherwise specified. IMPLICIT allows the programmer to define the initial letters for implied data types. If a variable is not given an explicit type, and its name begins with a letter defined in an IMPLICIT statement, its default type is that defined by the IMPLICIT statement.

**type var1,var2, . . . , varn**

**Type Declaration:**  
The given variable names are assigned the specified data type in the program unit. Type is one of INTEGER\*2, INTEGER\*4, REAL\*4, REAL\*8, DOUBLE PRECISION, COMPLEX\*8, LOGICAL\*4, LOGICAL\*1 or BYTE.

**DIMENSION**

Reserves storage space for the specified array(s).

**COMMON**

Reserves one or more blocks of storage space under the specified name to contain the variables associated with the block name.

**EQUIVALENCE**

Declares two or more variable names in the same program unit to be associated with the same storage location.

**EXTERNAL**

Permits the use of external procedures (functions, subroutines and FORTRAN library functions) as arguments to other subprograms. It informs the system that the names specified are those of routines not contained in the current program unit.

**DATA**

Permits the assignment of initial values to variables and array elements prior to program execution.

**PROGRAM**

Assigns a name to a main program unit. If present, it is the first statement in the main program.

**User-Written Subprograms**

**name(var1,var2, . . . )=  
expression**

**Arithmetic Statement Function:**  
Creates a user-defined function having the

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

|                   |  |
|-------------------|--|
|                   | variables as dummy arguments. When referenced, the expression is evaluated using the actual arguments in the function call.                    |
| <b>FUNCTION</b>   | Begins a <b>FUNCTION</b> subprogram, indicating the program name and any dummy variable names. An optional type specification can be included. |
| <b>SUBROUTINE</b> | Begins a <b>SUBROUTINE</b> subprogram, indicating the program name and any dummy variable names.   |
| <b>BLOCK DATA</b> | Specifies the subprogram which follows as a <b>BLOCK DATA</b> subprogram. An optional name for the program unit may be given.                  |

**Fortran Library Functions**

|                          |                                   |
|--------------------------|-----------------------------------|
| <b>ABS(X)</b>            | Real absolute value               |
| <b>IABS(X)</b>           | Integer absolute value            |
| <b>DABS(X)</b>           | Double Precision absolute value   |
| <b>CABS(Z)</b>           | Complex to Real, absolute value   |
| <b>FLOAT(I)</b>          | Integer to Real conversion        |
| <b>IFIX(X)</b>           | Real to Integer conversion        |
| <b>SNGL(X)</b>           | Double to Real conversion         |
| <b>DBLE(X)</b>           | Real to Double conversion         |
| <b>REAL(Z)</b>           | Complex to Real conversion        |
| <b>AIMAG(Z)</b>          | Complex to Real conversion        |
| <b>CMPLX(X,Y)</b>        | Real to Complex conversion        |
| <b>AINT(X)</b>           | Real to Real truncation           |
| <b>INT(X)</b>            | Real to Integer conversion        |
| <b>IDINT(X)</b>          | Double to Integer conversion      |
| <b>AMOD(X,Y)</b>         | Real remainder                    |
| <b>MOD(I,J)</b>          | Integer remainder                 |
| <b>DMOD(I,J)</b>         | Double Precision remainder        |
| <b>AMAXO(I,J, . . .)</b> | Real maximum from Integer list    |
| <b>AMAX1(I,J, . . .)</b> | Real maximum from Real list       |
| <b>MAXO(I,J, . . .)</b>  | Integer maximum from Integer list |
| <b>MAX1(X,Y, . . .)</b>  | Integer maximum from Real list    |
| <b>DMAX1(X,Y, . . .)</b> | Double maximum from Double list   |
| <b>AMINO(I,J, . . .)</b> | Real minimum of Integer list      |
| <b>AMIN1(X,Y, . . .)</b> | Real minimum of Real list         |
| <b>MINO(I,J, . . .)</b>  | Integer minimum of Integer list   |
| <b>MIN1(X,Y, . . .)</b>  | Integer minimum of Real list      |
| <b>DMIN1(X,Y, . . .)</b> | Double minimum from Double list   |
| <b>SIGN(X,Y)</b>         | Real transfer of sign             |
| <b>ISIGN(I,J)</b>        | Integer transfer of sign          |
| <b>DSIGN(X,Y)</b>        | Double Precision transfer of sign |
| <b>DIM(X,Y)</b>          | Real positive difference          |
| <b>IDIM(I,J)</b>         | Integer positive difference       |

**Table 7-2 FORTRAN IV Language Summary (Cont.)**

|             |   |
|-------------|---|
| EXP(X)      | e raised to the X power (X is Real)         |
| DEXP(X)     | e raised to the X power (X is Double)       |
| CEXP(Z)     | e raised to the Z power (Z is Complex)      |
| ALOG(X)     | Returns the natural log of X (X is Real)    |
| ALOG10(X)   | Returns the log base 10 of X (X is Real)    |
| DLOG(X)     | Returns the natural log of X (X is Double)  |
| DLOG10(X)   | Returns the log base 10 of X (X is Double)  |
| CLOG(Z)     | Returns the natural log of Z (Z is Complex) |
| SQRT(X)     | Square root of Real argument                |
| DSQRT(X)    | Square root of Double Precision argument    |
| CSQRT(Z)    | Square root of Complex argument             |
| SIN(X)      | Real sine                                   |
| DSIN(X)     | Double Precision sine                       |
| CSIN(Z)     | Complex sine                                |
| COS(X)      | Real cosine                                 |
| DCOS(X)     | Double Precision cosine                     |
| CCOS(Z)     | Complex cosine                              |
| TANH(X)     | Hyperbolic tangent                          |
| ATAN(X)     | Real arctangent                             |
| DATAN(X)    | Double Precision arctangent                 |
| ATAN2(X,Y)  | Real arctangent of (X/Y)                    |
| DATAN2(X,Y) | Double Precision arctangent of (X/Y)        |
| CONJG(Z)    | Complex conjugate                           |
| RAN(I,J)    | Returns a random number between 0 and 1     |

### **7.3 COMPILER OPERATION**

The FORTRAN IV compiler accepts a source written in the FORTRAN language as input and produces an object file and a listing file as output. The object file must be subsequently processed by the operating system's linker program, for example, the Linker or Task Builder, to produce an executable program.

#### **Command String Specification Options**

In the input/output file specification command string issued to the FORTRAN IV compiler to request program compilation, the user can specify a number of switch parameter options. Some of the parameters are:

#### **SPECIFY LISTING OPTIONS**

The user can request a number of listing options. By default, the user is supplied with diagnostics (if any), a source program listing, and the storage map. In addition, the user can request a generated code listing, or can combine any of the listing options in a single listing. The generated code listing contains a symbolic representation of the object code generated by the compiler, including a location offset from the base of the program unit, the symbolic Object Time System (OTS) routine names, and routine arguments. The code generated for each statement is labelled with the same internal sequence number that appears in the source program listing, for easy cross reference.

#### **COMPILE DEBUGGING STATEMENT LINES**

The user can request the compiler to include in the compilation those lines with a "D" in column one. These statements allow the inclusion of programmer-selected debugging aids (see below).

#### ENABLE/DISABLE THE COMMON SUBEXPRESSION OPTIMIZER

In general, the optimizer will make the program run faster. Disabling the optimizer can reduce program storage requirements, but will increase execution time.

#### INCLUDE OR SUPPRESS INTERNAL SEQUENCE NUMBERS

Suppressing internal sequence number accounting reduces program storage requirements for generated code and slightly increases execution time, but disables line number information during traceback.

#### ALLOCATE TWO WORDS FOR DEFAULT LENGTH OF INTEGER VARIABLES

Normally, single storage words will be the default allocation for integer variables not given an explicit length specification (i.e., INTEGER\*2 or INTEGER\*4). Only one word is used for computation. The user can request that the default allocation be two storage words.

#### ENABLE/DISABLE VECTORING OF ARRAYS

Array vectoring is a process which decreases the time necessary to reference elements of a multidimensional array by using some additional memory to store array accessing information. If array vectoring is enabled, the compiler decides whether to vector a multidimensional array based on the ratio of the amount of space required to vector the array over the total space required by the array. If this ratio is greater than 25%, the array is not vectored, and standard mapping is used instead. If size is a more critical factor than speed, the user can disable the vectoring of all arrays. If arrays are vectored, it is so noted in the storage map listing.

#### ENABLE/DISABLE COMPILER WARNING DIAGNOSTICS

Warning diagnostics report conditions which are not fatal error conditions, but which can be potentially dangerous at execution time, or which may present compatibility problems with other FORTRAN compilers running on PDP-11 operating systems. For example, a warning message is generated if a variable name exceeds six characters in length. This is potentially dangerous if another variable name has the same first six characters. Another example is that statement ordering restrictions are not imposed by the FORTRAN IV compiler, but are imposed by the FORTRAN IV-PLUS compiler. A program written for the FORTRAN IV compiler which does not conform to the FORTRAN IV-PLUS convention could not be compiled by both FORTRAN IV and FORTRAN IV-PLUS compilers. The warning diagnostics are normally enabled, but the user can suppress their inclusion in the diagnostics listing.

#### Internal Operation and Structure

Instead of using temporary files to process source programs, the FORTRAN IV compiler performs all its activities in main memory. It reads the entire source program once, stores it in memory in a compacted format, and processes the compacted code in memory. The advantage to this method is that the FORTRAN IV compiler does not require a disk device for its operation. In addition, since a disk device is not used for temporary file operations, compilation speed is significantly increased.

To reduce the memory requirements of such a compilation system, the FORTRAN IV compiler employs a multi-phase overlaid structure. The com-

piler consists of a large number of overlays, each of which occupies no more than 1K to 1.25K words of memory. Most of the space allocated to the compiler is occupied by the compressed source code. Figure 7-1 illustrates the compile-time memory map.

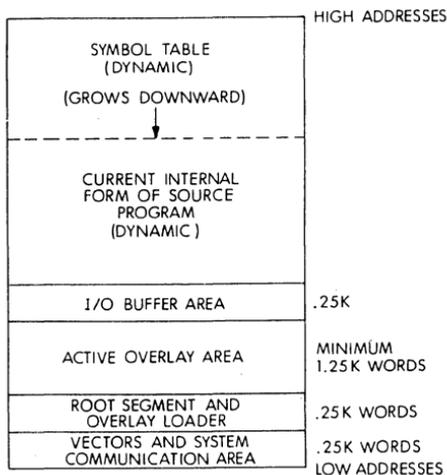


Figure 7-1 Compile-Time Memory Map

The compiler goes through a series of processing phases, one for each of its 18-20 overlays. Each program segment is processed separately, generally using the entire sequence of overlay phases. The basic processing phases are:

1. Source program compaction and listing
2. Syntax analysis and error reporting
3. Non-executable statement processing
4. Code generation
5. Code optimization

The compiler begins by reading in as much of the source subprogram as it can fit in memory. It then compresses the source code in memory by removing blanks and other unnecessary data. It continues to read in more source code, compressing it as it goes, until the entire program segment fits in memory.

Once the source code is compacted into memory, the compiler begins processing the internal form of the source code as a whole. Because the entire program segment is available to the compiler at a glance, FORTRAN IV does not require statement ordering restrictions.

During the first stage of code generation, the compiler immediately writes as much information as possible to the object file. This step is necessary to further compress the internal source code to enable the symbol table to grow in the later stages of processing.

The non-executable statements are eligible for immediate processing, since the information they provide is not needed until run-time. Therefore, the compiler searches for all the occurrences of non-executable statements, such as FORMAT and DATA statements, produces the beginning of the object module, and compacts the internal source code further.

To begin the next stage of code generation, the compiler enters all variables and constants not yet processed into the symbol table, and performs the syntax scan of the executable statements. The program is translated into an internal format in preparation for final code generation.

### **Object Code Generation**

A few executable FORTRAN statements can be translated directly into machine instructions. Typical FORTRAN operations, however, require long sequences of PDP-11 machine instructions. For example, standard sequences are needed to locate an element of a multidimensional array, initialize an I/O operation, or simulate a floating-point operation not supported by the hardware configuration.

The common sequences of PDP-11 machine instructions are contained in a library known as the FORTRAN Object Time System (OTS). The FORTRAN IV compiler does not actually generate pure machine instructions for the FORTRAN source code statements. It simply determines which combination of appropriate OTS routines are needed to implement a FORTRAN program. During the linking process for an object program, the linker utility includes the needed OTS routines into the load module. During program execution, these routines are chained together to effect the desired result.

The compiler references a library instruction sequence by generating a word containing the address of the first instruction in the OTS routine, followed by the information upon which the routine is to operate (the operands). For example, an OTS routine used to perform the end-of-DO-loop sequence must be passed the location of the index variable, the limit value, and the address of the beginning of the loop.

The compiler and OTS make use of the PDP-11 general register and indirect addressing facility to have the OTS routines executed at run-time. Register 4 (R4) is used to chain together the selected OTS routines. The last instruction executed in each library routine is a JMP @(R4)+, which transfers control to the next library instruction sequence.

### **Optimizations**

The FORTRAN IV compiler performs the following optimizations during code generation:

#### **1. Compiled FORMAT Statements**

The compiler interprets the FORMAT statements at compile-time, translating the format into an internal form. This not only increases the execution speed of the program, it decreases its size, because less run-time code is needed.

#### **2. Array Vectoring**

Array vectoring decreases the time necessary to reference elements of a multidimensional array by using additional memory to store the array. If an array is vectored, a particular element in the array can be

located by a simplified mapping function, without the need for multiplication operations. A table lookup is performed to determine the location of a particular element.

3. Constant Folding  
Integer constant expressions are evaluated at compile-time.
4. Compile-time Evaluation of Constant Subscript Expressions  
Constant subscript expressions in array calculations are evaluated at compile-time.
5. Elimination of Unreachable Code  
Statements that are never reached by flow of control are eliminated from the object code.
6. Common Subexpression Elimination  
Redundant subexpressions whose operands are not changed between computations are replaced by a temporary value calculated only once.
7. Peephole Optimizations  
The compiler examines the internal form of the object code on an operation-by-operation basis to replace sequences of operations with shorter and faster equivalent operations. For example, the compiler replaces a divide-by-two operation with a multiply-by-one-half operation. There are a large set of these kinds of operations.
8. Branch Optimizations for Arithmetic and Logical IF  
Branch structure optimizations improve program speed and decrease its size. For example, an arithmetic IF statement can often be improved:

```
IF(A-7.0)100, 200, 100 !goto 200 if A is equal to 7.0  
100 CONTINUE
```

The compiler will optimize this statement to:

```
IF(A .EQ. 7.0) GOTO 200
```

### Libraries

The FORTRAN programmer can create a library of commonly-used assembly language and FORTRAN functions and subroutines. The operating system's librarian utility provides a library creation and modification capability. Library files may be included in the command string to the linker utility. The linker recognizes the file as a library file and links only those routines in the library that are required in the executable program. By default, the linker also automatically searches the FORTRAN system library for any other required routines.

### Debugging a FORTRAN Program

Two debugging facilities are available to the FORTRAN programmer. The FORTRAN Object Time System provides the traceback feature for fatal run-time errors. This feature locates the actual program unit and line number of a run-time error. Immediately following the error message, the error handler will list the line number and program unit name in which the error occurred. If the program unit is a subroutine or function subprogram, the error handler will trace back to the calling program unit and display the name of that program unit and the line number where the call occurred. This process will continue until the calling sequence has been traced back to a specific line number in the main program. This

allows the exact determination of the location of an error even if the error occurs in a deeply nested subroutine.

In addition to the FORTRAN OTS error diagnostics which include the traceback feature, there is another debugging tool available. A "D" in column one of a FORTRAN statement allows that statement to be conditionally compiled. These statements are considered comment lines by the compiler unless the appropriate debugging lines switch is issued in the compiler command string. In this case, the lines are compiled as regular FORTRAN statements. Liberal use of the PAUSE statement and selective variable printing can provide the programmer with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program testing stages and later eliminated without source program modification.

#### **7.4 FORTRAN IV OPERATING ENVIRONMENTS**

The FORTRAN IV compiler and OTS is available as an optional language processor for the RT-11 operating system.

The operating system provides additional features particular to the environment. For example, the monitor programmed requests or executive directives are usually available as a library of FORTRAN-callable routines.

##### **Under RT-11**

The RT-11 System Subroutine Library (SYSLIB) is a collection of FORTRAN-callable routines which allow a FORTRAN user to utilize various features of the RT-11 Foreground/Background (F/B) and Single-Job monitors. SYSLIB also provides various utility functions, a complete character string manipulation package, and 2-word integer support. SYSLIB is provided as a library of object modules to be combined with FORTRAN programs at link-time.

##### **SUMMARY OF SYSLIB CAPABILITIES**

- Complete RT-11 I/O facilities, including synchronous, asynchronous, and event-driven modes of operation. FORTRAN subroutines may be activated on the completion of an input/output operation.
- Timed scheduling of asynchronous subjobs (completion routines). (Under F/B operation only.)
- Complete facilities for interjob communication. (Under F/B operation only.)
- FORTRAN interrupt service routines.
- Complete timer-support facilities, including timed suspension of execution (F/B only), conversion of different time formats, and time-of-day information. These timer facilities support either 50 or 60-cycle clocks.
- All auxiliary input/output functions provided by RT-11, including the capabilities of opening, closing, renaming, creating, and deleting files from any device.
- All monitor level informational functions, such as job partition parameters, device statistics, and input/output channel statistics.
- Access to the RT-11 Command String Interpreter (CSI) for acceptance and parsing of standard RT-11 command strings.

- A character string manipulation package supporting variable-length character strings.
- INTEGER\*4 support routines that allow two-word integer computations.

SYSLIB allows the RT-11 FORTRAN user to write almost all application programs in FORTRAN with no assembly-language coding.

Also available under RT-11 is the Scientific Subroutine Library, providing FORTRAN-language routines for mathematical and statistical applications.



## CHAPTER 8

# BASIC

### 8.1 FUNCTION AND FEATURES

BASIC is an incremental compiler which provides immediate translation and storage of a program written in the BASIC source language while it is being entered. A single-user BASIC system and a RT-11 multi-user BASIC system are available as an option for the RT-11 operating system.

BASIC provides the following features:

- incremental compiler for immediate source translation
- immediate mode for ease in debugging and use as a desk calculator
- ASCII sequential files compatible with FORTRAN
- integer, string and floating point virtual array files for random access
- dynamic allocation of string storage
- PRINT-USING statement for output formatting
- complete set of string manipulation functions
- user-defined functions
- programs chained together can pass data through common
- CALL statement for assembly language subroutines

### 8.2 LANGUAGE

The BASIC language is a conversational programming language which uses simple English-type statements and familiar mathematical notation to perform operations. BASIC is one of the simplest computer languages to learn, and once learned, provides advanced techniques to perform intricate data manipulations and efficient problem expression.

A BASIC program is composed of lines of statements containing instructions to the BASIC compiler. Each line of the program begins with a number that identifies that line as a statement and indicates the order of statement execution relative to other lines in the program. Each statement starts with an English word specifying the type of operation to be performed.

All BASIC statements and computations must be written on a single line. Statements cannot be continued on a following line. More than one statement, however, can be written on a single line when each statement after the first is preceded by a backslash. For example,

```
10 INPUT A,B,C
```

is a single statement line, while

```
20 LET X = 11 \ PRINT X,Y,Z \ IF X = A THEN 10
```

is a multiple statement line containing three statements: LET, PRINT, and IF.

### Constants and Variables

BASIC treats all numbers (real and integer) as decimal numbers. The advantage of treating all numbers as decimal numbers is that any number or symbol can be used in any mathematical expression without regard to its type. Numbers used must be in the approximate range  $10^{-38}$  to  $10^{+38}$ .

In addition to real and integer formats, BASIC accepts exponential notation. Numeric data can be input in any one or all of these formats. BASIC automatically uses the most efficient format for printing a number, according to its size. It automatically suppresses leading and trailing zeros in integer and decimal numbers and formats all exponential numbers.

Both floating point and integer formats are used when storing and calculating numbers. If a number can be stored as an integer, it is handled as an integer unless the operation requires that it be stored as floating point.

BASIC also processes information in the form of strings. A string is a sequence of alphabetic, numeric or special characters treated as a unit. A string constant is a list of characters enclosed in quotes. A string constant can be used in the PRINT, CALL, and CHAIN statements. These usages of string constants are allowed in versions of BASIC that do not support strings.

In BASIC with string support, string constants can also be used to assign a value to a string variable, for example, in the LET and INPUT statements.

BASIC recognizes four types of variables: numeric and subscripted numeric, string and subscripted string. A numeric variable is an algebraic symbol representing a number and is formed by a single letter or letter optionally followed by a single digit. For example: I, B3, or X.

Subscripted variables provide additional computing capabilities for dealing with lists, tables, matrices, or any set of related variables. In BASIC, variables are allowed one or two subscripts. For example, a list might be described as A(I) where I goes from 0 to 5:

```
A(0), A(1), A(2), A(3), A(4), A(5).
```

This allows reference to each of the six elements in the list, and can be considered a one-dimensional algebraic matrix. Two-dimensional matrices are also allowed.

Any variable name followed by a dollar sign (\$) character indicates a string variable. For example: A\$, C7\$. Any list or matrix variable name followed by the dollar sign character denotes the string form of that variable. For example: V\$(n), M2\$(n), C\$(m,n), G1\$(m,n).

The user can assign values to variables by indicating the values in a LET statement, by entering the value as data in an INPUT statement, or by a READ statement. The value assigned to variable does not change until the next time a statement is encountered that contains a new value for that variable.

### Operators

BASIC performs addition, subtraction, multiplication, division and exponentiation. The five operators used in writing most familiar formulas are:

|   |       |                                      |
|---|-------|--------------------------------------|
| + | A + B | Add to B and A                       |
| - | A - B | Subtract B from A                    |
| * | A * B | Multiply A by B                      |
| / | A / B | Divide A by B                        |
| ↑ | A ↑ B | Raise A to the B <sup>th</sup> power |

In addition, BASIC allows unary plus and minus arithmetic operators. For strings, the concatenation operator (+ or &) puts one string after another without any intervening characters.

Relational operators allow comparison of two values and are used to compare arithmetic expressions or strings in an IF-THEN statement. The relational operators are:

|    |  |
|----|--|
| =  | Equals (alphabetically equal)              |
| <  | Less than (alphabetically precedes)        |
| <= | Less than or equals (precedes or equals)   |
| >  | Greater than (alphabetically follows)      |
| >= | Greater than or equals (follows or equals) |
| <> | Not equals (not alphabetically equal)      |

### Statements

The following summary of BASIC statements gives a brief explanation of each statement's use.

|       |   |
|-------|---|
| REM   | Contains explanatory comments in a BASIC program.   |
| LET = | Assigns the value of an expression to the specified variable. Variable and expression must be of the same type.               |
| DIM   | Reserves space in memory for arrays according to the subscripts specified.  |
| DATA  | Used in conjunction with READ to input listed data into an executing program. Can contain any mixture of strings and numbers. |

|                                       |   |
|---------------------------------------|---|
| READ                                  | Assigns values listed in DATA statements to the specified values. Variables can be numeric or string.   |
| OPEN FOR<br>INPUT [OUTPUT]<br>AS FILE | Opens a file for input (or output) and associates the file with the specified logical unit number.  |
| INPUT                                 | Reads data from the file associated with the logical unit specified or from the user's terminal. Variables can be arithmetic or string.   |
| IF END                                | Tests for an end-of-file condition on input sequential file associated with logical unit expression.  |
| PRINT                                 | Prints the values of the specified expressions on the terminal or, when specified, to the file associated with the logical unit expression. The TAB function can also be included.  |
| PRINT-USING                           | Prints the values of the specified expression on the terminal or, when specified, to the file associated with logical unit expression in the format determined by the given string. Both numeric and string expressions can be used.                                      |
| RESTORE                               | Resets either the data pointer or, when specified, the input file associated with the given logical unit number to the beginning.   |
| RESET                                 | Equivalent to RESTORE.  |
| CLOSE                                 | Closes the file(s) associated with the logical unit number(s) and virtual file logical unit number(s) specified. If no logical unit number is specified, closes all open files.   |
| NAME-TO                               | Renames the specified file.   |
| KILL                                  | Deletes the specified file.   |
| RANDOMIZE                             | Causes the random number generator (RND function) to produce different random numbers every time the program is run.  |
| DEF FN                                | Defines a user function.  |
| CALL                                  | Used to call assembly language subroutines from a BASIC program.  |
| FOR-TO                                | Sets up a loop to be executed the specified number of times.  |
| NEXT                                  | Placed at the end of the FOR loop to return control to the FOR statement.   |
| IF                                    | Conditionally executes the specified statement or transfers control to the specified line number. When the condition is not satisfied, execution continues at the next sequential line. The expressions and the relational operator must all be string or all be numeric. |
| GOSUB                                 | Unconditionally transfers control to specified line of subroutine.  |

|          |   |
|----------|---|
| RETURN   | Terminates a subroutine and returns control to the statement following the last executed GOSUB statement.   |
| GO TO    | Unconditionally transfers control to specified line number.   |
| ON-GOSUB | Conditionally transfers control to the subroutine at one line number specified in the list. The value of the expression determines the line number to which control is transferred.                                   |
| ON-GO TO | Conditionally transfers control to one line number in the specified list. The value of the expression determines the line number to which control is transferred.   |
| CHAIN    | Terminates execution of the program, loads the program specified, and begins execution of the lowest line number or, when a line number is present in the statement, at the specified line number.                    |
| COMMON   | Preserves values and names of specified variables and arrays when the CHAIN statement is executed. Both string and arithmetic variables and arrays can be passed. The statement also dimensions the specified arrays. |
| END      | Placed at the end of the physical end of the program to terminate execution (optional).   |
| STOP     | Terminates execution of the program. Placed at the logical end of the program.  |

### Functions

BASIC provides a variety of functions to perform mathematical and string operations.

#### ARITHMETIC FUNCTIONS

|       |  |
|-------|--|
| ABS   | Returns the absolute value of an expression.   |
| ATN   | Returns the arctangent as an angle in radians.   |
| COS   | Returns the cosine of an expression in radians.  |
| EXP   | Returns the value of the constant e (approx. 2.71828) raised to a given power (expression).          |
| INT   | Returns the greatest integer less than or equal to a given expression.                               |
| LOG   | Returns the natural logarithm of an expression.  |
| LOG10 | Returns the base 10 logarithm of an expression.  |
| PI    | Returns the value of pi (3.141593 approx.)   |
| RND   | Returns a random number between 0 and 1.   |
| SGN   | Returns value indicating the sign of an expression.  |
| SIN   | Returns the sine of an expression in radians.  |
| SQR   | Returns the square root of an expression.  |
| TAB   | Causes the terminal print head to tab to column number given by an expression (valid only in PRINT). |
| SYS   | Special system function calls; control terminal I/O and perform special functions.                   |

## STRING FUNCTIONS

|       |  |
|-------|--|
| ASC   | Returns the ASCII code in decimal for the 1-character string expression.   |
| BIN   | Converts a string expression containing a binary number to a decimal value.  |
| CHR\$ | Generates a 1-character string whose ASCII value is the low-order 8 bits of the integer value of the given expression. |
| DAT\$ | Returns the date as a string.  |
| LEN   | Returns the number of characters in the given string.  |
| OCT   | Converts a string expression containing an octal number to a decimal value.  |
| POS   | Searches for and returns the position of the first occurrence of a substring in a string.                              |
| SEG\$ | Returns the string of characters in the given positions in the string.   |
| STR\$ | Returns the string which represents the numeric value of the given expression.   |
| TRM\$ | Returns the given string without trailing blanks.  |
| VAL   | Returns the value of the decimal number contained in the given string expression.                                      |

## USER-DEFINED FUNCTIONS

In some programs it may be necessary to execute the same sequence of statements in several different places. BASIC allows definition of unique operations or expressions and the calling of these functions in the same way as, for example, the square root or trigonometric functions. Each function is defined once and can appear anywhere in the program.

A function definition consists of the function name, a dummy variable list (up to five), and an expression.

When the user-defined function is used in the program, the expressions in the argument list passed to the function will replace the dummy variables in the defining expression. Any variable in the defining expression that is not in the dummy variable list will have the value that the variable is currently assigned.

### Programming Example

The POS function is used to find the position of a substring in a string. The POS function can be used to map a string of characters to a corresponding integer value which can be used for subsequent processing. This technique is called a table look-up. The table string is the first argument of the POS function and the string to be mapped is the second argument. For example:

```
LISTNH
10  REM PROGRAM TO TRANSLATE MONTH NAMES
15  REM TO NUMBERS
20  T$ = "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
100 PRINT "TYPE THE FIRST 3 LETTERS OF A MONTH";
110 INPUT M$
120 IF LEN(M$) <> 3 GO TO 200
130 M = (POS(T$,M$,1) +2)/3
```

```

140 REM CHECK IF MONTH IS SPELLED CORRECTLY
150 IF M < > INT(M) GO TO 200
160 PRINT M$ " IS MONTH NUMBER " M
170 GO TO 100
200 PRINT "INVALID ENTRY—TRY AGAIN" \ GO TO 100
READY
RUNNH
TYPE THE FIRST 3 LETTERS OF A MONTH? NOV
NOV IS MONTH NUMBER 11
TYPE THE FIRST 3 LETTERS OF A MONTH? DEC
DEC IS MONTH NUMBER 12
TYPE THE FIRST 3 LETTERS OF A MONTH? JUN
INVALID ENTRY—TRY AGAIN
TYPE THE FIRST 3 LETTERS OF A MONTH?↑C
STOP AT LINE 110
READY

```

### 8.2.1 BASIC Files

Data is stored either in sequential files or in random access, virtual array files. Sequential files are treated in the same way as terminal I/O; data is read by an INPUT statement and written by a PRINT statement. Sequential files are useful for storing data that is processed serially.

Virtual array files are similar to arrays stored in memory. An element of data in a virtual array can be part of any BASIC expression just the same as an element of a normal array. An element of a virtual array file can be assigned a value by a special form of the LET statement. Virtual array files allow data to be accessed in a random, non-serial manner and are the only BASIC files in which existing data can be updated without re-writing the entire file.

There are three data types for virtual array files: integer, floating point, and string. A file can contain only one data type.

### 8.2.2 Creating, Modifying and Executing BASIC Programs

A BASIC program is entered in the system using the editing commands. Once a program has been entered, it can be saved, retrieved, listed or executed using the editing commands. When the BASIC system is running, it prints the message READY on the terminal to indicate that it is ready to accept an editing command.

The BASIC system's editor is a replacement editor. That is, an incorrect line is changed by entering a new line with the same line number as the incorrect line. The editor replaces the old line with the new line entered. A line can be deleted by typing its line number and a carriage return. Both the line number and the line are removed from the program. The following provides a summary of the BASIC editing commands.

**NEW** Clears the user area in memory and assigns a specified name to the current program. Used to create a new program.

|         |   |
|---------|---|
| LIST    | Types on the terminal the program currently in memory.  |
| LISTNH  | A range of line numbers can be specified. If the command does not have the "NH" suffix, the program header is printed.  |
| RUN     | If issued with no file specification, executes the program currently in memory. If a file specification is issued, clears the user area, reads a program in from the file, and executes the program. If the command does not have the "NH" suffix, the program header is printed. |
| RUNNH   |   |
| CLEAR   | Clears the contents of the user array and string buffers. This command is used when a program has been executed and then edited. Before rerunning the program, the array and string buffers are cleared to provide more memory space.   |
| RENAME  | Changes the current program name to a specified name.   |
| SAVE    | Copies the contents of the user area to a file, lists the contents on the line printer, or punches the contents on paper tape.  |
| OLD     | Clears the user area and reads a program from a specified file into the user area in memory.  |
| REPLACE | Replaces the specified file with the program currently in memory.   |
| APPEND  | Merges the program currently in memory with a program stored in a file. All lines in the program in memory that have duplicate line numbers with the program in the file are replaced by the lines from the program in the file.  |
| LENGTH  | Displays on the terminal the amount of storage required by the BASIC program currently in memory. This information is useful in determining the minimum user area in which a specific program can run.  |
| SCR     | Erases the user area in memory.   |
| UNSAVE  | Deletes the specified file.   |
| BYE     | If the BASIC system supports multiple users, terminates the session at the terminal.  |

In addition to the editing commands, the BASIC system recognizes the following special control characters:

|        |  |
|--------|--|
| CTRL/C | Interrupts program execution and prints the READY message. |
| CTRL/O | Enables/disables console output.                           |
| CTRL/U | Deletes the current line being entered.                    |
| RUBOUT | Deletes the last character typed.                          |

### 8.3 COMPILER OPERATION

When the user enters a BASIC program, the BASIC system does not store the program exactly as it is typed or read from the input file. Instead, it translates the program into an intermediate form which can be used in two different ways. The intermediate code can be returned to its initial

form by the LIST or SAVE commands to produce an ASCII program which looks like the input program, or the translated code can be quickly interpreted by the RUN command to execute a program under the operating system.

### Immediate Mode of Execution

It is not necessary to write a complete program to use BASIC. Almost any BASIC statement can be executed in immediate mode. This latter facility makes BASIC an extremely powerful calculator.

BASIC distinguishes between those lines entered for immediate execution and those entered for later execution by the presence or absence of a line number. Statements which begin with line numbers are stored; those without line numbers are executed immediately.

Immediate mode operation is especially useful for program debugging and desk calculation problems.

To facilitate debugging a program, STOP statements can be placed throughout the program. When the program is run, each STOP statement causes the program to halt. The data values can then be examined and modified in immediate mode. The immediate mode statement

```
GO TO line number
```

is used to continue program execution. The values assigned to variables when the RUN command was issued remain intact until a SCRatch, CLEAR or another RUN command is issued.

If the STOP statement occurs in the middle of a FOR loop, modifications can not be made to the section of the program which precedes the FOR statement.

If CTRL/C is used to halt program execution, the GO TO command can be used to continue execution at the line where execution stopped.

When using immediate mode, nearly all the standard statements can be used to generate or print immediate mode results. Multiple statements can be used on a single line in immediate mode. For example:

```
A=1 \ PRINT A  
1
```

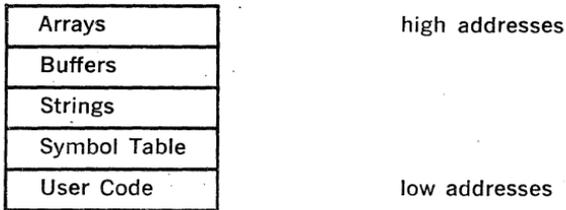
Program loops in immediate mode are allowed in multiple statement lines. Thus a table of square roots can be produced as follows:

```
FOR I=1 TO 10 \ PRINT I,SQR(I) \ NEXT I  
1          1  
2          1.41421  
3          1.73205  
4          2  
5          2.23607  
6          2.44949  
7          2.64575  
8          2.82843  
9          3  
10         3.16228
```

Certain statements, while not illegal, make no sense when used in immediate mode, such as COMMON, DEF, DIM, DATA and RANDOMIZE. The INPUT statement is illegal in immediate mode. Also, function references in immediate mode are illegal unless the program containing the definition was previously executed.

### User Area Allocation and Program Size

The BASIC system stores each user's program in memory in the following format:



The symbol table and user code area are created when the program is entered. When the RUN command is issued, the user program is scanned and arrays are set up. The string area is created during program execution.

The SCRatch command clears all the user code, symbol table, strings, and arrays from memory. The CLEAR command clears the arrays and strings but does not affect the user code or symbol table.

A symbol table entry is created for each distinct line number or variable name referenced in the program. These entries are not deleted, however, even when all references in the program to a particular line number or variable are removed. Thus, if the program in memory is heavily modified, it may be desirable to save it with the SAVE command and then restore the program with the OLD command to obtain the largest possible user area.

The LENGTH command displays on the terminal the amount of storage required by the BASIC program in memory. This information is useful in determining the minimum user area in which a specific program can run.

LENGTH prints the number of words used and the number of words remaining free in the user's area. The LENGTH command always returns the current memory requirements; they may differ depending on when the command is issued. The number of words in use includes memory currently needed by the BASIC program itself, arrays, string variables, and file buffers in the user area. To determine the size of the program alone, issue the LENGTH command immediately after an OLD or CLEAR command. Arrays are created after the RUN command is issued and file buffers are created when the OPEN statement is executed. The memory required for string variables and string arrays varies with the current values of the strings.

## 8.4 BASIC ENVIRONMENTS

Two BASIC versions are available for the RT-11 operating system: the standard single-user version and a special multi-user version.

The following paragraphs discuss the features and capabilities of the BASIC versions available. Table 8-1 compares the features and language elements of each version.

### Under RT-11: Single-user Version

BASIC/RT-11 interfaces with the RT-11 monitor. BASIC is loaded under control of the monitor by typing the "R BASIC" command. Users can access any RT-11 supported device, including paper tape reader/punch and floppy disk (RXV11). BASIC/RT-11 files can be processed by FORTRAN IV/RT-11. At least 8K words of memory are required to run BASIC. In systems with more than 8K words of memory, BASIC/RT-11 provides alphanumeric character string I/O and string variable support.

### Under RT-11: Multi-user version

MU BASIC/RT-11 is a multi-user BASIC system, capable of accommodating up to eight users simultaneously. Each user independently creates and executes BASIC programs. All of the features of MU BASIC, including statements, commands, functions and immediate mode execution are available to all users.

MU BASIC runs under the RT-11 monitor. Users can access the following devices supported by RT-11: paper tape reader/punch, line printer (LAV11) and floppy disk (RXV11).

Up to eight users can be supported on Single-job systems with at least 24K words of memory. Up to four users can be supported on Single-job systems with at least 16K words of memory, or on Foreground/Background systems with 28K words of memory.

To accommodate multiple users, MU BASIC provides a scheduling supervisor and terminal handler. In addition, the system provides a log on procedure and file protection as options.

The log on procedure requires that users obtain a user ID and password from the system manager to gain access to the system. The HELLO and BYE commands are used in this case to log on and log off the system.

The file protection system provides several degrees of file access. There are three classes of files: public, group, and private. Public library files are available to all users. Group library files are accessible to all users have the same first character in their user ID. A private file is accessible only to the user who created it.

If file protection is desired, a file can be given any of the following access characteristics:

|        |   |
|--------|---|
| Run    | Allows access by the RUN command or CHAIN statement.  |
| Read   | Allows access by the OLD or APPEND command or the OPEN FOR INPUT or OVERLAY statement or use of the value of an element of a virtual array. |
| Update | Allows a virtual array file to be updated.  |

**Complete** Allows access by all of the above and by the SAVE, REPLACE or UNSAVE command or the OPEN FOR OUTPUT, NAME TO or KILL statement.

A nonprivileged user is allowed complete access only to the user's own private files. A nonprivileged user can have Run and Read access to files in the public library and the user's own group library. Nonprivileged users are not allowed access to other user's private library files or other group's files. The access allowed a nonprivileged user to all files other than the user's own files can be modified by the inclusion of a protection code in the filename.

A privileged user has complete access to all files. Group library and public library files can only be created by a privileged user.

In addition to the log on procedure and file protection, MU BASIC includes the following commands:

- HELLO** Allows the user to log on to the system (optional).
- ASSIGN** Assigns a specified device to a user if it is available.
- DEASSIGN** Deassigns a specified device.
- TAPE** Disables echoing for the low-speed paper tape reader.
- KEY** Enables echoing after the TAPE command (or a disable echoing system function call).
- SET TTY** Sets the system to allow different terminal characteristics.

MU BASIC provides a comprehensive set of system functions. Certain system functions are available to all users. These functions enable the programmer to cancel CTRL/O typed at a terminal, disable/enable echoing, enter single character input mode, scratch the user area in memory and return to the READY message, and return the current user's ID. Certain other system function calls can be executed only by a privileged user. These functions include the ability to disable the CTRL/C interrupt, set the user ID, terminate the privileged user status, and cause BASIC to exit and return control to the RT-11 monitor.

The single character input mode system function call is useful for special read operations. It returns the decimal ASCII value of the next character input from the terminal or a file. It is the only method for BASIC programs to process terminal input without waiting for a carriage return to be typed. This allows interactive programs to use single character response and not require a carriage return.

Single character input mode allows data in any file to be read with no need for separating commas or carriage returns. Binary files can be copied exactly.

**Table 8-1 BASIC Language Implementations**

| Language Elements   | BASIC<br>RT    | MU<br>BASIC<br>RT |
|---|----------------|-------------------|
| <b>FUNCTIONS</b>  |                |                   |
| Mathematical (ABS, SIN, COS, SGN, LOG, EXP, ATN, INT, RND, SQR) | X              | X                 |
| String LEFT, RIGHT, MIDDLE LENGTH                               |                |                   |
| String LEN, SEG   | X <sup>1</sup> | X <sup>1</sup>    |
| String search for substring                                     | X <sup>1</sup> | X <sup>1</sup>    |
| TRM\$ (delete trailing spaces)                                  | X <sup>1</sup> | X <sup>1</sup>    |
| ASCII code of character   | X <sup>1</sup> | X <sup>1</sup>    |
| Character equivalent to ASCII code (CHR\$)                      | X <sup>1</sup> | X <sup>1</sup>    |
| Date  | X <sup>1</sup> | X <sup>1</sup>    |
| Numeric string conversion                                       | X <sup>1</sup> | X <sup>1</sup>    |
| Octal and binary functions (OCT and BIN)                        | X              | X                 |
| Common log (LOG10)  | -              | X                 |
| TAB for print positioning                                       | X              | X                 |
| Assembly language routines                                      | X              | X                 |
| <b>PROGRAM LINES</b>  |                |                   |
| Multiple statement lines  | X              | X                 |
| <b>Assignment</b>   |                |                   |
| LET   | X              | X                 |
| Word LET optional   | X              | X                 |
| <b>Control</b>  |                |                   |
| GOTO  | X              | X                 |
| IF-THEN   | X              | X                 |
| IF-GOTO   | X              | X                 |
| FOR, NEXT   | X              | X                 |
| FOR-WHILE, FOR-UNTIL  | -              | -                 |
| ON-GOTO   | -              | X                 |
| CHAIN   | X              | X                 |
| <b>Subroutines and Functions</b>                                |                |                   |
| GOSUB   | X              | X                 |
| RETURN  | X              | X                 |
| ON-GOSUB  | -              | X                 |
| DEF, single line  | X              | X                 |
| <b>I/O</b>  |                |                   |
| READ  | X              | X                 |
| DATA  | X              | X                 |
| RESTORE   | X              | X                 |
| INPUT   | X              | X                 |
| LINPUT  | -              | X                 |
| PRINT   | X              | X                 |
| PRINT USING   | -              | X                 |

**Table 8-1 BASIC Language Implementations (Cont.)**

| Language Elements                | BASIC<br>RT | MU<br>BASIC<br>RT |
|----------------------------------|-------------|-------------------|
| <b>Files</b>                     |             |                   |
| OPEN                             |             |                   |
| CLOSE                            | X           | X                 |
| NAME-AS                          | -           | -                 |
| NAME-TO                          | -           | X                 |
| KILL                             | -           | X                 |
| IF END                           | X           | X                 |
| Virtual files                    | X           | X                 |
| <b>Specifications</b>            |             |                   |
| REM                              | X           | X                 |
| DIM                              | X           | X                 |
| RANDOMIZE                        | X           | X                 |
| COMMON                           | -           | X                 |
| <b>Miscellaneous</b>             |             |                   |
| CHANGE string to/from array      | -           | -                 |
| STOP                             | X           | X                 |
| END                              | X           | X                 |
| END statement optional           | X           | X                 |
| <b>Editing</b>                   |             |                   |
| RENAME                           | X           | X                 |
| LIST                             | X           | X                 |
| LENGTH                           | -           | X                 |
| <b>Program storage/retrieval</b> |             |                   |
| NEW                              | X           | X                 |
| OLD                              | X           | X                 |
| SAVE                             | X           | X                 |
| REPLACE                          | X           | X                 |
| UNSAVE                           | -           | X                 |
| APPEND                           | -           | X                 |
| <b>Miscellaneous</b>             |             |                   |
| KEY                              | -           | X                 |
| TAPE                             | -           | X                 |
| RUN                              | X           | X                 |
| HELLO                            | -           | X                 |
| BYE                              | -           | X                 |
| Immediate Mode                   | X           | X                 |

section**5**  
dec services

**CHAPTER 1 EDUCATIONAL SERVICES**

**CHAPTER 2 DECUS**

**CHAPTER 3 MAINTENANCE**



# EDUCATIONAL SERVICES

### 1.1 GENERAL

Like DIGITAL's computer systems, training facilities span the globe—Japan, Australia, Great Britain, Germany, France, the Netherlands, Sweden, Italy, and throughout the United States. Services are centered around 14 fully equipped Regional Education Centers and a staff of seasoned educators dedicated to providing all aspects of education and training needed in support of all DIGITAL systems.

### 1.2 CATALOG COURSES

Catalog courses are regularly scheduled classes offered at training centers. Presently, there are more than 100 scheduled classes that cover the range from first-time user to highly specialized training on theory of operation. Most catalog courses include extensive hands-on laboratory time, and all incorporate the use of a broad assembly of student workbooks, reference manuals, and other instructional materials.

### 1.3 CUSTOM COURSES

Specialized training is available for users with unique applications or training situations. This approach is designed to give the student the maximum relevant material for specific applications, while minimizing extraneous information. The custom courses are tailored to the individual customer's schedule and typically comprise a series of courses. These can be modified from existing courses or be entirely new programs based on mutually agreed upon objectives.

### 1.4 ON-SITE INSTRUCTION

Customers with a group of individuals to train may find it more economical to have Educational Services conduct courses at the user's home site. On-site instruction of both catalog and custom courses eliminates travel and other expenses incurred by students attending classes at training centers. This method of instruction further enhances training by allowing DIGITAL instructors to emphasize points of particular value to the student's applications and operations.

### 1.5 AUDIO-VISUAL COURSES

By taking advantage of the latest in audio-visual techniques, Educational Services has developed a series of courses that offers independent learning. Audio-visual courses are convenient, self-contained, and modular in topic. The self-instructional format allows students to progress at their own rates, study when and where they wish, and play back modules for review. Audio-visual course material is available in several forms—videotape, videocassette, or audio/filmstrip cassette—all supported by student workbooks.

### 1.6 LSI-11 AND PDP-11/03 RELATED COURSES

DIGITAL's educational group offers a series of courses on the hardware

and software for your LSI-11, PDP-11/03 systems. A list of these courses, with a brief abstract of each follows:

- **INTRODUCTION TO MINICOMPUTERS**—This course is designed for the individual with no computer experience or the programmer with compiler-level background only. It covers computer concepts and the fundamentals of Assembly Language programming and provides background for further hardware or software training on any of our systems. It is also useful as a stand-alone overview course.
- **INTRODUCTION TO DIGITAL COMPUTER LOGIC**—This course concentrates on the internal construction and functional operation of the logic circuitry of which DIGITAL's computer systems are built. This examination of the internal building blocks used in DIGITAL's computers can be useful to anyone about to enter the field of computers. This course is a prerequisite for entering hardware familiarization or maintenance courses on other DIGITAL computers.

The course covers the operation of digital computer circuitry from a logic rather than electronic point of view.

A DIGITAL computer laboratory exercises to implement some of the logic circuits discussed.

- **LSI-11 AND PDP-11/03 HARDWARE AND INTERFACING**—This course is intended to provide design oriented engineering personnel with an overview of LSI-11 system operation and detailed interfacing information.

Emphasis will be placed on Bus timing structure and standard DIGITAL interfacing modules.

Laboratory periods will be provided for student familiarization with the LSI and 11/03 system including machine language programming and standard DIGITAL interfaces.

- **PDP-11 FUNDAMENTALS AND INSTRUCTIONS**—This course is intended for data processing personnel who will be programming for any model within the PDP-11 family. It covers the instructions and features common to all PDP-11 models. It does not cover options or features unique to the more powerful models. The course is also appropriate for a manager or supervisor requiring a fairly detailed knowledge of the PDP-11 and its instruction set.
- **MACRO-11**—This course is intended to provide the inexperienced assembly language programmer with a MACRO-11 programming background for entry to any of the following operating systems courses: DOS-11, RT-11, RSX-11M, RSX-11D, and IAS. Classroom and laboratory exercises will supplement the lectures.
- **PDP11 REAL-TIME OPERATING SYSTEM (RT-11) STANDARD**—This course is designed for users who have limited operating systems experience and wish to acquire a working knowledge of RT-11, including both Single-Job and Foreground/Background monitors. Emphasis is placed on system programs and on techniques for programming in a foreground/background real-time environment.

For complete information on course content, prerequisites, pricing, and scheduling, consult the Educational Courses Catalog available through DIGITAL's Educational Centers listed below:

**Boston area:**

Digital Equipment Corporation  
Educational Services Department  
Maynard, Massachusetts 01754  
Telephone: (617) 897-5111  
Ext. 3819 or 6331

**Chicago area:**

Digital Equipment Corporation  
Educational Services Department  
5600 Apollo Drive  
Rolling Meadows, Illinois 60008  
Telephone: (312) 640-5520

**Philadelphia area:**

Digital Equipment Corporation  
Educational Services Department  
Whitpain Office Campus  
1740 Walton Road  
Blue Bell, Pennsylvania 19422  
Telephone: (215) 825-4200 Ext. 26

**Washington, D.C. area:**

Digital Equipment Corporation  
Educational Services Department  
Lanham 30 Office Building  
5900 Princess Garden Parkway  
Lanham, Maryland 20801  
Telephone: (301) 459-7900 Ext. 315 or 215

**San Francisco area:**

Digital Equipment Corporation  
Educational Services Department  
310 Soquel Way  
Sunnyvale, California 94086  
Telephone: (408) 984-0200 Ext. 293 or 294

## CHAPTER 2

### DECUS

Additional programs and applications packages may be obtained from DECUS, the Digital Equipment Computer Users Society. DECUS is a not-for-profit computer users group (the largest such group, worldwide) that sponsors technical symposia, publishes a periodic newsletter and symposia proceedings, and maintains a library of more than 2200 programs for the various DIGITAL computers. Every customer who has purchased or ordered a computer manufactured by DIGITAL is eligible for an installation membership in DECUS. Associate membership is also available to any person with a bona fide interest in DIGITAL computers. Membership in DECUS is strictly voluntary, and does not require payment of dues. Programs from the DECUS library are available to all members for nominal reproduction and handling charges. A complete catalog of available programs may be obtained from the Society.

Further information on the DECUS Library, publications, and other DECUS activities is available from the DECUS offices listed below:

#### **AUSTRALIA AND NEW ZEALAND:**

DECUS  
P.O. Box 491  
Crows Nest, N.S.W. 2065  
Australia

#### **CANADA:**

DECUS  
P.O. Box 11500  
Ottawa, Ontario K2H 8K8  
Canada

#### **EUROPE AND MIDDLE EAST:**

DECUS  
Case Postale 340  
1211 Geneva 26  
Switzerland

#### **ALL OTHERS:**

DECUS  
146 Main Street  
Maynard, Massachusetts 01754  
U.S.A.



# MAINTENANCE

### 3.1 GENERAL

DIGITAL offers a wide range of maintenance services to LSI-11, PDP-11/03, and PDP-11V03 customers. These services are provided through DIGITAL's Customer Services Organization and have been designed to meet our customer's complete maintenance needs, either on-site or off-site. These service plans provide complete DIGITAL maintenance on-site by our factory-trained engineers, or provide module and unit repairs off-site for those customers desiring to perform their own maintenance.

### 3.2 ON-SITE SERVICE

DIGITAL's service organization provides on-site maintenance service with a staff of over 4,000 factory trained engineers in 300 locations worldwide. Each service office maintains adequate inventory to support its customers and is fully supported by our logistics operation in Maynard, Massachusetts.

- **Service Agreements**—On-site contract service is available for all PDP-11V03 systems, and PDP-11/03's subject to minimum hardware configurations. This service provides corrective maintenance, preventive maintenance, and all applicable engineering changes to ensure your products are operational and kept completely up to date. In addition to priority service, contractual maintenance allows DIGITAL customers to budget for their annual maintenance needs. The monthly contract charge covers all travel, labor, and material.
- **Per Call**—DIGITAL also offers on-site per call service. DIGITAL will respond to maintenance needs on a billable travel, time, and materials basis.
- **Installation and Warranty**—On-site installation and warranty service is also available for PDP-11V03's, and PDP-11/03's subject to minimum hardware configurations. This service must be purchased at the time of original order.

### 3.3 OFF-SITE SERVICE

DIGITAL offers complete unit and module repair services to customers capable of performing their own maintenance. The Customer Returns Area (CRA) has been established in Maynard, Massachusetts, to offer single-point interfacing for all off-site repairs for North American customers. The CRA assures the customer of complete "one-stop shopping" for all factory-level warranty and post-warranty services. All repairs are affected at our Module Repair Facility in Maynard.

For European, Australian, and Japanese customers, we have established Product Repair Centers (PRCs) in eleven countries. Customers can return defective material to the PRC in their country without the burden of customs, duties, and licensing requirements. The PRCs offer the same services to these customers as the CRA in Maynard.

For information on services in Latin and South America, contact the CRA in Maynard.

- **Warranty Service**—All products are warranted against defects in workmanship and materials under normal and proper use in their unmodified condition for a period of ninety (90) days from date of initial shipment. As a condition of this warranty, customers must obtain a DIGITAL Repair Authorization (RA) number and return the products prepaid, together with a written description of the claimed defect, to the nearest authorized DIGITAL Repair Center as listed here.

RA numbers may be obtained by contacting the CRA in Maynard (PRC if non-U.S.) and providing the following information:

1. Customer name and location.
  2. Part number/serial number of failing item.
  3. Part number/serial number of next higher assembly if a module or subassembly.
  4. Product line and date purchased.
- **Post-Warranty Service**—DIGITAL offers its post-warranty services in several forms:
    1. **Loose piece subassembly repair.** For a minimum order, customers may elect to return loose piece subassemblies and take advantage of flat rate pricing.
    2. **Prepaid module mailers.** Available on specific module types, this is the fastest repair service in the industry. Customers with current discount agreements immediately qualify and, upon payment, are assured of seven-day turnaround of their subassemblies.
    3. **Annual subassembly contracts.** A new and unique service offered only by DIGITAL, this contract compliments DIGITAL's subassembly warranty policy. For a fixed annual fee, all subassemblies used in DIGITAL supplied equipment are eligible for return and repair. This is the most cost-effective service available anywhere. DIGITAL customers realize considerable repair cost savings by performing the diagnosis of their system and returning only the failing subassembly. This service is made available only on PDP-11/03 and PDP-11V03 systems.
    4. **Firm quote product and option repair.** For the smaller customer with only occasional service needs or those who do not have any in-house troubleshooting capability, this service offers total support with no risk. Upon return and inspection, the customer is quoted a fixed fee for product repair and/or refurbishment. He may choose exactly what level of service he feels is right for his environment.

For more complete information and pricing on any of the services listed, contact the repair center nearest you.

The following repair centers have been established to provide complete off-site repair services. These centers should be contacted for all off-site warranty and post-warranty services and prices. All defective material should be set to the address indicated with your RA number appearing on the shipping label.

**North America**

Digital Equipment Corporation  
Customer Returns Area  
111 Powder Mill Road  
Maynard, Massachusetts 01754

RA Number

Telephone Number: 617-897-5111, Ext 6871

**Canada**

Digital Equipment of Canada, Ltd.  
100 Herzberg Road  
Kanata, Ontario, Canada  
ATTN: Forward to Customer Returns Area

RA Number

Telephone Number: 613-592-5111

**Europe****Austria**

Product Repair Center Manager  
Digit Equip Corp, GES. M.B.H.  
Gumpendorferstrasse 65  
P.O. Box 438  
A-1061 Wien  
Austria  
Telephone: (0222) 57-36-49

**Belgium**

Product Repair Center Manager  
Digital Equipment Sa/Nv  
108 Rue D'Arlon  
B-1040 Bruxelles, Belgium  
Telephone: (02) 13-92-56

**France**

Product Repair Center Manager  
Digital Equipment France  
2, Place Gustave Eiffel  
Cidex L225  
18, Rue Saarinen  
94 533 Rungis, France  
Telephone: (01) 687-2333

**Germany**

Product Repair Center Manager  
Digital Equipment GmbH  
D-8000 Munchen 40  
Wallensteinplatz 2  
West Germany  
Telephone: 35031

**Holland**

Product Repair Center Manager  
Digital Equipment Bv  
KaaP Horndreef 38  
P.O. Box 9064  
Utrecht, Holland  
Telephone: (030) 63 12 22

**Italy**

Product Repair Center Manager  
Digital Equipment S.P.A.  
Corso Garibaldi 49  
I-10121 Milano  
Italy  
Telephone: (02) 879051

**Sweden**

Product Repair Center Manager  
Digital Equipment AB  
Englodsvaegen 7  
S-17141 Solna  
Sweden  
Telephone: (08) 730-08-00

**Switzerland**

Product Repair Center Manager  
Digital Equipment Corp. AG/SA  
Schaffhauserstrasse 315  
CH-8050 Zurich/Oerlikon  
Switzerland  
Telephone: (01) 46 41 91

**United Kingdom**

Product Repair Center Manager  
Digital Equipment Corp., Ltd.  
Digital House, Kings Road  
Reading RG1-4HS  
England  
Telephone: (734) 58 35 55

**General International Area**

At this time, the only service offered in the GIA is firm quote product/option repair through the Tokyo and Sydney repair centers. During Fiscal Year 1977 this will be expanded to include other services and locations and will be announced separately.

## **GIA Product Repair Centers**

### **Australia**

Product Repair Center Manager  
Digital Equip. Australia Pty. Ltd.  
132-125 Willoughby Road  
P.O. Box 491  
Crows Nest  
New South Wales, 2065 Australia  
Telephone: (02) 435 2566

### **Latin America**

#### **South America**

Contact the CRA, Maynard.

### **Japan**

Product Repair Center  
Digital Equipment Corp. Int.  
#5 Shin/Taiso Building  
2-10-7 Dogenzaka, Shibuya/K.U.  
Tokyo 150, Japan  
Telephone: 404-4082



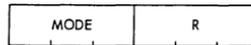
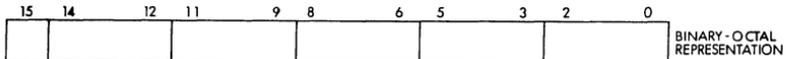
# appendices



## GENERAL REFERENCE DATA

## A.1 SUMMARY OF LSI-11 INSTRUCTIONS

## WORD FORMAT



| Mode | Name               | Symbolic | Description                  |
|------|--------------------|----------|------------------------------|
| 0    | register           | R        | (R) is operand [ex. R2 = %2] |
| 1    | register deferred  | (R)      | (R) is address               |
| 2    | auto-increment     | (R)+     | (R) is adrs; (R) +(1 or 2)   |
| 3    | auto-incr deferred | @(R)+    | (R) is adrs of adrs; (R) +2  |
| 4    | auto-decrement     | -(R)     | (R) -(1 or 2); is adrs       |
| 5    | auto-decr deferred | @-(R)    | (R) -2; (R) is adrs of adrs  |
| 6    | index              | X(R)     | (R) + X is adrs              |
| 7    | index deferred     | @X(R)    | (R) + X is adrs of adrs      |

## PROGRAM COUNTER ADDRESSING Reg = 7



|   |                   |     |                                    |
|---|-------------------|-----|------------------------------------|
| 2 | immediate         | #n  | operand n follows instr            |
| 3 | absolute          | @#A | address A follows instr            |
| 6 | relative          | A   | instr adrs + 4 + X is adrs         |
| 7 | relative deferred | @A  | instr adrs + 4 + X is adrs of adrs |

## LEGEND

| Op Codes                               | Operations                  |
|--|-----------------------------|
| ■ = 0 for word/1 for byte              | ( ) = contents of           |
| SS = source field (6 bits)             | s = contents of source      |
| DD = destination field (6 bits)        | d = contents of destination |
| R = gen register (3 bits),<br>0 to 7   | r = contents of register    |
| XXX = offset (8 bits), +127<br>to -128 | ← = becomes                 |

## Op Codes

N = number (3 bits)  
 NN = number (6 bits)

## Operations

X = relative address  
 % = register definition

## Boolean

$\wedge$  = AND  
 $\vee$  = inclusive OR  
 $\nabla$  = exclusive OR  
 $\sim$  = NOT

## Condition Codes

\* = conditionally set/cleared  
 — = not affected  
 0 = cleared  
 1 = set

### SINGLE OPERAND: OPR dst



| Mne-<br>monic | Op Code | Instruction | dst Result | N | Z | V | C |
|---------------|---------|-------------|------------|---|---|---|---|
|---------------|---------|-------------|------------|---|---|---|---|

#### General

|        |         |                    |          |   |   |   |   |
|--------|---------|--------------------|----------|---|---|---|---|
| CLR(B) | ■ 050DD | clear              | 0        | 0 | 1 | 0 | 0 |
| COM(B) | ■ 051DD | complement (1's)   | $\sim d$ | * | * | 0 | 1 |
| INC(B) | ■ 052DD | increment          | $d + 1$  | * | * | * | — |
| DEC(B) | ■ 053DD | decrement          | $d - 1$  | * | * | * | — |
| NEG(B) | ■ 054DD | negate (2's compl) | $-d$     | * | * | * | * |
| TST(B) | ■ 057DD | test               | d        | * | * | 0 | 0 |

#### Rotate & Shift

|        |         |                   |                    |   |   |   |   |
|--------|---------|-------------------|--------------------|---|---|---|---|
| ROR(B) | ■ 060DD | rotate right      | $\rightarrow C, d$ | * | * | * | * |
| ROL(B) | ■ 061DD | rotate left       | $C, d \leftarrow$  | * | * | * | * |
| ASR(B) | ■ 062DD | arith shift right | $d/2$              | * | * | * | * |
| ASL(B) | ■ 063DD | arith shift left  | $2d$               | * | * | * | * |
| SWAB   | 0003DD  | swap bytes        |                    | * | * | 0 | 0 |

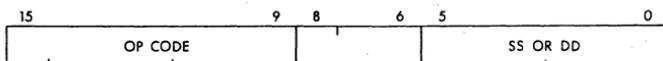
#### Multiple Precision

|        |         |                |           |   |   |   |   |
|--------|---------|----------------|-----------|---|---|---|---|
| ADC(B) | ■ 055DD | add carry      | $d + C$   | * | * | * | * |
| SBC(B) | ■ 056DD | subtract carry | $d - C$   | * | * | * | * |
| SXT    | 0067DD  | sign extend    | 0 or $-1$ | — | * | 0 | — |

#### Processor Status (PS) Operators

|      |        |                   |                   |   |   |   |   |
|------|--------|-------------------|-------------------|---|---|---|---|
| MFPS | 1067DD | move byte from PS | $d \leftarrow PS$ | * | * | 0 | — |
| MTPS | 1064SS | move byte to PS   | $PS \leftarrow s$ | * | * | * | * |

### DOUBLE OPERAND: OPR src, dst OPR src, R or OPR R, dst



| Mne-<br>monic  | Op Code | Instruction    | Operation                        | N | Z | V | C |
|----------------|---------|----------------|----------------------------------|---|---|---|---|
| <b>General</b> |         |                |                                  |   |   |   |   |
| MOV(B)         | ■ 1SSDD | move           | $d \leftarrow s$                 | * | * | 0 | - |
| CMP(B)         | ■ 2SSDD | compare        | $s - d$                          | * | * | * | * |
| ADD            | 06SSDD  | add            | $d \leftarrow s + d$             | * | * | * | * |
| SUB            | 16SSDD  | subtract       | $d \leftarrow d - s$             | * | * | * | * |
| <b>Logical</b> |         |                |                                  |   |   |   |   |
| BIT(B)         | ■ 3SSDD | bit test (AND) | $s \text{ } d$                   | * | * | 0 | - |
| BIC(B)         | ■ 4SSDD | bit clear      | $d \leftarrow (\sim s) \wedge d$ | * | * | 0 | - |
| BIS(B)         | ■ 5SSDD | bit set (OR)   | $d \leftarrow s \vee d$          | * | * | 0 | - |
| XOR            | 074RDD  | exclusive (OR) | $d \leftarrow r \vee d$          | * | * | 0 | - |

#### Optional EIS

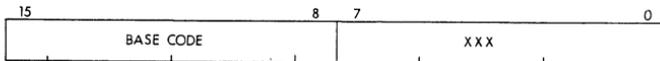
|      |        |                      |                           |   |   |   |   |
|------|--------|----------------------|---------------------------|---|---|---|---|
| MUL  | 070RSS | multiply             | $r \leftarrow r \times s$ | * | * | 0 | * |
| DIV  | 071RSS | divide               | $r \leftarrow r/s$        | * | * | * | * |
| ASH  | 072RSS | shift arithmetically |                           | * | * | * | * |
| ASHC | 073RSS | arith shift combined |                           | * | * | * | * |

#### Optional FIS

|      |        |                   |  |   |   |   |   |
|------|--------|-------------------|--|---|---|---|---|
| FADD | 07500R | floating add      |  | * | * | 0 | 0 |
| FSUB | 07501R | floating subtract |  | * | * | 0 | 0 |
| FMUL | 07502R | floating multiply |  | * | * | 0 | 0 |
| FDIV | 07503R | floating divide   |  | * | * | 0 | 0 |

**BRANCH:** B—location

If condition is satisfied  
 Branch to location,  
 $\text{New PC} \leftarrow \text{Updated PC} + (2 \times \text{offset})$   
 adrs of br instr + 2



Op Code = Base Code + XXX

| Mne-<br>monic   | Base<br>Code | Instruction             | Branch Condition |
|-----------------|--------------|-------------------------|------------------|
| <b>Branches</b> |              |                         |                  |
| BR              | 000400       | branch (unconditional)  | (always)         |
| BNE             | 001000       | br if not equal (to 0)  | $\neq 0$ $Z = 0$ |
| BEQ             | 001400       | br if equal (to 0)      | $= 0$ $Z = 1$    |
| BPL             | 100000       | branch if plus          | $+$ $N = 0$      |
| BMI             | 100400       | branch if minus         | $-$ $N = 1$      |
| BVC             | 102000       | br if overflow is clear | $V = 0$          |
| BVS             | 102400       | br if overflow is set   | $V = 1$          |

| Mne-<br>monic                        | Base<br>Code | Instruction                      | Branch Condition                   |
|--------------------------------------|--------------|----------------------------------|------------------------------------|
| BCC                                  | 103000       | br if carry is clear             | $C = 0$                            |
| BCS                                  | 103400       | br if carry is set               | $C = 1$                            |
| <b>Signed Conditional Branches</b>   |              |                                  |                                    |
| BGE                                  | 002000       | br if greater or<br>equal (to 0) | $\geq 0$ $N \oplus V = 0$          |
| BLT                                  | 002400       | br if less than (0)              | $< 0$ $N \oplus V = 1$             |
| BGT                                  | 003000       | br if greater than (0)           | $> 0$ $Z \vee (N \oplus V) = 0$    |
| BLE                                  | 003400       | br if less or equal (to 0)       | $\leq 0$ $Z \vee (N \oplus V) = 1$ |
| <b>Unsigned Conditional Branches</b> |              |                                  |                                    |
| BHI                                  | 101000       | branch if higher                 | $>$ $C \vee Z = 0$                 |
| BLOS                                 | 101400       | branch if lower or same          | $\leq$ $C \vee Z = 1$              |
| BHIS                                 | 103000       | branch if higher or same         | $\geq$ $C = 0$                     |
| BLO                                  | 103400       | branch if lower                  | $<$ $C = 1$                        |

### JUMP & SUBROUTINE

| Mne-<br>monic | Op Code | Instruction                       | Notes  |
|---------------|---------|-----------------------------------|--|
| JMP           | 0001DD  | jump                              | PC $\leftarrow$ dst  |
| JSR           | 004RDD  | jump to subroutine                | } use same R   |
| RTS           | 0002OR  | return from<br>subroutine         |  |
| MARK          | 0064NN  | mark                              | aid in subr return   |
| SOB           | 077RNN  | subtract 1 & br<br>(if $\neq 0$ ) | (R) $- 1$ , then if (R) $\neq 0$ :<br>PC $\leftarrow$ Updated PC $-$<br>(2 x NN) |

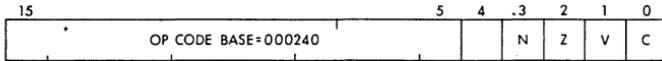
### TRAP & INTERRUPT:

| Mne-<br>monic | Op Code             | Instruction                            | Notes              |
|---------------|---------------------|--|--------------------|
| EMT           | 104000<br>to 104377 | emulator trap<br>(not for general use) | PC at 30, PS at 32 |
| TRAP          | 104400<br>to 104777 | trap                                   | PC at 34, PS at 36 |
| BPT           | 000003              | breakpoint trap                        | PC at 14, PS at 16 |
| IOT           | 000004              | input/output trap                      | PC at 20, PS at 22 |
| RTI           | 000002              | return from interrupt                  |                    |
| RTT           | 000006              | return from interrupt                  | inhibit T bit trap |

### MISCELLANEOUS:

| Mnemonic | Op Code | Instruction        |
|----------|---------|--------------------|
| HALT     | 000000  | halt               |
| WAIT     | 000001  | wait for interrupt |
| RESET    | 000005  | reset external bus |
| NOP      | 000240  | (no operation)     |

## CONDITION CODE OPERATORS:



0 = CLEAR SELECTED COND. CODE BITS  
 1 = SET SELECTED COND. CODE BITS

| Mnemonic | Op Code | Instruction       | N | Z | V | C |
|----------|---------|-------------------|---|---|---|---|
| CLC      | 000241  | clear C           | - | - | - | 0 |
| CLV      | 000242  | clear V           | - | - | 0 | - |
| CLZ      | 000244  | clear Z           | - | 0 | - | - |
| CLN      | 000250  | clear N           | 0 | - | - | - |
| CCC      | 000257  | clear all cc bits | 0 | 0 | 0 | 0 |
| SEC      | 000261  | set C             | - | - | - | 1 |
| SEV      | 000262  | set V             | - | - | 1 | - |
| SEZ      | 000264  | set Z             | - | 1 | - | - |
| SEN      | 000270  | set N             | 1 | - | - | - |
| SCC      | 000277  | set all cc bits   | 1 | 1 | 1 | 1 |

## A.2 NUMERICAL OP CODE LIST

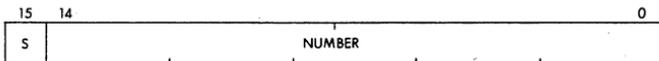
| Op Code  | Mnemonic     | Op Code  | Mnemonic   | Op Code  | Mnemonic |
|----------|--------------|----------|------------|----------|----------|
| 00 00 00 | HALT         | 00 60 DD | ROR        | 10 40 00 | } EMT    |
| 00 00 01 | WAIT         | 00 61 DD | ROL        |          |          |
| 00 00 02 | RTI          | 00 62 DD | ASR        | 10 43 77 |          |
| 00 00 03 | BPT          | 00 63 DD |            |          | } TRAP   |
| 00 00 04 | IOT          | 00 64 NN | MARK       |          |          |
| 00 00 05 | RESET        | 00 67 DD | SXT        | 10 44 00 |          |
| 00 00 06 | RTT          |          |            |          |          |
| 00 00 07 | {(unused)}   | 00 70 00 | {(unused)} | 10 47 77 |          |
| 00 00 77 |              |          |            |          |          |
| 00 01 DD | JMP          | 00 77 77 |            | 10 50 DD | CLRB     |
| 00 02 0R | RTS          |          |            | 10 51 DD | COMB     |
|          |              | 01 SS DD | MOV        | 10 52 DD | INCB     |
| 00 02 10 | } (reserved) | 02 SS DD | CMP        | 10 53 DD | DECB     |
|          |              | 03 SS DD | BIT        | 10 54 DD | NEGB     |
|          |              | 04 SS DD | BIC        | 10 55 DD | ADCB     |
|          |              | 05 SS DD | BIS        | 10 56 DD | SBCB     |
| 00 02 27 |              | 06 SS DD | ADD        | 10 57 DD | TSTB     |
| 00 02 40 | NOP          |          |            |          |          |
|          |              | 07 0R SS | MUL        | 10 60 DD | RORB     |
| 00 02 41 | } cond codes | 07 1R SS | DIV        | 10 61 DD | ROLB     |
|          |              | 07 2R SS | ASH        | 10 62 DD | ASRB     |
|          |              | 07 3R SS | ASHC       | 10 63 DD | ASLB     |
|          |              | 07 4R DD | XOR        | 10 64 SS | MTPS     |
| 00 02 77 |              |          |            | 10 67 DD | MFPS     |
| 00 03 DD | SWAB         | 07 50 0R | FADD       |          |          |
|          |              | 07 50 1R | FSUB       | 11 SS DD | MOVB     |

| Op Code   | Mne-<br>monic | Op Code   | Mne-<br>monic | Op-Code  | Mne-<br>monic   |  |
|-----------|---------------|-----------|---------------|----------|-----------------|--|
| 00 04 XXX | BR            | 07 50 2R  | FMUL          | 12 SS DD | CMPB            |  |
| 00 10 XXX | BNE           | 07 50 3R  | FDIV          | 13 SS DD | BITB            |  |
| 00 14 XXX | BEQ           |           |               | 14 SS DD | BICB            |  |
| 00 20 XXX | BGE           | 07 50 40  | } (unused)    | 15 SS DD | BISB            |  |
| 00 24 XXX | BLT           |           |               | 16 SS DD | SUB             |  |
| 00 30 XXX | BGT           |           |               |          |                 |  |
| 00 34 XXX | BLE           | 07 67 77  |               |          |                 |  |
| 00 4R DD  | JSR           | 07 7R NN  | SOB           | 17 00 00 | } RE-<br>SERVED |  |
| 00 50 DD  | CLR           | 10 00 XXX | BPL           |          |                 |  |
| 00 51 DD  | COM           | 10 04 XXX | BMI           | 17 77 77 |                 |  |
| 00 52 DD  | INC           | 10 10 XXX | BHI           |          |                 |  |
| 00 53 DD  | DEC           | 10 14 XXX | BLOS          |          |                 |  |
| 00 54 DD  | NEG           | 10 20 XXX | BVC           |          |                 |  |
| 00 55 DD  | ADC           | 10 24 XXX | BVS           |          |                 |  |
| 00 56 DD  | SBC           | 10 30 XXX | BCC,<br>BHIS  |          |                 |  |
| 00 57 DD  | TST           |           | BCS,<br>BLO   |          |                 |  |
|           |               | 10 34 XXX |               |          |                 |  |

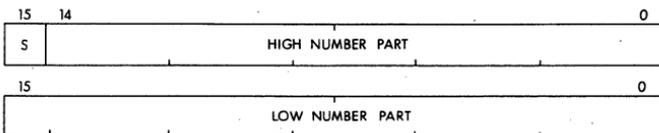
### A.3 EIS/FIS OPERAND FORMATS

#### EIS Data (Fixed Point)

16-bit single word:



32-bit double word:

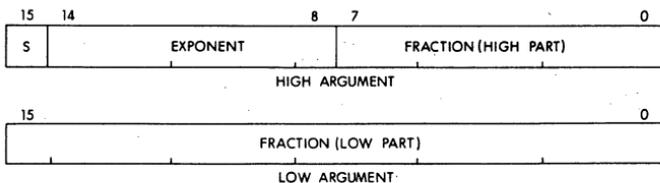


S is the sign bit.

S = 0 for positive quantities

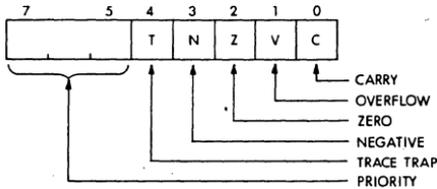
S = 1 for negative quantities; number is in 2's complement notation

#### FIS Data (Floating Point)



S = sign of fraction; 0 for positive, 1 for negative  
 Exponent = 8 bits for the exponent, in excess (200), notation  
 Fraction = 23 bits plus 1 hidden bit (all numbers are assumed to be normalized)

#### A.4 PROCESSOR STATUS WORD



#### A.5 ABSOLUTE LOADER

Starting

Address: -500

Memory Size:

|     |     |   |
|-----|-----|---|
| 4K  | 017 | Relocation Software Switch Register Address = 516       |
| 8K  | 037 | (eg.: relocation switch register address in a 4K system |
| 12K | 057 | = @17516)   |
| 16K | 077 |   |
| 20K | 117 |   |
| 24K | 137 |   |
| 28K | 157 |   |

#### A.6 RESERVED TRAP AND INTERRUPT VECTORS

|     |  |
|-----|--|
| 000 | (Reserved)   |
| 004 | Bus Timeout and Illegal Instructions (eg. JMP R0) (Odd Address and Stack Overflow Traps Not Implemented on LSI-11) |
| 010 | Illegal and Reserved Instruction   |
| 014 | BPT Instruction and T Bit  |
| 020 | IOT Instruction  |
| 024 | Power Fail   |
| 030 | EMT Instruction  |
| 034 | TRAP Instruction   |
| 060 | Console Input Device   |
| 064 | Console Output Device  |
| 100 | External Event Line Interrupt  |
| 200 | LAV11  |
| 244 | FIS (Optional)   |
| 264 | RXV11  |
| 300 | Floating Vectors start here  |

## A.7 RXVII BOOTSTRAPS

### Full Length Version

@1000/000000 12702 <LF>  
001002/000000 1002n7 <LF>\*  
001004/000000 12701 <LF>  
001006/000000 177170 <LF>  
001010/000000 130211 <LF>  
001012/000000 1776 <LF>  
001014/000000 112703 <LF>  
001016/000000 7 <LF>  
001020/000000 10100 <LF>  
001022/000000 10220 <LF>  
001024/000000 402 <LF>  
001026/000000 12710 <LF>  
001030/000000 1 <LF>  
001032/000000 6203 <LF>  
001034/000000 103402 <LF>  
001036/000000 112711 <LF>  
001040/000000 111023 (LF)  
001042/000000 30211 <LF>  
001044/000000 1776 <LF>  
001046/000000 100756 <LF>  
001050/000000 103766 <LF>  
001052/000000 105711 <LF>  
001054/000000 100771 <LF>  
001056/000000 5000 <LF>  
001060/000000 22710 <LF>  
001062/000000 240 <LF>  
001064/000000 1347 <LF>  
001066/000000 122702 <LF>  
001070/000000 247 <LF>  
001072/000000 5500 <LF>  
001074/000000 5007 <CR>

### Abbreviated Version (DRIVE 0 ONLY):

@1000/000000 5000 <LF>  
001002/000000 12701 <LF>  
001004/000000 177170 <LF>  
001006/000000 105711 <LF>  
001010/000000 1776 <LF>  
001012/000000 12711 <LF>  
001014/000000 3 <LF>  
001016/000000 5711 <LF>  
001020/000000 1776 <LF>  
001022/000000 100405 <LF>  
001024/000000 105711 <LF>  
001026/000000 100004 <LF>  
001030/000000 116120 <LF>  
001032/000000 2 <LF>  
001034/000000 770 <LF>  
001036/000000 0 <LF>  
001040/000000 5007 <CR>

\*n = 4 for Unit 0

n = 6 for Unit 1

<LF> = Line Feed

<CF> = Carriage Return

Starting address = 1000

## A.8 DEVICE REGISTER ADDRESSES

| Device                                | Registers  | Device Address                  | Interrupt Vector |
|---------------------------------------|--|---------------------------------|------------------|
|                                       | Line Time Clock<br>(external event)<br>interrupt |                                 | 100              |
| Console Terminal                      |  |                                 |                  |
| Input Control/Status                  | RCSR   | 177560                          | 60               |
| Input Buffer                          | RBUF   | 177562                          |                  |
| Output Control/Status                 | XCSR   | 177564                          | 64               |
| Output Buffer                         | XBUF   | 177566                          |                  |
| LAV11 High-Speed<br>Printer           |  |                                 |                  |
| Printer Status                        |  | 177514                          | 200              |
| Printer Buffer                        |  | 177516                          |                  |
| High-Speed Paper Tape<br>Reader/Punch |  |                                 |                  |
| Reader Status                         |  | 177550                          | 70               |
| Reader Buffer                         |  | 177552                          |                  |
| Punch Status                          |  | 177554                          | 74               |
| Punch Buffer                          |  | 177556                          |                  |
| RXV11 Floppy Disk<br>System           |  |                                 |                  |
| Status                                | RXCS   | 177170                          | 264              |
| Buffer                                | RXDB   | 177172                          |                  |
| REV11 ROM Programs                    |  | 165000-165776,<br>173000-173776 |                  |

## A.9 CONSOLE ODT COMMANDS

| Format     | Octal Code        | Description  |
|------------|-------------------|--|
| RETURN     | 015               | Close opened location and accept next command.   |
| LINE FEED  | 012               | Close current location; open next sequential location.                                     |
| ↑ or ]     | 135               | Open previous location.  |
| ← or —     | 137               | Take contents of opened location, index by opened location plus 2, and open that location. |
| @          | 100               | Take contents of opened location as an absolute address and open that location.            |
| r/         | 057               | Open location r.   |
| /          | 057               | Open last location.  |
| \$n or Rn  | 044 or 122        | Open general register n (0-7) or S (PS register).  |
| r; G or rG | 073 107 or<br>107 | Go to location r, initialize the bus, and start program.                                   |

| Format           | Octal Code     | Description  |
|------------------|----------------|--|
| nL               | 114            | Execute bootstrap loader using n as device CSR address.  |
| ;P or P          | 073 120 or 120 | Proceed with program execution.  |
| RUBOUT or DELeTe | 177            | Erase previous character. Response is a backslash \ (134) each time RUBOUT is entered.   |
| M                | 115            | Maintenance. Display of an internal CPU register follows the M command. Only the last digit displayed is significant, indicating how the CPU entered the Halt (ODT) mode, as follows.<br>Last<br>Digit                      Halt Source<br>0 or 4    HALT instruction or BHALT L bus signal asserted.<br>1 or 5    Bus error occurred while getting device interrupt vector.<br>2 or 6    Bus error occurred while doing memory refresh.<br>3        Double bus error occurred (stack was non-existent value).<br>4        Reserved instruction trap occurred (non-existent Micro-PC address occurred on internal CPU bus).<br>7        A combination of 1, 2, and 4 occurred. |
| CTRL-SHIFT-S     | 023            | For manufacturing tests only. Escape this command function by typing NULL and @ (000 and 100).   |

---

## A.10 REV11-A, REV11-C COMMANDS/OPERATION

| Command   | Function   |
|-----------|--|
| \$OD      | Halt processor; system responds to console ODT commands.   |
| \$XM <CR> | Execute memory diagnostic program. Program result in displaying:<br>\$            Pass condition, or<br><u>xxxxxx</u> Halt (fail) condition<br><u>@</u>            |
| \$XC <CR> | Execute processor diagnostic program. Program execution results in displaying:<br>\$            Pass condition,<br><u>xxxxxx</u> Halt (fail) condition<br><u>@</u> |

**Command**

\$AL <CR>, or  
\$ALdddddd <CR>

\$AR <CR>, or  
\$ARdddddd <CR>

\$DX <CR>, or  
\$DXn <CR>

**Function**

Execute XC, XM, and absolute loader programs using console device (default) or device CSR = ddddd. Successful load results in automatic program start or program halt at 165626.

Execute XC, XM, and absolute loader programs for relocated loading operation using console device (default) or device CSR = ddddd. The program halts and allows the relocation address (nnnnn) bias to be entered into the software switch register (R4) as follows; restart program execution by entering P command:

@R4/xxxxxx nnnnn <CR>

@P

Successful load results in automatic program start or program halt with 165412 display.

Execute RXV11 floppy disk system bootstrap for disk 0 (default) or disk n (0 or 1).

**NOTES**

1. \$ is the prompt character for all REV11-A and REV11-C commands.
  2. <CR> is a carriage RETURN (octal code = 015) command delimiter required by all commands except OD.
  3. REV11-A and REV11-C starting address is 173000, resulting in non-memory modifying processor diagnostic test execution. Successful completion results in the \$ prompt character being displayed.
-

## A.11 7-BIT ASCII CODE

| Octal Code | Char | Octal Code | Char | Octal Code | Char   | Octal Code | Char |
|------------|------|------------|------|------------|--------|------------|------|
| 000        | NUL  | 040        | SP   | 100        | @      | 140        | \    |
| 001        | SOH  | 041        | !    | 101        | A      | 141        | a    |
| 002        | STX  | 042        | "    | 102        | B      | 142        | b    |
| 003        | ETX  | 043        | #    | 103        | C      | 143        | c    |
| 004        | EOT  | 044        | \$   | 104        | D      | 144        | d    |
| 005        | ENQ  | 054        | %    | 105        | E      | 145        | e    |
| 006        | ACK  | 046        | &    | 106        | F      | 146        | f    |
| 007        | BEL  | 047        | '    | 107        | G      | 147        | g    |
| 010        | BS   | 050        | (    | 110        | H      | 150        | h    |
| 011        | HT   | 051        | )    | 111        | I      | 151        | i    |
| 012        | LF   | 052        | *    | 112        | J      | 152        | j    |
| 013        | VT   | 053        | +    | 113        | K      | 153        | k    |
| 014        | FF   | 054        | ,    | 114        | L      | 154        | l    |
| 015        | CR   | 055        | .    | 115        | M      | 155        | m    |
| 016        | SO   | 056        | .    | 116        | N      | 156        | n    |
| 017        | SI   | 057        | /    | 117        | O      | 157        | o    |
| 020        | DLE  | 060        | 0    | 120        | P      | 160        | p    |
| 021        | DC1  | 061        | 1    | 121        | Q      | 161        | q    |
| 022        | DC2  | 062        | 2    | 122        | R      | 162        | r    |
| 023        | DC3  | 063        | 3    | 123        | S      | 163        | s    |
| 024        | DC4  | 056        | 4    | 124        | T      | 164        | t    |
| 025        | NAK  | 065        | 5    | 125        | U      | 165        | u    |
| 026        | SYN  | 066        | 6    | 126        | V      | 166        | v    |
| 027        | ETB  | 067        | 7    | 127        | W      | 167        | w    |
| 030        | CAN  | 070        | 8    | 130        | X      | 170        | x    |
| 031        | EM   | 071        | 9    | 131        | Y      | 171        | y    |
| 032        | SUB  | 072        | :    | 132        | Z      | 172        | z    |
| 033        | ESC  | 073        | ;    | 133        | [      | 173        | {    |
| 034        | FS   | 074        | <    | 134        | \      | 174        |      |
| 035        | GS   | 075        | =    | 135        | ] or ↑ | 175        | }    |
| 036        | RS   | 076        | >    | 136        | ^      | 176        | ~    |
| 037        | US   | 077        | ?    | 137        | - or ← | 177        | DEL  |

## A.12 LSI-11 BUS PIN ASSIGNMENTS

| Row A<br>(Same as Row C)       |         | Row B<br>(Same as Row D) |          |
|--------------------------------|---------|--------------------------|----------|
| Module Side 1 (Component Side) |         |                          |          |
| AA1                            | BSPARE1 | BA1                      | BDCOK H  |
| AB1                            | BSPARE2 | BB1                      | BPOK H   |
| AC1                            | BAD16   | BC1                      | SSPARE4  |
| AD1                            | BAD17   | BD1                      | SSPARE5  |
| AE1                            | SSPARE1 | BE1                      | SSPARE6  |
| AF1                            | SSPARE2 | BF1                      | SSPARE7  |
| AH1                            | SSPARE3 | BH1                      | SSPARE8  |
| AJ1                            | GND     | BJ1                      | GND      |
| AK1                            | MSPAREA | BK1                      | MSPARE B |
| AL1                            | MSPAREA | BL1                      | MSPARE B |
| AM1                            | GND     | BM1                      | GND      |
| AN1                            | BDMR L  | BN1                      | BSACK L  |
| AP1                            | BHALT L | BP1                      | BSPARE6  |
| AR1                            | BREF L  | BR1                      | BEVNT L  |
| AS1                            | PSPARE3 | BS1                      | PSPARE4  |
| AT1                            | GND     | BT1                      | GND      |
| AU1                            | PSPARE1 | BU1                      | PSPARE2  |
| AV1                            | +5B     | BV1                      | +5       |
| Module Side 2 (Solder Side)    |         |                          |          |
| AA2                            | +5      | BA2                      | +5       |
| AB2                            | -12     | BB2                      | -12      |
| AC2                            | GND     | BC2                      | GND      |
| AD2                            | +12     | BD2                      | +12      |
| AE2                            | BDOUT L | BE2                      | BDAL2 L  |
| AF2                            | BRPLY L | BF2                      | BDAL3 L  |
| AH2                            | BDIN L  | BH2                      | BDAL4 L  |
| AJ2                            | BSYNC L | BJ2                      | BDAL5 L  |
| AK2                            | BWTBT L | BK2                      | BDAL6 L  |
| AL2                            | BIRQ L  | BL2                      | BDAL7 L  |
| AM2                            | BIAKI L | BM2                      | BDAL8 L  |
| AN2                            | BIAKO L | BN2                      | BDAL9 L  |
| AP2                            | BBS7 L  | BP2                      | BDAL10 L |
| AR2                            | BDMGI L | BR2                      | BDAL11 L |
| AS2                            | BDMGO L | BS2                      | BDAL12 L |
| AT2                            | BINIT L | BT2                      | BDAL13 L |
| AU2                            | BDALO L | BU2                      | BDAL14 L |
| AV2                            | BDAL1 L | BV2                      | BDAL15 L |



## INSTRUCTION TIMING

**B.1 LSI-11 INSTRUCTION EXECUTION TIME**

The execution time for an instruction depends on the instruction itself, the modes of addressing used, and the type of memory referenced. In most cases the instruction execution time is the sum of a Basic Time, a Source Address (SRC) Time, and a Destination Address (DST) Time.

$$\text{INSTR TIME} = \text{Basic Time} + \text{SRC Time} + \text{DST Time}$$

$$(\text{BASIC Time} = \text{Fetch Time} + \text{Decode Time} + \text{Execute Time})$$

Some of the instructions require only some of these times. All timing information is in microseconds, unless otherwise noted. Times are typical; process timing can vary  $\pm 20$  percent. A 350ns microcycle is assumed.

**B.2 SOURCE AND DESTINATION TIME**

| MODE | SRC TIME<br>(Word) | SRC TIME<br>(Byte) | DST TIME<br>(Word) | DST TIME<br>(Byte) |
|------|--------------------|--------------------|--------------------|--------------------|
| 0    | 0                  | 0                  | 0                  | 0                  |
| 1    | 1.40 $\mu\text{s}$ | 1.05 $\mu\text{s}$ | 2.10 $\mu\text{s}$ | 1.75 $\mu\text{s}$ |
| 2    | 1.40               | 1.05               | 2.10               | 1.75               |
| 3    | 3.50               | 3.15               | 4.20               | 4.20               |
| 4    | 2.10               | 1.75               | 2.80               | 2.45               |
| 5    | 4.20               | 3.85               | 4.90               | 4.90               |
| 6    | 4.20               | 3.85               | 4.90               | 4.55               |
| 7    | 6.30               | 5.95               | 6.65               | 7.00               |

NOTE FOR MODE 2 and MODE 4 if R6 or R7 used with Byte operation, add 0.35  $\mu\text{s}$  to SRC time and 0.70  $\mu\text{s}$  to DST time.

**B.3 BASIC TIME**

| DOPS (Double Operand) | DM0                | DM1-7              |
|-----------------------|--------------------|--------------------|
| MOV                   | 3.50 $\mu\text{s}$ | 2.45 $\mu\text{s}$ |
| ADD,XOR,SUB,BIC,BIS   | 3.50               | 4.20               |
| CMP,BIT               | 3.50               | 3.15               |
| MOVB                  | 3.85               | 3.85               |
| BICB,BISB             | 3.85               | 3.85               |
| CMP,BITB              | 3.15               | 2.80               |

**NOTE**

DM0 = Destination Mode 0

DM1-7 = Destination Modes 1 through 7

| SOPS (Single Operand) | DMO          | DM1-7        |
|-----------------------|--------------|--------------|
| CLR                   | 3.85 $\mu$ s | 4.20 $\mu$ s |
| INC,ADC,DEC,SBC       | 4.20         | 4.90         |
| COM,NLG               | 4.20         | 4.55         |
| ROL,ASL               | 3.85         | 4.55         |
| TST                   | 4.20         | 3.85         |
| ROR                   | 5.25         | 5.95         |
| ASR                   | 5.60         | 6.30         |
| CLRB,COMB,NEGB        | 3.85         | 4.20         |
| ROLB,ASLB             | 3.85         | 4.20         |
| INCB,DECB,SBCB,ADC    | 3.85         | 4.55         |
| TSTB                  | 3.85         | 3.50         |
| RORB                  | 4.20         | 4.90         |
| ASRB                  | 4.55         | 5.95         |
| SWAB                  | 4.20         | 3.85         |
| SXT                   | 5.95         | 6.65         |
| MFPS (1067DD)         | 4.90         | 6.65         |
| MTPS (1064SS)         | 7.00         | 7.00 *       |

\* For MTPS use Byte DST time not SRC time.

\* Add 0.35  $\mu$ s to instr. time if Bit 7 of effective OPR = 1

| JMP/JSR MODE | DST TIME     |
|--------------|--------------|
| 1            | 0.70 $\mu$ s |
| 2            | 1.40         |
| 3            | 1.75         |
| 4            | 1.40         |
| 5            | 2.45         |
| 6            | 2.45         |
| 7            | 4.20         |

| INSTRUCTION          | BASIC TIMES                     |
|----------------------|---------------------------------|
| JMP                  | 3.50 $\mu$ s                    |
| JSR (PC = LINK)      | 5.25                            |
| JSR (PC $\neq$ LINK) | 8.40                            |
| ALL BRANCHES         | 3.50 (CONDITION MET OR NOT MET) |
| SOB (BRANCH)         | 4.90                            |
| SOB (NO BRANCH)      | 4.20                            |
| SET CC               | 3.50                            |
| CLEAR CC             | 3.50                            |
| NOP                  | 3.50                            |
| RTS                  | 5.25                            |
| MARK                 | 11.55                           |
| RTI                  | 8.75 *                          |
| RTT                  | 8.75 * +                        |

| INSTRUCTION          | BASIC TIMES                                 |
|----------------------|---|
| TRAP,EMT             | 16.80 * $\mu$ S                             |
| IOT,RPT              | 18.55 *                                     |
| WAIT                 | 6.30  |
| HALT                 | 5.60  |
| RESET                | 5.95 + 10.0 $\mu$ S for INIT + 90.0 $\mu$ S |
| MAINT INST. (00021R) | 20.30                                       |
| RSRVD INST. (00022N) | 5.95 (TO GET TO $\mu$ ADDRESS 3000)         |

\* If NEW PS HAS BIT 4 or BIT 7 SET ADD 0.35  $\mu$ S FOR EACH  
+ IF NEW PS HAS BIT 4 (T BIT) SET ADD 2.10  $\mu$ S

#### B.4 EXTENDED ARITHMETIC (KEV11) INSTRUCTION TIMES

##### EIS Instruction Times

| MODE | SRC TIME     |
|------|--------------|
| 0    | 0.35 $\mu$ S |
| 1    | 2.10         |
| 2    | 2.80         |
| 3    | 3.15         |
| 4    | 2.80         |
| 5    | 3.85         |
| 6    | 3.85         |
| 7    | 5.60         |

| INSTRUCTION  | BASIC TIME                                      |
|--------------|---|
| MUL          | 24.0 to 37.0 $\mu$ S<br>64.0 $\mu$ S Worst Case |
| DIV          | 78.0 $\mu$ S Worst Case                         |
| ASH (RIGHT)  | 10.1 + 1.75 per shift                           |
| ASH (LEFT)   | 10.8 + 2.45 per shift                           |
| ASHC (RIGHT) | 10.1 + 2.80 per shift                           |
| ASHC (LEFT)  | 10.1 + 3.15 per shift                           |

If both numbers less than 256 in absolute value  
16 bit multiply

##### FIS Instruction Times (us)

INST. TIME = BASIC TIME + SHIFT TIME FOR BINARY POINTS + SHIFT TIME FOR NORMALIZATION

| INSTRUCTION | BASIC TIME   |
|-------------|--------------|
| FADD        | 42.1 $\mu$ S |
| FSUB        | 42.4         |

| EXPONENT DIFFERENCE | ALIGN BINARY POINTS           |
|---------------------|-------------------------------|
| 0- 7                | 2.45 $\mu$ s per shift        |
| 8-15                | 3.50 + 2.45 per shift over 8  |
| 16-23               | 7.00 + 2.45 per shift over 16 |

| EXPONENT DIFFERENCE | NORMALIZATION               |
|---------------------|-----------------------------|
| 0- 7                | 2.1 $\mu$ s per shift       |
| 8-15                | 2.1 + 2.1 per shift over 8  |
| 16-23               | 4.2 + 2.1 per shift over 16 |

| INSTRUCTION BASIC TIME ( $\mu$ s) |  |
|-----------------------------------|--|
| FMUL                              | 74.2 to 80.9 $\mu$ s if either argument has only 7 bits of precision, i.e., the second word of the 32 bit argument is 0.<br>121.1 $\mu$ s worst case (i.e., arguments have more than 7 bits of precision). |
| FDIV                              | 151 $\mu$ s typical<br>232 $\mu$ s worst case  |

### B.5 DMA (DIRECT MEMORY ACCESS) LATENCY

DMA latency, which is the time from request (BDMRL) to bus mastership for the first DMA device, is 5.25  $\mu$ s, maximum. This time is the longest processor DATIO cycle which occurs for an ASR instruction with destination modes of 1 through 7. DMA requests are honored during memory refresh by the processor.

### B.6 INTERRUPT LATENCY (ALL TIMES IN MICROSECONDS)

- a. If processor is performing memory refresh (regardless whether KEV11 is present):
 

|  |                   |
|--|-------------------|
| Time from interrupt request (BIRQ L) to acknowledgement (BIAK L)                 | 118 $\mu$ s max   |
| Time from acknowledgement (BIAK L) to fetch of first service routine instruction | 16.5 $\mu$ s max  |
| Total time from request to first service routine instruction                     | 134.5 $\mu$ s max |
  
- b. If processor is not performing memory refresh (and KEV11 not present):
 

|   |                   |
|---|-------------------|
| Time from interrupt request (BIRQ L) to acknowledgement (BIAK L) (Longest instruction is IOT) | 18.55 $\mu$ s max |
| Time from acknowledgement (BIAK L) to fetch of first service routine instruction              | 16.5 $\mu$ s max  |
| Total time from request to first service routine instruction                                  | 35.05 $\mu$ s max |

- c. If processor is not performing memory refresh and KEV11 option is present:

|  |                  |
|--|------------------|
| Time from interrupt request (BIRQ L) to acknowledgement (BIAK L)                 | 27.6 $\mu$ s max |
| Time from acknowledgement (BIAK L) to fetch of first service routine instruction | 16.5 $\mu$ s max |
| Total time from request to first service routine instruction                     | 44.1 $\mu$ s max |

#### NOTE

During all KEV11 instructions (EIS and FIS), device and event interrupt requests are periodically scanned. If present, the instruction is aborted and all processor state information is backed up to the beginning of the instruction. After the interrupt is processed, the KEV11 instruction is re-executed from the beginning. Caution should be observed with the frequency of event interrupts; if the frequency is too high, the KEV11 instruction will never complete. It is suggested a maximum frequency of 3.3 kHz be used on the event input if the KEV11 option is present. Without the KEV11, the maximum frequency should not exceed 20 kHz. Both times allow approximately 50  $\mu$ s for the interrupt service routine.



**APPENDIX C**

**LSI-11, PDP-11 PROGRAMMING/HARDWARE  
DIFFERENCE LIST**

|    | Activity<br>ALL = Assembly Level Language<br>HLL = High Level Language<br>HO = Hardware Only   | Applicable to<br>Programming<br>Language Levels |     |    | Applicable to<br>the PDP-11 |        |       |       |       |    |
|----|--|---|-----|----|-----------------------------|--------|-------|-------|-------|----|
|    |  | ALL   | HLL | HO | NOTE                        | LS1-11 | 05/10 | 15/20 | 35/40 | 45 |
| 1. | OPR %R, (R)+ or OPR %R, -(R) using the same register as both source and destination: contents of R are incremented (decremented) by 2 before being used as the source operand.     | X   |     |    |                             |        |       | X     | X     |    |
|    | OPR %R, (R)+ or OPR %R, -(R) using the same register as both register and destination: initial contents of R are used as the source operand.                                       | X   |     |    |                             | X      | X     |       |       | X  |
| 2. | OPR %R, @ (R)+ or OPR %R, @ -(R) using the same register as both source and destination: contents of R are incremented (decremented) by 2 before being used as the source operand. | X   |     |    |                             |        |       | X     | X     |    |
|    | OPR %R, @ (R)+ or OPR %R, @ -(R) using the same register as both source and destination: initial contents of R are used as the source operand.                                     | X   |     |    |                             | X      | X     |       |       | X  |
| 3. | OPR PC, X(R); OPR PC, X(R); OPR PC, @ A: OPR PC, A: location A will contain the PC of OPR +4.  | X   |     |    |                             |        |       | X     | X     |    |
|    | OPR PC, X(R); OPR PC, @ X(R); OPR PC, A: OPR PC, @ A: location A will contain the PC of OPR +2.  | X   |     |    |                             | X      | X     |       |       | X  |

|    |  |   |  |   |   |   |     |
|----|--|---|--|---|---|---|-----|
| 4. | JMP (R)+ or JSR reg, (R)+: contents of R are incremented by 2, then used as the new PC address.  | X |  | X | X |   |     |
|    | JMP (R)+ or JSR reg, (R)+: initial contents of R are used as the new PC.   | X |  | X |   |   | X X |
| 5. | JMP %R or JSR reg, %R traps to 4 (illegal instruction).  | X |  | X | X | X | X   |
|    | JMP %R or JSR reg, %R traps to 10 (illegal instruction).   | X |  |   |   |   | X   |
| 6. | SWAB does <b>not</b> change V.   | X |  |   |   | X |     |
|    | SWAB clears V.   | X |  | X | X |   | X X |
| 7. | Register addresses (177700—177717) are valid program addresses when used by CPU.   | X |  |   | X |   |     |
|    | Register addresses (177700—177717) timeout when used as a program address by the CPU. Can be addressed under console operation. Note addresses cannot be addressed under console for LSI-11. | X |  | X |   | X | X X |
| 8. | <b>Basic instructions</b> noted in PDP-11 processor handbook.  | X |  | X | X | X | X   |
|    | SOB, MARK, RTT, SXT instructions.  | X |  | X |   |   | X X |
|    | ASH, ASHC, DIV, MUL, SPL instructions.   | X |  |   |   |   | X   |



|     |   |   |  |   |   |   |     |
|-----|---|---|--|---|---|---|-----|
|     | Power fail acts the same as 11/45 (22 ms with about 300 ns minimum). Power fail during RESET fetch is fatal with no power down sequence.      | X |  |   | X |   |     |
|     | Reset instruction consists of 10 $\mu$ s of INIT followed by a 90 $\mu$ s pause. Power fail not recognized until the instruction is complete. | X |  | X |   |   |     |
| 10. | No RTT instruction.   | X |  |   | X | X |     |
|     | If RTT sets the T bit, the T bit trap occurs after the instruction following RTT.   | X |  | X |   |   | X X |
| 11. | If RTI sets "T" bit, "T" bit trap is acknowledged after instruction following RTI.  | X |  |   | X | X |     |
|     | If RTI sets "T" bit, "T" bit trap is acknowledged immediately following RTI.  | X |  | X |   |   | X X |
| 12. | If an interrupt occurs during an instruction that has the "T" bit set, the "T" bit trap is acknowledged before the interrupt.                 | X |  | X | X | X | X   |
|     | If an interrupt occurs during an instruction and the "T" bit is set, the interrupt is acknowledged before "T" bit trap.                       | X |  |   |   |   | X   |
| 13. | "T" bit trap will sequence out of WAIT instruction.   | X |  |   | X | X | X   |
|     | "T" bit trap will not sequence out of WAIT instruction. Waits until an interrupt.   | X |  | X |   |   | X   |

|     | Activity   | Applicable to Programming Language Levels |     |    |      | Applicable to the PDP-11 |       |       |       |    |
|-----|--|---|-----|----|------|--------------------------|-------|-------|-------|----|
|     |  | ALL                                       | HLL | HO | NOTE | LS1-11                   | 05/10 | 15/20 | 35/40 | 45 |
| 14. | Explicit reference (direct access) to PS can load "T" bit. Console can also load "T" bit.  | X   |     |    |      |                          | X     | X     |       |    |
|     | Only implicit references (RTI, RTT, traps and interrupts) can load "T" bit. Console cannot load "T" bit.   | X   |     |    |      | X                        |       |       | X     | X  |
| 15. | Odd address/non-existent references using the SP cause a HALT. This is a case of double bus error with the second error occurring in the trap servicing the first error. Odd address trap not in LSI-11. |   |     | X  |      | X                        | X     | X     |       |    |
|     | Odd address/non-existent references using the stack pointer cause a fatal trap. On bus error in trap service, new stack created at 0/2.  |   |     | X  |      |                          |       |       | X     | X  |
| 16. | The first instruction in an interrupt routine will not be executed if another interrupt occurs at a higher priority level than was assumed by the first interrupt.                                       | X   | X   |    |      | X                        |       |       | X     | X  |
|     | The first instruction in an interrupt service is guaranteed to be executed.  | X   | X   |    |      |                          | X     | X     |       |    |
| 17. | 8 general-purpose registers.   | X   |     |    |      | X                        | X     | X     | X     |    |

|     |  |   |   |   |   |   |   |     |
|-----|--|---|---|---|---|---|---|-----|
|     | 16 general-purpose registers.  | X |   |   |   |   |   | X   |
| 18. | PSW address, 177776, not implemented must use new instructions, MTPS (Move To PS) and MFPS (Move From PS). | X | X |   | X |   |   |     |
|     | PSW address implemented, MTPS and MFPS not implemented.  | X | X |   |   | X | X | X X |
| 19. | Only one interrupt level (BR4) exists.   | X | X |   | X |   |   |     |
|     | Four interrupt levels exist.   | X | X |   |   | X | X | X X |
| 20. | Stack overflow not implemented.  | X |   |   | X |   |   |     |
|     | Some sort of stack overflow implemented.   | X |   |   |   | X | X | X X |
| 21. | Odd address trap not implemented.  |   |   | X | X |   |   |     |
|     | Odd address trap implemented.  |   |   | X |   | X | X | X X |
| 22. | FMUL and FDIV instructions implicitly use R6 (one push and Pop); hence R6 must be set up correctly.        | X | X |   | X |   |   |     |
|     | FMUL and FDIV instructions do not implicitly use R6.   | X | X |   |   |   |   | X   |
| 23. | Due to their execution time, EIS instructions can abort because of a device interrupt.                     | X |   | X | X |   |   |     |
|     | EIS instructions do not abort because of a device interrupt.   | X |   | X |   |   |   | X X |

|     | Activity   | Applicable to Programming Language Levels |     |    |      | Applicable to the PDP-11 |       |       |       |    |
|-----|--|---|-----|----|------|--------------------------|-------|-------|-------|----|
|     |  | ALL                                       | HLL | HO | NOTE | LS1-11                   | 05/10 | 15/20 | 35/40 | 45 |
| 24. | Due to their execution time, FIS instructions can abort because of a device interrupt.                           | X   |     | X  |      | X                        |       |       | X     |    |
| 25. | EIS instructions do a DATIP (or DATIO) and DATO bus sequence when fetching source operand.                       |   |     | X  |      | X                        |       |       |       |    |
|     | EIS instructions do a DATI bus sequence when fetching source operand.  |   |     | X  |      |                          |       |       | X     | X  |
| 26. | MOV instruction does just a DATO bus sequence for the last memory cycle.   |   |     | X  |      | X                        |       |       | X     | X  |
|     | MOV instruction does a DATIP and DATO bus sequence for the last memory cycle.                                    |   |     | X  |      |                          | X     | X     |       |    |
| 27. | If PC contains non-existent memory address and a bus error occurs, PC will have been incremented.                | X   |     |    |      | X                        | X     | X     |       | X  |
|     | If PC contains non-existent memory address and a bus error occurs, PC will be unchanged.                         | X   |     |    |      |                          |       |       | X     |    |
| 28. | If register contains non-existent memory address in mode 2 and a bus error occurs, register will be incremented. | X   |     |    |      | X                        |       | X     | X     | X  |
| 29. | If register contains an odd value in mode 2 and a bus error occurs, register will be incremented.                | X   |     |    |      | X                        | X     |       | X     | X  |

|     |   |   |  |   |   |   |   |   |
|-----|---|---|--|---|---|---|---|---|
|     | If register contains an odd value in mode 2 and a bus error occurs, register will be unchanged.   | X |  |   |   |   | X |   |
| 30. | Condition codes restored to original values after FIS interrupt abort (EIS doesn't abort on 35/40).   | X |  |   |   |   |   | X |
|     | Condition codes that are restored after EIS/FIS interrupt abort are indeterminate.  |   |  |   |   | X |   |   |
| 31. | Op codes 075040 through 075377 unconditionally trap to 10 as reserved op codes.   |   |  | X | X | X | X | X |
|     | If KEV11 option is present, op codes 75040 through 075377 perform a memory read using the register specified by the low order 3 bits as a pointer. If the register contents are a non-existent address, a trap to 4 occurs. If the register contents are an existent address, a trap to 10 occurs if user microcode is not present. If no KEV11 option is present, a trap to 10 occurs. | X |  | X |   |   |   |   |
| 32. | Op codes 210 through 217 trap to 10 as reserved op codes.   | X |  |   |   |   | X | X |
|     | Op codes 210 through 217 are used as a maintenance instruction.   | X |  | X |   |   |   |   |
| 33. | Op codes 75040 through 75777 trap to 10 as reserved op codes.   | X |  |   |   |   | X | X |

| Activity   | Applicable to Programming Language Levels |     |    |      | Applicable to the PDP-11 |       |       |       |    |
|--|---|-----|----|------|--------------------------|-------|-------|-------|----|
|  | ALL                                       | HLL | HO | NOTE | LS1-11                   | 05/10 | 15/20 | 35/40 | 45 |
| Only if KEV11 option is present, op codes 75040 through 75377 can be used as escapes to user microcode. Op codes 75400 through 75777 can also be used as escapes to user microcode and KEV11 option need not be present. If no user microcode exists, a trap to 10 occurs. | X   |     |    |      | X                        |       |       |       |    |
| 34. Op codes 170000 through 177777 trap to 10 as reserved instructions.  | X   |     |    |      |                          | X     | X     | X     |    |
| Op codes 170000 through 177777 are implemented as floating point instructions.   | X   |     |    |      |                          |       |       | X     | X  |
| Op codes 170000 through 177777 can be used as escapes to user microcode. If no user microcode exists, a trap to 10 occurs.   | X   |     |    |      | X                        |       |       |       |    |

**HARDWARE DIFFERENCES—TRAPS  
(TRANSPARENT TO SOFTWARE)**

| Activity | LSI-11  | PDP-11/05, 10   | PDP-11/15, 20  | PDP-11/35, 40   | PDP-11/45  |
|----------|---|---|--|---|--|
| 35.      | Priority of processor traps:<br>Bus error trap<br>Memory refresh<br>TRAP instructions<br>TRACE Trap<br>Power Fail Trap<br>Halt Line<br>Event Line Interrupt Device (BUS)<br>Interrupt Request | Priority of internal processor traps,<br>external interrupts,<br>HALT and WAIT:<br><br>Bus Error Trap<br>Trap instructions<br>TRACE Trap<br>OVFL Trap<br>PWR Fail Trap<br>UNIBUS BUS REQUEST<br>CONSOLE STOP<br>WAIT LOOP | Priority of internal processor traps,<br>external interrupts,<br>HALT and WAIT:<br><br>Bus Error Trap<br>Trap Instructions<br>TRACE Trap<br>OVFL Trap<br>PWR Fail Trap<br>CONSOLE BUS REQUEST<br>UNIBUS BUS REQUEST<br>WAIT LOOP | Priority of internal processor traps,<br>external interrupts,<br>HALT and WAIT:<br><br>Memory Parity Errors<br>Memory Management Fault<br>OVFL Trap (red zone)<br>TRAP Instructions<br>TRACE Trap<br>OVFL Trap (yellow zone)<br>PWR Fail Trap<br>CONSOLE BUS REQUEST<br>UNIBUS BUS REQUEST<br>WAIT LOOP | Priority of internal processor traps,<br>external interrupts,<br>HALT and WAIT:<br><br>Memory Parity Errors<br>BUS Error Traps<br>TRAP Instruction<br>CONSOLE Bus Request<br>Memory Management<br>OVFL Trap<br>FLOATING POINT Trap<br>PROGRAM INTERRUPT Request<br>UNIBUS BUS Request<br>WAIT LOOP<br>TRACE Trap |



**LSI-11, PDP-11/03 ENGINEERING BULLETIN****LSI-11, PDP-11/03 Differences**

The purpose of this engineering bulletin is to document the specific differences existing between the several versions of the KD11-F and KD11-J processor modules, which serve as the CPU in LSI-11 and PDP-11/03 microcomputer systems. Also included are programming recommendations pertinent to the several revisions.

At present, LSI-11 and PDP-11/03 systems are equipped with processor modules having two levels of revision: one level is revisions C and D, and the other level is revision E and subsequent revisions. In the descriptions of differences and in programming recommendations, these two levels are referred to as Rev C/D and Rev E. The revision designation for a given processor module is indicated by the letter following the date code on the etch side of the processor module quick-release handle.

Each difference is defined in detail in this bulletin, along with necessary recommendations for resolving any departures from conventional PDP-11 programming techniques imposed by the difference.

**Revision Level Differences****Length of BUS INIT**

The period of the signal BUS INIT, whether asserted by a RESET instruction or upon power up, will be approximately 100  $\mu$ s on revision level C/D systems rather than the specified 12  $\mu$ s, as on the revision level E systems. For those users having revision level C/D systems, it is recommended that only the leading edge or BUS INIT be used and not the signal pulse width, when designing user interfaces.

**RTT Instruction and Internal Refresh**

If the T bit is set to 1 by an RTT instruction on systems at revision level C/D, and the CPU performs a memory refresh immediately after executing the RTT, the one instruction delay for the T bit will not occur. As a result, the T bit trap will occur immediately after refresh is completed. The difference is present only when the CPU performs a refresh cycle so that the incidence should be very low since there is normally very little use of the T bit by user software. This difference can be eliminated on the present machines by having refresh done externally. As an alternative, this difference can be handled in a program using the T-bit, by making sure that the PC has changed from the previous trace trap. If the PC did not change, the program knows that a refresh cycle intervened.

On revision level E systems, this one instruction cycle delay occurs normally after execution of an RTT.

**Possibility of Memory Locations Being Altered When a BUS Error Occurs**  
Due to a race condition on revision level C/D systems, the contents of

locations 4, 6, 14, or 16 could possibly be altered during the processing of a bus error by the CPU, due to conflict on the WD tri-state DAL bus lines.

It is recommended that intentional bus errors, such as memory sizing, be kept to a minimum. This alteration, if it occurs, takes place before the content of the trap location is read by the CPU. Hence, if location 4 is altered, the resultant PC from that location would not be correct.

### **PSW Not Initialized for all Power-Up Options**

If a power-up option other than to 24 (Option #0) is selected with the CPU module jumpers, on revision level C/D systems, the event and device interrupts are inhibited until a specific MTPS instruction is executed. However, since the PSW bit 7 may be off, this condition is not apparent. Also, the remaining bits in the PSW are indeterminate at this time. On Rev C/D systems, for power-up modes other than to location 24, it is recommended that the PSW be initialized through the Stack Pointer (register R6). An MTPS instruction will not clear the T-bit, which could be set to one in this circumstance.

In the following example, a PC and the desired PSW are pushed onto the Stack, and an RTI is executed to load the PSW. The PC pushed onto the Stack is the address of the instruction following the RTI, so that after execution of the RTI, normal program execution can continue.

```
MOV #PSW -(SR)   Push PSW onto Stack
MOV #TAG1-(SR)   Push new PC onto Stack
RTI              Pop new PC and PSW of Stack
```

TAG: Program continues here

However, any trap or interrupt following power-up options 1, 2, or 3 should be avoided, because if the T bit is on, it will be saved on the stack and upon return, the T bit will be honored, since the microlevel flag is now set.

For revision level E systems, the PSW is initialized to 200 so that interrupts are inhibited for the power-up to 173000 (option 2). The remaining power-up options, other than power-up to 24, specifically options 1 and 3, still have the above difference. If microcode or ODT power-up are selected (options 3 and 1, respectively), the PSW will not be initialized. For these two cases, the effect of this difference should be minimal since most users will type a "G" (GO) which will clear the PSW. However, those who select the ODT power-up option, then enter a program, set the PC, and type "P", could experience difficulty. To avoid this difficulty, the PSW should be explicitly cleared using the "RS" command.

### **Certain Reserved Op Codes in Group 7 through 77 Not Reserved**

In the op code group 7 through 77, those codes that have bit 3 = 1 will be executed as the maintenance instruction (21R), rather than trapping to 10, revision level C/D systems. Those codes having bit 3 = 0 will try to execute microcode at micro PC 3000. But as long as a user does not add microcode that responds to micro PC 3000, these op codes will trap to 10.

### **Rubout Function Does Not Work for Register Numbers**

This description clarifies a characteristic of revision level C/D systems; specifically, if a user types the following (where the underlined characters are those Micro-ODT characters previously typed):

@R5\6/012345 The response by the CPU is a backslash which is the symbol for RUBOUT.

This action will not open register 6(R6) but rather memory location 6. Once the "R" is typed, attempting to rubout only the register number will change modes and open the corresponding memory location instead. Other commands such as "line feed" will indicate that the user is in memory mode by typing memory locations and not register numbers. It is recommended that when the wrong register number is selected, another "R" be typed along with the new register number. For example, @R5R6/123456.

### **SRUN Differences**

The signal SRUN is available at module finger CH1 on Rev E processor modules. This signal is asserted by Rev E processor modules under the following circumstances:

1. Once each time the CPU fetches an instruction.
2. Once each time the CPU executes a Reset instruction.
3. Twice each time the CPU responds to assertion of the bus signal BDCOK H.
4. Once each time the CPU times out on a bus cycle,(including timeouts occurring when in the terminal mode, and when sizing memory under the console "L" command).

In the PDP-11/03 systems, this signal is connected to a retriggerable one-shot to drive the front panel RUN indicator light.

On Rev C/D processor modules, SRUN does not appear on module finger CH1. However, on Rev C/D processor modules, SRUN can be decoded from the SROM fingers DD1, DE1, DF1, and DH1 (SROM signals SROM0 H, SROM1 H, SROM2 H and SROM3 H). These signals are stable and valid only while the signal SPH3 H on finger DD1 is true, so that SRUN can be decoded once each time the CPU fetches an instruction. Decoding of the SROM signals is based on SROM1 being true and SROM0, SROM2, and SROM3 being false, with SPH3 H true serving as the decoding strobe. On Rev C/D processor modules, this particular assertion of the SROM signals occurs once during the fetching of each instruction and at no other time.

### **Condition Codes Not Restored to Original Values After EIS/FIS Interrupt Abort**

This difference applies only to diagnostics that may check this condition. Otherwise, it is transparent to a programmer since all the EIS and FIS instructions affect all the condition codes.

### **Programming Recommendations**

**Micro-ODT Typeout on Power-Up is Dependent on the Character in the UAR/T Buffer (DLV11)**

For both revision level C/D and E systems and jumpered for power-up to location 24 option, the normal micro-ODT type out is the contents of the PC followed by a carriage return, line feed, and prompt character. If that character is a RUBOUT (177), the typeout will be the contents of the PC followed by a backslash as a result of a RUBOUT character in the UAR/T buffer. The content of the UAR/T buffer can be a RUBOUT character as a consequence of DLV11 power up. The carriage return, line feed, and prompt character will not be typed in this case. However, the state of micro-ODT does not change and after the backslash it is ready to accept commands. Consider this example: A power-up to 24 (24 contains 0 and 0 contains 0) normally would type:

000002

@

If a RUBOUT character is contained in the UAR/T buffer, the typeout would be

00002/

Although no prompt character is typed, ODT will accept all commands in the normal manner. Also, if the machine halts for any reason and a RUBOUT character is in the UAR/T input buffer, the same erroneous typeout will occur.

#### **Caution on Storing MTPS and EIS Instructions in ROM**

If MTPS and EIS instructions are executed out of ROM on both revision level C/D and E systems, some addressing modes will cause a bus timeout error. This occurs because the processor performs a read-modify-write rather than a read when obtaining the source operand. Therefore, any mode where the effective source operand address is stored in ROM will cause a timeout error.

The following example causes a timeout:

```
MUL #123,R0
```

The timeout can be avoided, however, by first moving the literal to a general register or to RAM, as follows:

```
MOV #123,R4  
MUL R4,R0
```

or another alternative is:

```
MOV #123,TEMP  
MUL TEMP,R0
```

where TEMP is in RAM.

#### **Event Line not Disabled when "G" and "L" Commands are Used**

If a free-running clock, such as 60 Hz from the power supply, is attached to the BEVNT bus line on both revision level C/D and E systems, an interrupt to location 100 will occur when using the "G" and "L" commands prior to executing the first instruction. Therefore, with an MTPS

as the first instruction, a program can not disable the BEVNT bus line by inhibiting interrupts.

User programs requiring a free-running clock attached to the BEVNT bus line can temporarily avoid this situation by setting the PSW (RS) to 200, loading the PC with the starting address instead of using the "G" command, and then using the "P" command. Before using the "L" command, the PSW (RS) can be set to 200, thereby inhibiting interrupts, to avoid receiving the event interrupt after loading the ABS loader.

#### **T Bit is On and a WAIT Instruction is Executed**

If the T bit is = 1 when a WAIT instruction is executed on both revision level C/D and E systems, the processor will hang and will not honor any device, event, halt, or power-down interrupts. However, refresh will still occur. In general, this situation will occur when an application program is linked to ODT-11, and the user is single stepping his program and a WAIT instruction is encountered. To escape this state, the processor must be powered-down and then powered-up again, with an assertion of the DCOK signal. Note that the content of dynamic MOS RAM could be lost if DCOK is asserted for longer than 400  $\mu$ S. The use of a one-shot multivibrator to assert DCOK is recommended in these circumstances. This difference will not affect user software since the T bit is not generally used by application programs.

#### **T Bit and EIS/FIS Instructions Being Aborted by Device Interrupts**

If the T bit is set while executing an EIS or FIS instruction, a subsequent event interrupt causes that instruction to be aborted, and the new PSW at the T bit vector location plus two (location 16) inhibits interrupts. Consequently, the processor will enter an endless loop executing the EIS or FIS instruction and the T bit trap and will never service the device or event interrupt.

The reason for this circumstance is that the RTT instruction, which is at the end of the T bit service routine, does not arbitrate interrupts if the T bit has been set, thereby popping the old PSW from the stack.

This restriction is imposed in order to guarantee that the T-bit trap is not immediately executed in this situation as with the RTI instruction. Most system and application software do not use the T bit and only a utility such as ODT-11 would use this facility. In those cases where the T bit is used, the solution is to allow interrupts to be honored in the T bit service routine by lowering the PS priority. Irrespective of T bit state, the limiting case concerns an interrupt such as a free-running clock tied to the BEVNT bus line which is occurring at such a rapid rate that the EIS or FIS instruction never finishes. In such a case, every time the interrupt is serviced and a return is made to re-try the EIS or FIS instruction, the interrupt occurs again and the instruction is aborted.

#### **Caution When Clearing Device Interrupt Enable Bits**

On both revision level C/D and E systems, clearing device Interrupt Enable bits while the device is still active can lead to a bus timeout error when the processor attempts to receive the interrupt vector from that device. Consider the example:

```
PSW = 0
CLR @ #177564
```

As a result, the DLV11 Serial Line Unit interrupt enable bit is being cleared. Now, assume that the transmitter is still active and sending characters, and further assume that the Done bit in the status register becomes set shortly after the CLR instruction is fetched, but before the Interrupt Enable bit can be cleared. The device will now post an interrupt request because Done bit has been set and Interrupt Enable bit is still set. The CLR instruction will complete execution and the processor will recognize the interrupt request since there was not enough time for the device to disable the interrupt request. The processor will then attempt to obtain a vector from the interrupting device. However, a bus timeout error will occur because the device now has had enough time to remove the interrupt request and will not respond. The processor treats this timeout as a fatal condition and halts by entering Micro-ODT. If multiple interrupt requests were pending at this time, a timeout would not occur since the next device would respond with an interrupt vector.

One method of avoiding this problem is to disable interrupts immediately before the Interrupt Enable bit is cleared. For example:

```
MTPS #200
CLR @ #177564
MTPS #0
```

In this situation, enough time has been allowed for the interrupt request to be removed by the device. This feature was included to permit detection of faulty interrupt operation; specifically when an interrupting device does not properly respond within the required time period.

PERIPHERALS

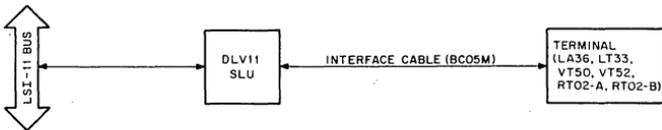
**E.1 CONSOLE DEVICE APPLICATIONS**

Peripherals available for use in PDP-11/03 and LSI-11 I/O applications as the console terminal are listed below. All peripherals listed are serial line devices which interface via the DLV11 Serial Line Unit interface module. Refer to Table E-1 for peripheral types, models, brief specifications, and required interface options (DLV11 and either the BC05M 20 mA current loop interface cable or the BC05C EIA interface cable). Contact your local Digital Equipment Corporation Sales Office for detailed information on any of the peripherals listed.

Typical applications are shown in Figures E-1 through E-3. Although the RT02-A is not capable of full console operation, it can be used as the console device, but it is limited to the following ODT commands:

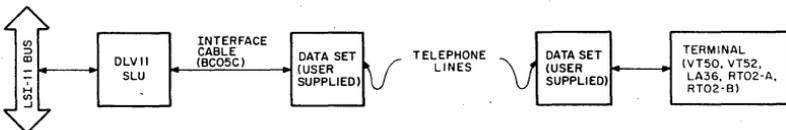
| ODT Command | RT02-A Keys     |
|-------------|-----------------|
| CR          | SEND            |
| LF          | SHIFT and CLEAR |
| /           | SHIFT and ÷     |
| @           | SHIFT and @     |
| G           | SHIFT and GO    |
| RO          | SHIFT and ERROR |

The console device can either be directly interfaced to the DLV11 or it can be operated in a remote location and interfaced via data sets or acoustic couplers and telephone lines. However, only the LA36, LT33, VT50, and VT52 are capable of remotely placing the LSI-11 system in the Halt state by asserting a line break (continuous "space" transmission). (This feature is jumper-enabled on the DLV11 through the use of framing error detection.)



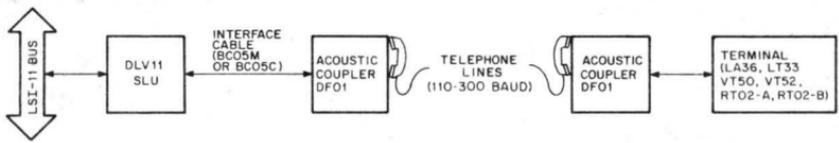
CP-1770

Figure E-1 Direct 20 mA Current Loop Interface



CP-1771

Figure E-2 Telephone Line Interface Via Data Sets



CP-1772

Figure E-3 Telephone Line Interface Via Acoustic Couplers

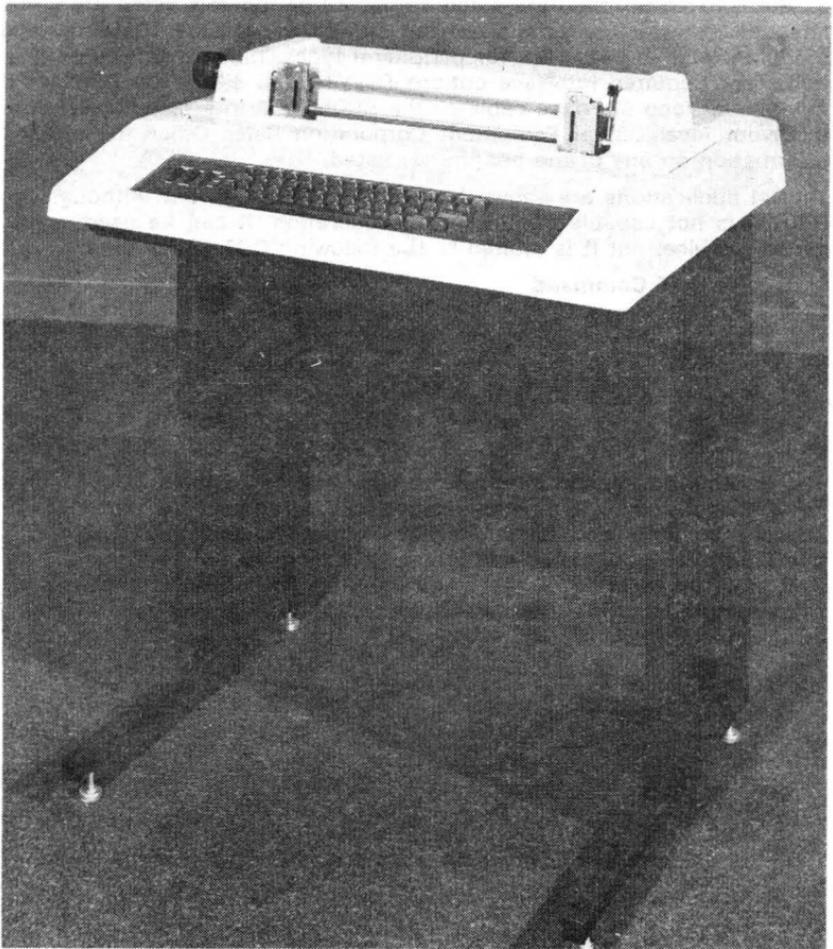


Figure E-4 LA36 DECwriter II

**Table E-1 LSI-11 Console Terminal Peripheral Options**

| <b>Model</b> | <b>Name</b>  | <b>Figure</b> | <b>Terminal Type</b>                               | <b>Display Capacity</b>                    | <b>I/O Speed (baud rate)</b>            | <b>BREAK Key</b> | <b>Serial Interface Type</b>     | <b>Required Interface Options</b>  |
|--------------|--|---------------|--|--|---|------------------|----------------------------------|------------------------------------|
| LA36         | DECwriter II   | E-4           | Keyboard/Printer                                   | 132 characters/line                        | 300                                     | Yes              | 20 mA loop<br>optional EIA       | DLV11,<br>BC05M<br>DLV11,<br>BC05C |
| LT33         | Teletypewriter   |               | Keyboard/Printer<br>and Paper Tape<br>Reader/Punch | 72 characters/line                         | 110                                     | Yes              | 20 mA loop                       | DLV11,<br>BC05M                    |
| VT50         | DECscope   | E-5           | Keyboard/CRT<br>Display                            | 960 characters<br>(80 char<br>× 12 lines)  | 75—9600                                 | Yes              | 20 mA<br>loop or<br>optional EIA | DLV11,<br>BC05M<br>DLV11,<br>BC05C |
| VT52         | DECscope   | E-6           | Keyboard/CRT<br>Display                            | 1920 characters<br>(80 char<br>× 24 lines) | 75—9600                                 | Yes              | 20 mA<br>loop or<br>optional EIA | DLV11,<br>BC05M<br>DLV11,<br>BC05C |
| RT02-A       | 30 Character<br>Keyboard Re-<br>mote Terminal<br>(Limited con-<br>sole ODT<br>command set) | E-7           | Alphanumeric<br>Data Entry<br>Terminal             | 32 characters                              | 110—300<br>(20 mA)<br>110—1200<br>(EIA) | No               | 20 mA loop<br>or EIA             | DLV11,<br>BC05M<br>DLV11,<br>BC05C |
| RT02-B       | Alphanumeric<br>Terminal   | E-8           | Full Alpha-<br>numeric Data<br>Entry Terminal      | 32 characters                              | 110—300<br>(20 mA)<br>110—1200<br>(EIA) | No               | 20 mA loop<br>or EIA             | DLV11,<br>BC05M<br>DLV11,<br>BC05C |



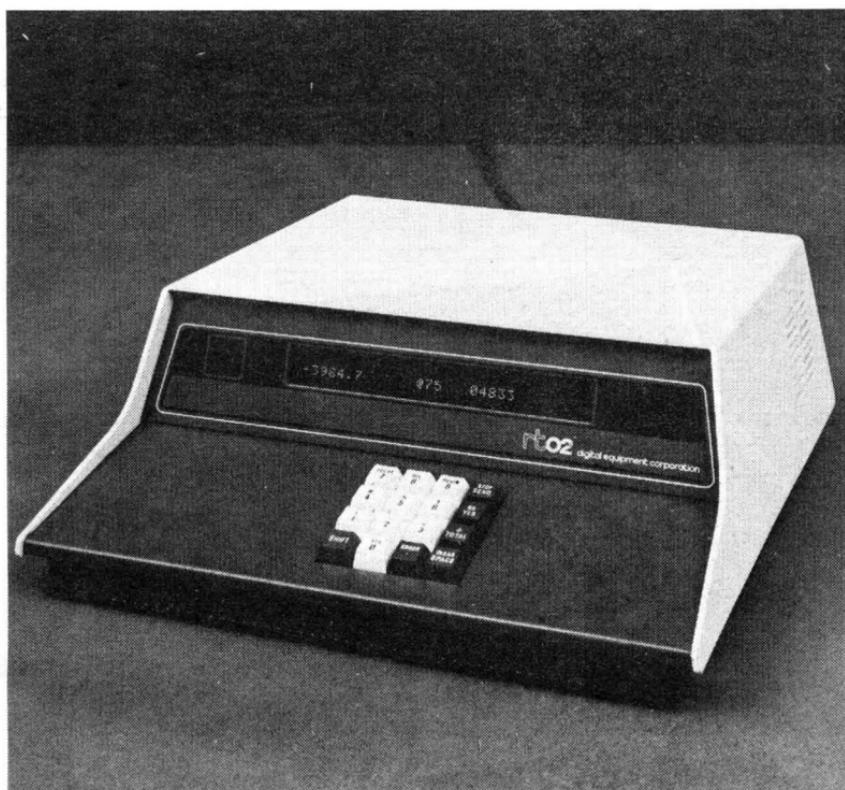


Figure E-7 RT02-A 30-Character Keyboard Remote Terminal



Figure E-8 RT02-B Alphanumeric Terminal

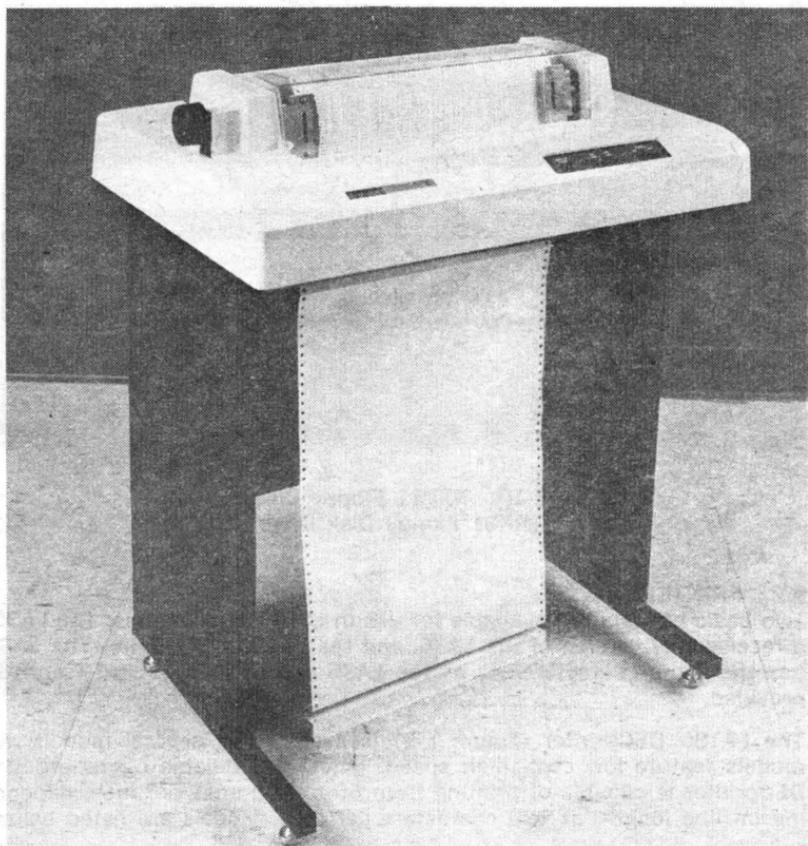


Figure E-9 LA180 DECprinter

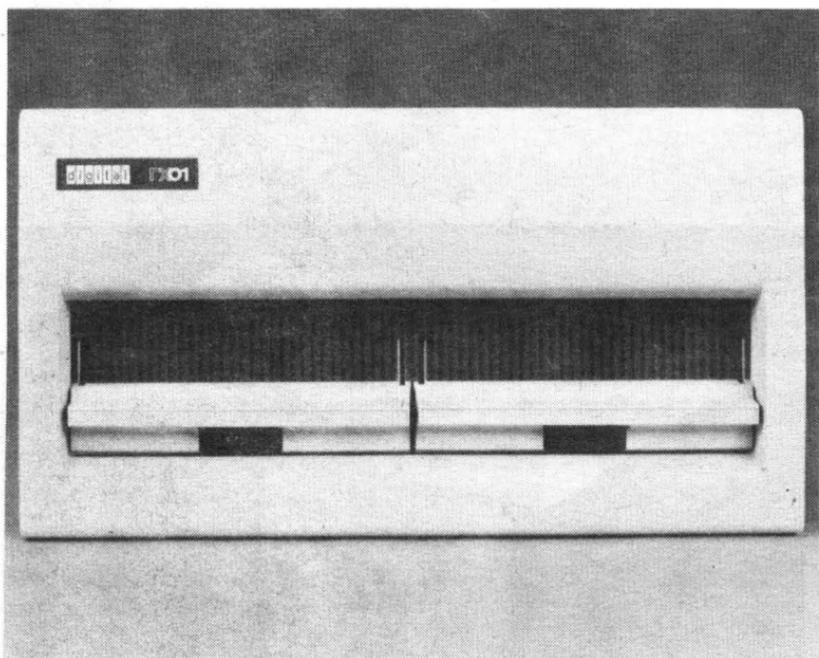


Figure E-10 RXV11 Floppy Disk System  
(RX01 Floppy Disk Drive)

## E.2 PRINTERS

Two basic printers are available for use in LSI-11 applications: The LA35, a receive-only version of the LA36, and the LA180 DECprinter. The LA35 has the same characteristics as the LA36 except no keyboard (send) is provided.

The LA180 DECprinter (Figure E-9) is available in several models. All models feature low cost, high speed, quiet, and reliable operation. The DECprinter is capable of printing from 60 to 400 lines per inch, depending on line length, at 180 characters per inch. Models are listed below:

| Model    | Description  |
|----------|--|
| LAV11-PA | 115 V, 60 Hz LA180, LSI-11 bus interface controller module, BC11S interface cable.   |
| LAV11-PD | 230 V, 50 Hz LA180, LSI-11 bus interface controller module, BC11S interface cable.   |
| LA180    | Serial LA180 DECprinter. Requires one (optional) DLV11 serial line interface and one interface cable. The model numbers are defined below: |
| LA180-CA | 115 V, 60 Hz serial LA180, 20 mA current loop interface (use BC05M interface cable).   |

| <b>Model</b> | <b>Description</b>   |
|--------------|--|
| LA180-CD     | 230 V, 50 Hz serial LA180, 20 mA current loop interface (use BC05M interface cable). |
| LA180-EA     | 115 V, 60 Hz serial LA180, EIA interface (use BC05C interface cable).                |
| LA180-ED     | 230 V, 50 Hz serial LA180, EIA interface (use BC05C interface cable).                |

### **E.3 RXV11 FLOPPY DISK SYSTEM**

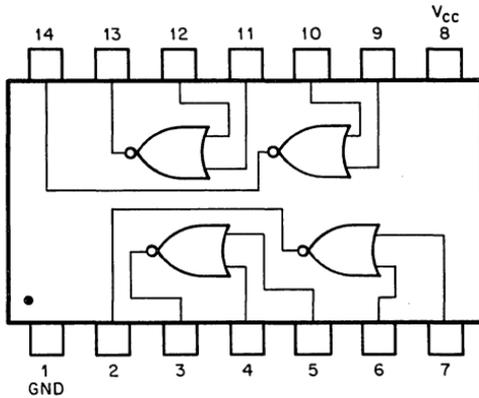
The RXV11 Floppy Disk System is a low cost, random access, mass memory device that stores data in fixed-length blocks on preformatted flexible diskettes. The RXV11 is available in single drive and dual drive models (Figure E-10). Each drive is capable of storing up to 256K 8-bit bytes of data. Each floppy disk system includes an RX01 floppy disk drive (single or dual disk drive assembly), a 15-foot interface cable, and an RXV11 interface that plugs into the LSI-11 or PDP-11/03 system backplane. RXV11 models are described below:

| <b>Model</b> | <b>Description</b>                            |
|--------------|---|
| RXV11-AA     | 115 V, 60 Hz single drive, floppy disk system |
| RXV11-AD     | 230 V, 50 Hz single drive, floppy disk system |
| RXV11-BA     | 115 V, 60 Hz dual drive, floppy disk system   |
| RXV11-BD     | 230 V, 50 Hz dual drive, floppy disk system   |



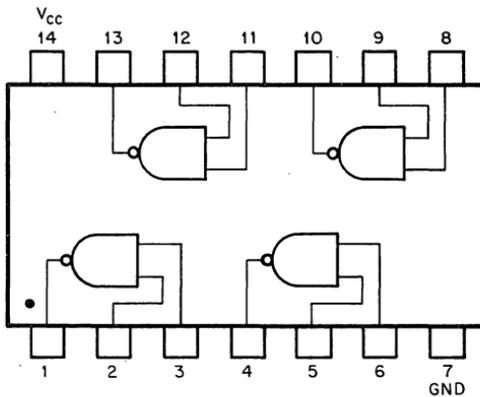
## APPENDIX F

### INTEGRATED CIRCUIT DIAGRAMS



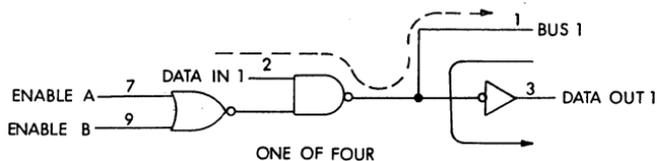
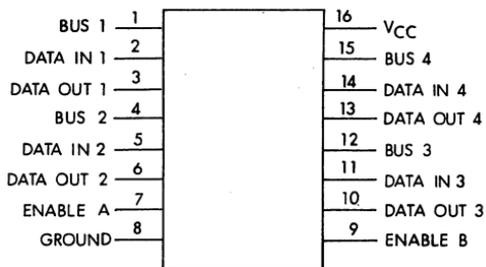
CP-1271

F.1 DEC 8640 QUAD 2-INPUT NOR GATES  
(Bus Receiver)



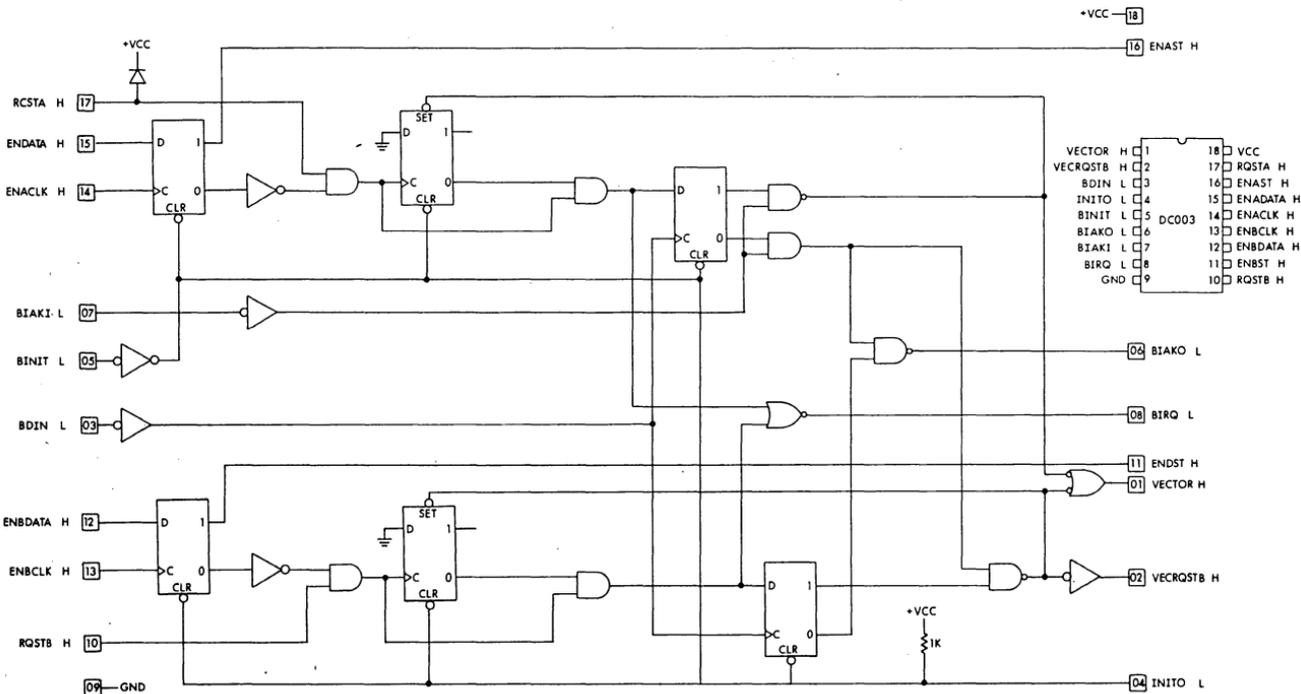
CP-1272

F.2 DEC 8881 QUAD 2-INPUT NAND GATE  
(Bus Driver)



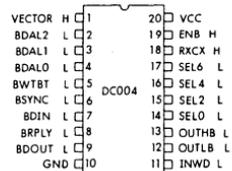
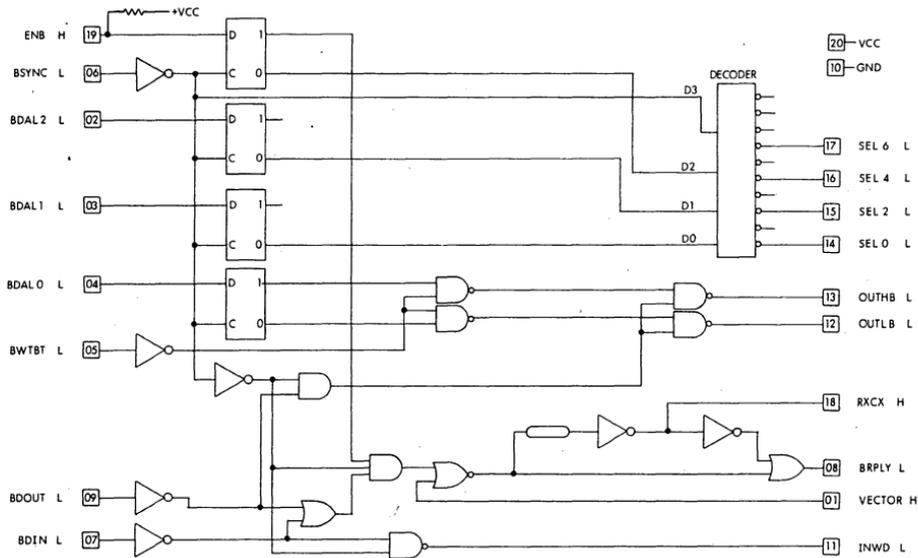
F.3 DEC 8641 QUAD UNIFIED BUS TRANSCEIVER  
(Bus Receiver/Driver)

F.4 DC003 INTERRUPT CHIP

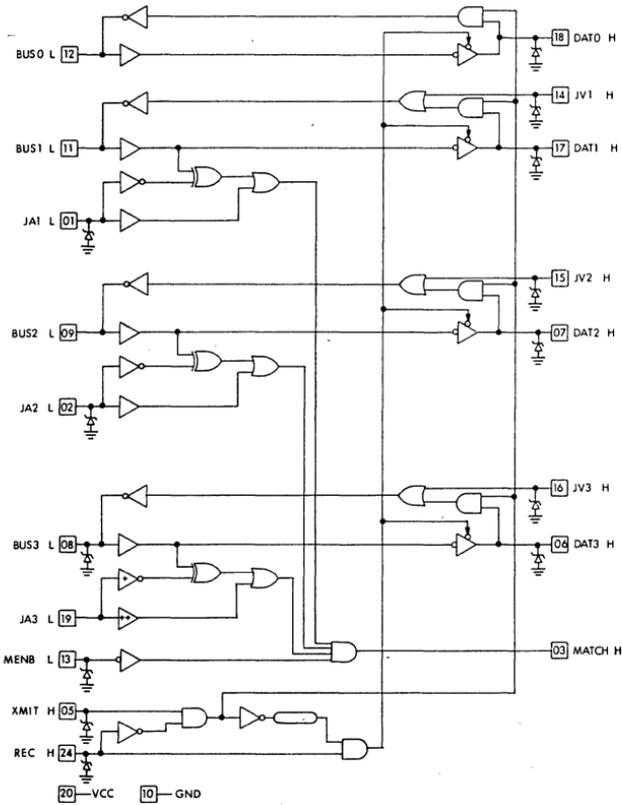


F-4

F.5 DC004 PROTOCOL CHIP



|         |    |    |        |
|---------|----|----|--------|
| JA1 L   | 1  | 20 | VCC    |
| JA2 L   | 2  | 19 | JA3 L  |
| MATCH H | 3  | 18 | DAT0 H |
| REC H   | 4  | 17 | DAT1 H |
| XMIT H  | 5  | 16 | JV3 H  |
| DAT3 H  | 6  | 15 | JV2 H  |
| DAT2 H  | 7  | 14 | JV1 H  |
| BUS3 L  | 8  | 13 | MENB L |
| BUS2 L  | 9  | 12 | BUS0 L |
| GND     | 10 | 11 | BUS1 L |



F.6 DC005 TRANSCEIVER



## APPENDIX G

### ABSOLUTE LOADER FORMAT

The Absolute Loader uses the eight general registers (R0-R7) and does not preserve or restore their previous contents. Therefore, caution should be taken to restore or load these registers, when necessary, after using the loader.

A block of data punched on paper tape in absolute binary format has the following format.

|         |     |                                      |
|---------|-----|--------------------------------------|
| FRAME 1 | 001 | start frame                          |
| 2       | 000 | null frame                           |
| 3       | xxx | byte count (low 8 bits)              |
| 4       | xxx | byte count (high 8 bits)             |
| 5       | yyy | load address (low 8 bits)            |
| 6       | yyy | load address (high 8 bits)           |
|         | •   | data is                              |
|         | •   | placed                               |
|         | •   | here                                 |
| zzz     |     | last frame contains a block checksum |

A program on paper tape can consist of one or more blocks of data. Each block with a byte count (frames 3 and 4) greater than six causes subsequent data to be loaded into core (starting at the address specified in frames 5 and 6 for a normal load). The byte count is a positive integer denoting the total number of bytes in the block, excluding the checksum. When the byte count of a block is six, the specified load address is checked to see whether the address is to an even or to an odd location. If the address is even, the absolute Loader transfers control to the address specified. Thus, the loaded program is automatically started upon completion of loading. If the address is odd, the absolute loader halts.

The transfer address (TRA) may be explicitly specified in the source program by placing the desired address in the operand field following the .END statement. For example,

```
.END ALPHA
```

specifies the symbolic location ALPHA as the TRA, and

```
.END
```

causes the absolute Loader to halt. With

```
.END nnnnnn
```

the absolute Loader also halts if the address (nnnnnn) is odd.

The checksum is located in the low byte of general register R0. Upon completion of a load, the low byte of R0 should be all zeros. Otherwise,

a checksum error has occurred, indicating that the load was not correct. The checksum is the low-order byte of the negation of the sum of all the previous bytes in the block. When all bytes of a block including the checksum are added together, the low-order byte of the result should be zero. If not, some data was lost during the load or erroneous data was picked up; the load was incorrect. When a checksum error is indicated the entire program should be reloaded.

# digital

**DIGITAL EQUIPMENT CORPORATION, Corporate Headquarters: Maynard, Massachusetts 01754, Telephone: (617) 897-5111**

## **SALES AND SERVICE OFFICES**

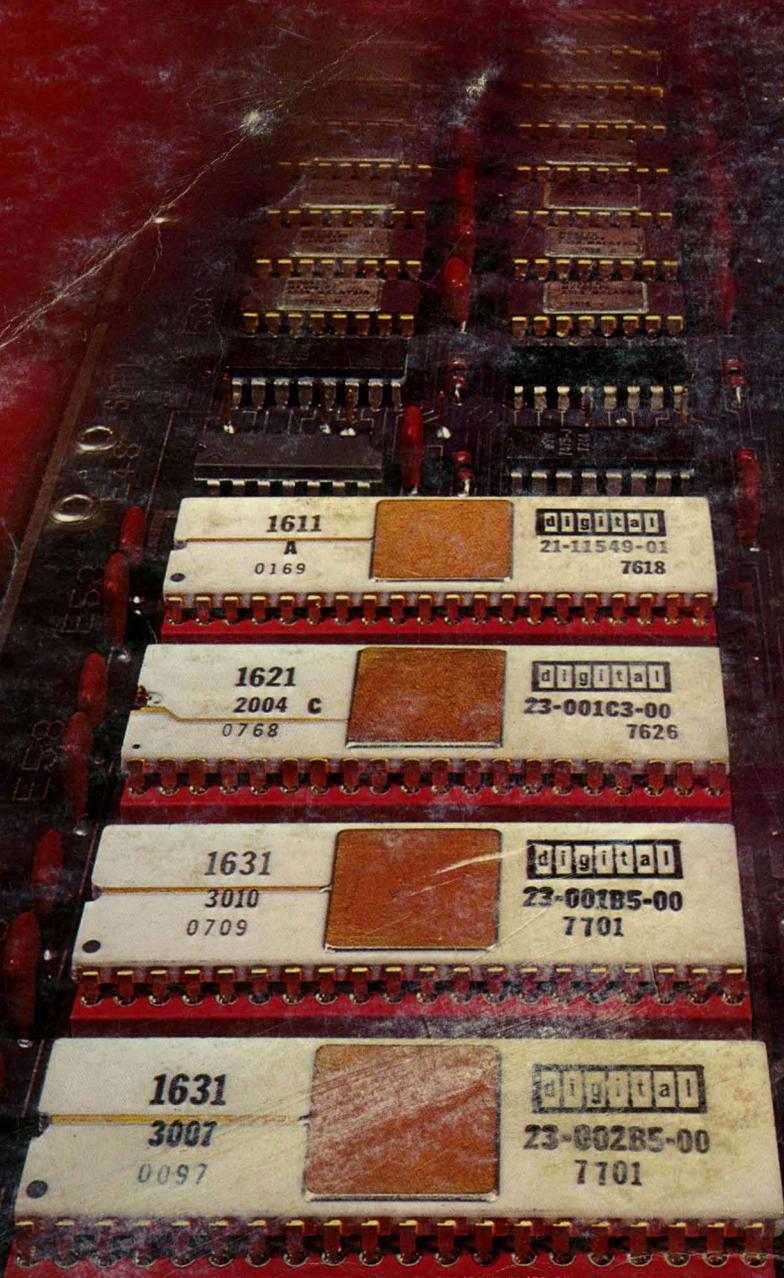
**UNITED STATES—ALABAMA, Huntsville • ARIZONA, Phoenix and Tucson • CALIFORNIA, El Segundo, Los Angeles, Oakland, Ridgecrest, San Diego, San Francisco (Mountain View), Santa Ana, Santa Clara, Stanford, Sunnyvale and Woodland Hills • COLORADO, Englewood • CONNECTICUT, Fairfield and Meriden • DISTRICT OF COLUMBIA, Washington (Lanham, MD) • FLORIDA, Ft. Lauderdale and Orlando • GEORGIA, Atlanta • HAWAII, Honolulu • ILLINOIS, Chicago (Rolling Meadows) • INDIANA, Indianapolis • IOWA, Bettendorf • KENTUCKY, Louisville • LOUISIANA, New Orleans (Metairie) • MARYLAND, Odenton • MASSACHUSETTS, Marlborough, Waltham and Westfield • MICHIGAN, Detroit (Farmington Hills) • MINNESOTA, Minneapolis • MISSOURI, Kansas City (Independence) and St. Louis • NEW HAMPSHIRE, Manchester • NEW JERSEY, Cherry Hill, Fairfield, Metuchen and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Albany, Buffalo (Cheektowaga), Long Island (Huntington Station), Manhattan, Rochester and Syracuse • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland (Euclid), Columbus and Dayton • OKLAHOMA, Tulsa • OREGON, Eugene and Portland • PENNSYLVANIA, Allentown, Philadelphia (Bluebell) and Pittsburgh • SOUTH CAROLINA, Columbia • TENNESSEE, Knoxville and Nashville • TEXAS, Austin, Dallas and Houston • UTAH, Salt Lake City • VIRGINIA, Richmond • WASHINGTON, Bellevue • WISCONSIN, Milwaukee (Brookfield) •**

**INTERNATIONAL—ARGENTINA, Buenos Aires • AUSTRALIA, Adelaide, Brisbane, Canberra, Melbourne, Perth and Sydney • AUSTRIA, Vienna • BELGIUM, Brussels • BOLIVIA, La Paz • BRAZIL, Rio de Janeiro and Sao Paulo • CANADA, Calgary, Edmonton, Halifax, London, Montreal, Ottawa, Toronto, Vancouver and Winnipeg • CHILE, Santiago • DENMARK, Copenhagen • FINLAND, Helsinki • FRANCE, Grenoble and Paris • GERMAN FEDERAL REPUBLIC, Cologne, Frankfurt, Hamburg, Hannover, Munich, Stuttgart and West Berlin • HONG KONG • INDIA, Bombay • INDONESIA, Jakarta • IRELAND, Dublin • ITALY, Milan and Turin • JAPAN, Osaka and Tokyo • MALAYSIA, Kuala Lumpur • MEXICO, Mexico City • NETHERLANDS, Utrecht • NEW ZEALAND, Auckland • NORWAY, Oslo • PUERTO RICO, Santurce • SINGAPORE • SWEDEN, Gothenburg and Stockholm • SWITZERLAND, Geneva and Zurich • UNITED KINGDOM, Birmingham, Bristol, Edinburgh, Leeds, London, Manchester and Reading • VENEZUELA, Caracas •**

digital

# microcomputer handbook

1976-77



1611

A

0169

digital

21-11549-01

7618

1621

2004 C

0768

digital

23-001C3-00

7626

1631

3010

0709

digital

23-001B5-00

7701

1631

3007

0097

digital

23-002B5-00

7701