



**iRMX 86 OPERATING SYSTEM
PART II (I/O)**

WORKSHOP NOTEBOOK

VERSION 5.0 DECEMBER 1982

**iRMX™
86
OPERATING
SYSTEM**

IRMX 86 OPERATING SYSTEM PART I I (I/O) WORKSHOP NOTEBOOK
By LUIS ZIEGENHIRT

With contributions by Stan Mazor and layout and artwork by Mary Lou Faraco.

© 1982 INTEL CORPORATION

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BXP, CREDIT, i, ICE, i² ICE, iCS, i_m, iMMX, Insite, INTEL, int_el, Intelelevision, Inteltec, int_el_igent Identifier™, int_el_igent Programming™, Intellink, iOSP, iPDS, iRMS, iSBC, iSBX, iSXM, Library Manager, MCS, Megachassis, Micromainframe, MULTIBUS, Multichannel™ Plug-A-Bubble, MULTIMODULE, PROMPT, Promware, RMX/80, RUPI, System 2000, and UPI, and the combination of ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

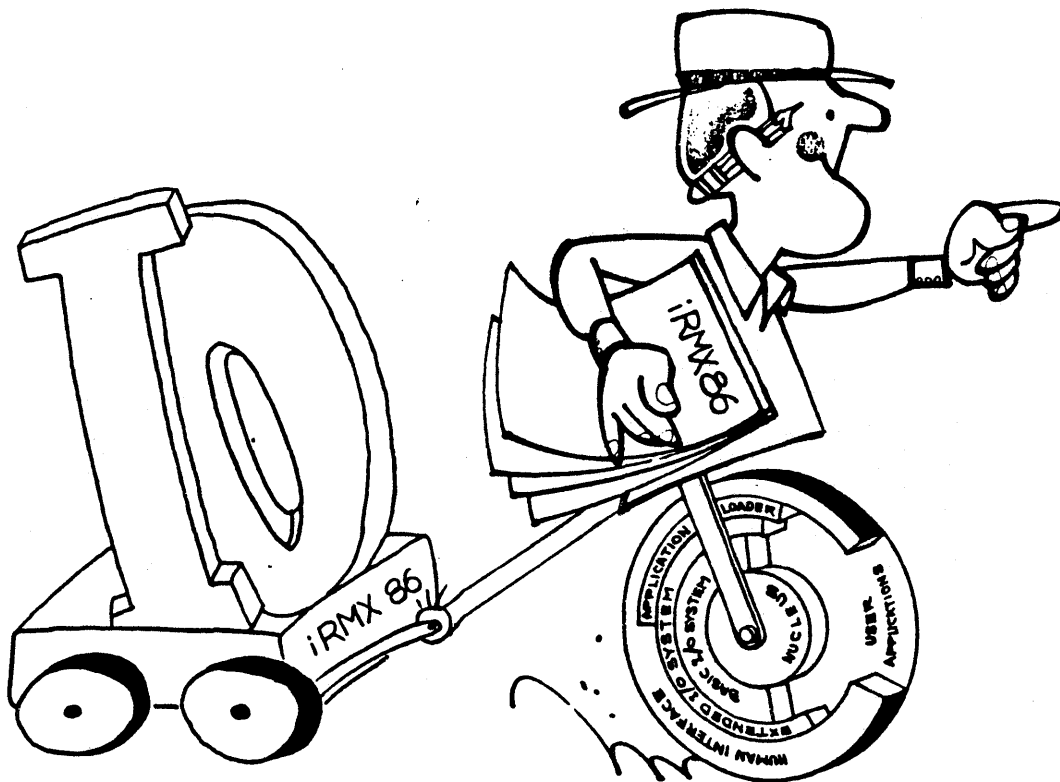
MDS is an ordering code only and is not used as a product name or trademark. MDS* is a registered trademark of Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

iRMX 86 OPERATING SYSTEM PART I I (I/O)

WORKSHOP NOTEBOOK

VERSION 5.0



DECEMBER 1982

TABLE OF CONTENTS

CHAPTER

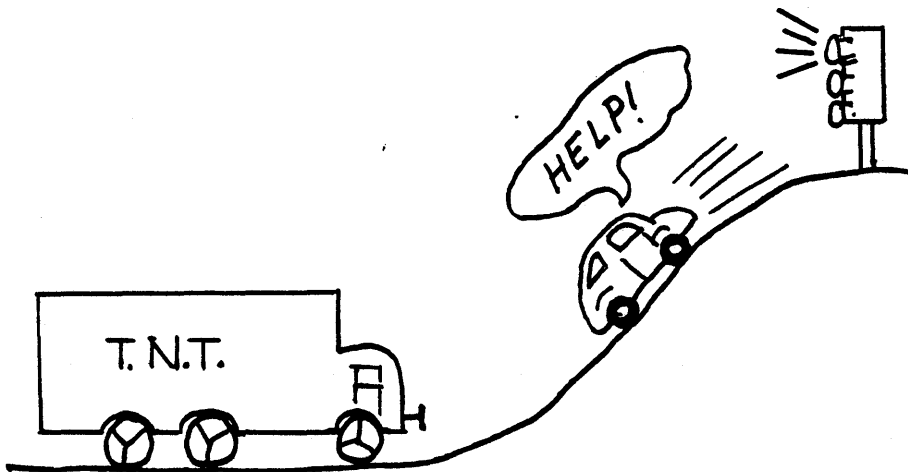
1.	IRMX 86 BASIC I/O SYSTEM (APPLICATION PROGRAMMER'S VIEW)	
	FILES1-1
	FILE CLASSES1-6
	I/O OPERATIONS1-7
	HIERARCHICAL FILE STRUCTURE.1-10
	FILE CONNECTIONS1-18
2.	IRMX 86 BASIC I/O SYSTEM (SYSTEM PROGRAMMER'S VIEW)	
	DEVICE CONNECTIONS2-1
	USER OBJECTS2-10
	FILE ACCESS LISTS.2-12
	SUMMARY.2-15
	LAB ONE: USING THE BIOS.2-16
3.	EXTENDED I/O SYSTEM (APPLICATION PROGRAMMER'S VIEW)	
	WHY USE THE EIOS?.3-1
	ACCESSING NAMES FILES.3-4
	EIOS BUFFERING3-9
	FILE CREATION.3-14
4.	IRMX 86 EXTENDED I/O SYSTEM (SYSTEM PROGRAMMER'S VIEW)	
	LOGICAL NAMES4-1
	DEVICE CONNECTIONS.4-3
	I/O JOBS.4-5
	DEFAULT TOKENS.4-10
	LAB TWO: USING THE EIOS4-13
5.	WRITING DEVICE DRIVERS	
	INDEPENDENT I/O5-1
	DRIVER COMPONENTS5-3
	THE DUIB.5-5
	THE I/O REQUEST5-8
	DRIVER FUNCTIONS.5-11
6.	CUSTOM DEVICE DRIVERS	
	THE INTERRUPT TASK AND HANDLER.6-3
	THE QUEUE\$IO PROCEDURE.6-7
	THE INIT\$IO PROCEDURE6-9
7.	RANDOM AND COMMON DEVICE DRIVERS	
	COMPONENTS.7-1
	THE INTERRUPT PROCEDURE7-5
	THE START PROCEDURE7-8
	DEVICE INFORMATION TABLE.7-18
	LAB THREE: WRITING A COMMON DEVICE DRIVER7-21

8.	BASIC I/O CONFIGURATION	
	TABLES8-1
	ICU868-10
	LAB FOUR: BASIC I/O SYSTEM GENERATION8-11
9.	EXTENDED I/O CONFIGURATION	
	TABLES9-1
	ICU869-8
	LAB FIVE: BASIC I/O SYSTEM GENERATION9-9
10.	THE iRMX 86 APPLICATION LOADER	
	LOADER FUNCTIONS10-1
	TYPES OF LOADABLE CODE10-3
	SYSTEMS WITHOUT THE EIOS10-7
	LOADER RESULT SEGMENT10-9
	SYSTEMS WITH THE EIOS10-15
11.	APPLICATION LOADER CONFIGURATION	
	TABLES11-1
	LAB SIX: APPLICATION LOADER11-7
12.	iMMX 800	
	BASIC CONCEPTS12-2
	CHANNELS12-4
	iMMX SYSTEM CALLS12-12
	THE iMMX JOB12-17
	LAB SEVEN: INTERDEVICE COMMUNICATION12-29
13.	THE HUMAN INTERFACE	
	COMMANDS	13-1
	SYSTEM CALLS	13-3
	THE RESIDENT USER	13-5
	DEFINITION FILES	13-6
	LAB EIGHT: HUMAN INTERFACE CONFIGURATION	13-10
14.	UNIVERSAL DEVELOPMENT INTERFACE	
	SPECIFICATIONS	14-1
	LIBRARIES	14-3
	DEVELOPMENT PROCESS	14-4
	SYSTEM CALLS	14-7

APPENDICES

- A ALTER TEXT EDITOR
- B PL/M 86
- C BOOTSTRAP LOADER

MANUAL vs. AUTOMATIC TRANSMISSIONS



WHY BUY MANUAL?

- ECONOMICAL

(MINIMUM I CAN BUY THE CAR WITH)
(I CAN SAVE ON GAS, IF I DON'T ABUSE IT)
(I DON'T NEED A BATTERY TO START THE CAR)

- PERFORMANCE

(I AM A RACE CAR DRIVER)
(TERRAIN CALLS FOR IT, MOUNTAINOUS, HILLS....ETC.)



- DISADVANTAGE

(NEED A SKILLED DRIVER)
(HAVE TO DO MORE WORK)
(CAN'T HOLD ON TO "PERSON" FRIEND WHILE DRIVING)

WHY BUY AUTOMATIC?

- LAZY

(LESS WORK, NO PUSHING CLUTCH, NO CHANGING GEARS)

- SKILL

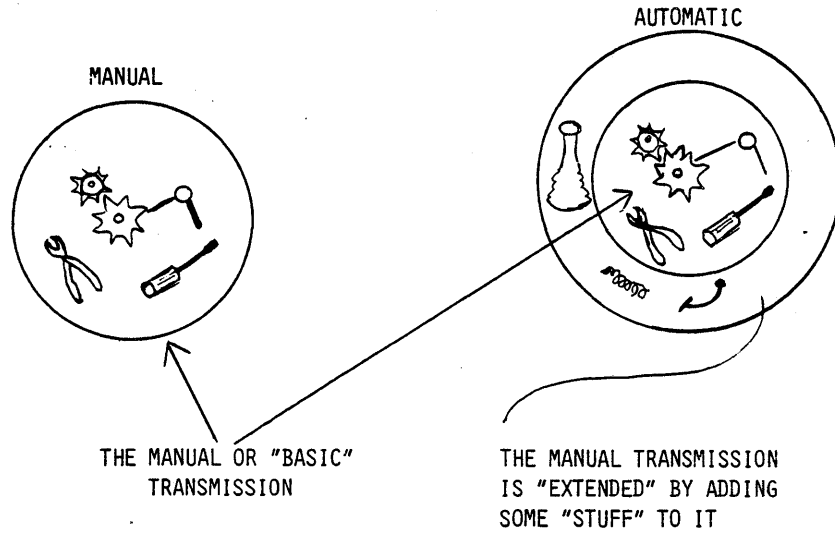
(DON'T NEED TO WORRY ABOUT ROLLING DOWN HILLS)



- DISADVANTAGE

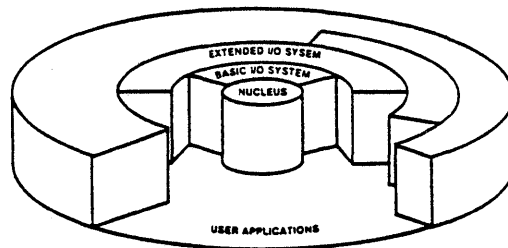
(COSTS MORE)

A CLOSER LOOK



SPEAKING OF BASIC AND EXTENDED

IRMX 86 LAYERS



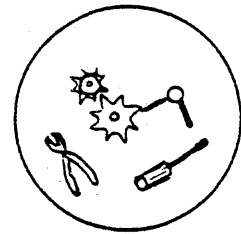
CHAPTER 1

RMX 86 BASIC I/O SYSTEMS

-An Applications Programmer's View

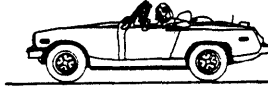
- FILES
- FILE CLASSES
- I/O OPERATIONS
- HIERARCHICAL FILE STRUCTURES
- FILE CONNECTION

MANUAL

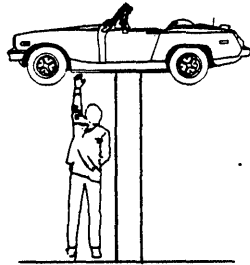


PROGRAMMING ROLES

THERE ARE TWO PROGRAMMING ROLES ASSOCIATED WITH THE iRMX 86 OPERATING SYSTEM.



- THE APPLICATION PROGRAMMER USES SYSTEM CALLS AND OBJECTS THAT AFFECT ONLY HIS OWN JOB

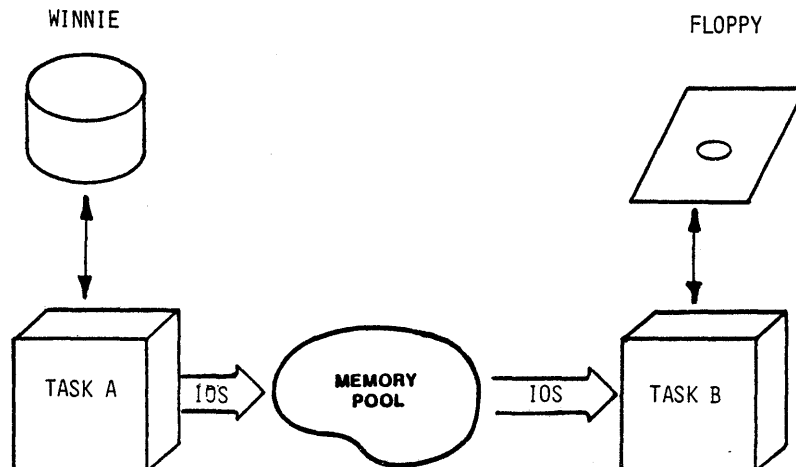


- THE SYSTEM PROGRAMMER CONTROLS SYSTEM RESOURCES AND CHARACTERISTICS

1-1

I/O COMMUNICATION

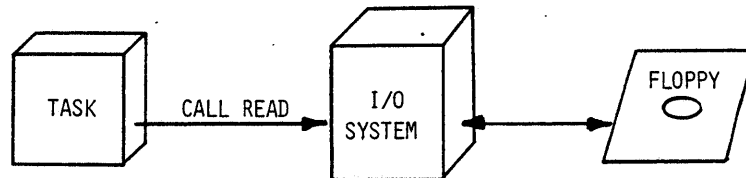
- THROUGH iRMX 86 I/O SYSTEMS, TASKS COMMUNICATE WITH EACH OTHER AND THE EXTERNAL WORLD.



1-2

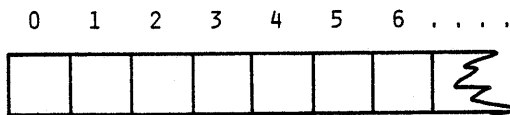
COMMUNICATION

- TASKS MAKE "SYSTEM CALLS" TO THE BIOS TO COMMUNICATE WITH THE FILE



1-3

RMX I/O IS DONE TO/FROM FILES

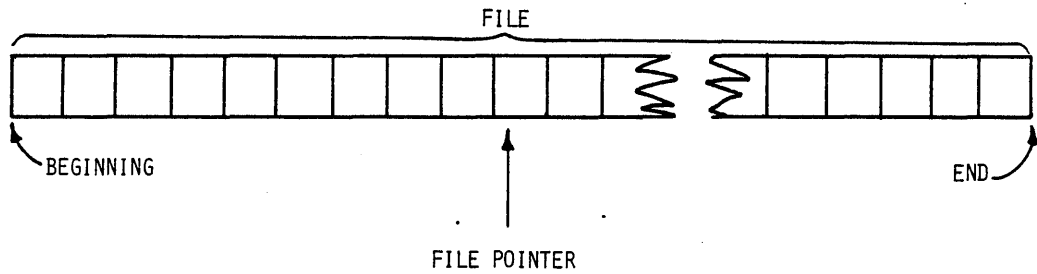


- A FILE IS AN UNBOUNDED SEQUENCE OF COMPONENTS
- FILES ARE USED FOR
 - LONG TERM STORAGE
 - TEMPORARY DATA STORAGE EXPANSION

1-4

THE FILE

- FILES HAVE A FILE POINTER



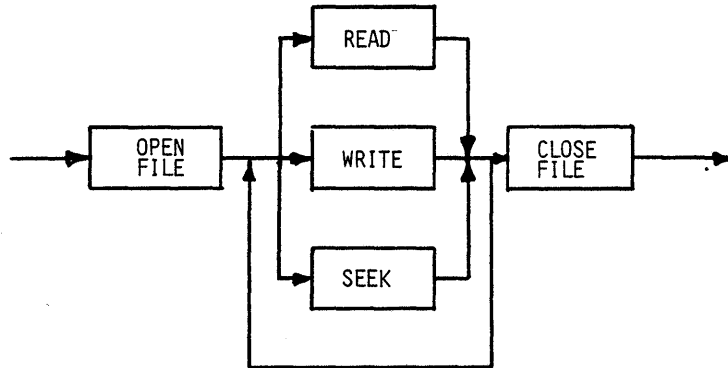
1-5

CLASSES OF FILES

- PHYSICAL FILE - CONTINUOUS SEQUENCE OF BYTES ON A DEVICE WHERE NO FILE STRUCTURE IS IMPOSED (E.G., PRINTER, TERMINAL)
- STREAM FILE - MEMORY BASED BYTE STREAMS. DESTRUCTIVE-READ SERVES COMMUNICATION BETWEEN TASKS
- NAMED FILE - DATA FILES RESIDING ON RANDOM ACCESS STORAGE DEVICES. ACCESSED VIA ASCII NAMES (E.G., FLOPPY DISK, HARD DISK).

1-6

ACCESSING NAMED FILES



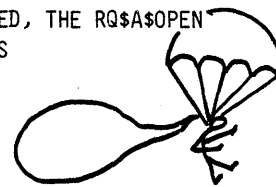
1-7

OPEN

AFTER A FILE CONNECTION HAS BEEN ESTABLISHED, THE `RQAOPEN` SYSTEM CALL OPENS A FILE FOR I/O OPERATIONS

- OPEN FILE COMMANDS

- READ
- WRITE
- SEEK

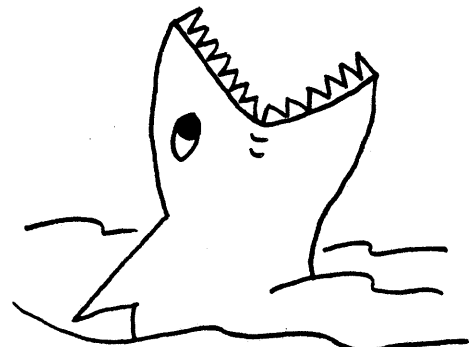


- THE `RQAOPEN` SPECIFIES

- A FILE MAY BE
 - READ ONLY
 - WRITE ONLY
 - READ OR WRITE

- TYPE OF SHARING DESIRED

- READERS
- WRITERS
- ALL



1-8

OPENING A FILE

```
CALL RQ$A$OPEN (FILE$CONNECTION$TOKEN, MODE, SHARE, RSP$MBOX, @STATUS);
```

MODE: MODE OF ACCESS DESIRED

<u>VALUE</u>	<u>MODE</u>
1	OPEN FOR READING
2	OPEN FOR WRITING
3	OPEN FOR READING AND WRITING

SHARE: KIND OF SHARING DESIRED

0	PRIVATE USE ONLY
1	SHARE WITH READERS ONLY
2	SHARE WITH WRITERS ONLY
3	SHARE WITH ALL USERS

1-9

THE RQ\$A\$READ SYSTEM CALL

- READ 'COUNT' BYTES FROM AN OPEN FILE INTO THE BUFFER
- BYTES ARE READ STARTING AT FILE POINTER

```
CALL RQ$A$READ (FILE$CONNECTION$TOKEN, @BUFFER, COUNT, RSP$MBOX, @STATUS);
```

1-10

THE RQ\$A\$WRITE SYSTEM CALL

- WRITE ANY NUMBER OF BYTES FROM A USER BUFFER INTO AN OPEN FILE
- THE DATA IS WRITTEN BEGINNING AT THE CURRENT SETTING OF THE FILE POINTER

```
CALL RQ$A$WRITE (FILE$CONNECTION$TOKEN, @BUFFER, COUNT, RESP$MBOX, @STATUS);
```

1-11

THE RQ\$A\$SEEK SYSTEM CALL

- MOVES THE FILE POINTER TO ANY BYTE POSITION IN THE OPEN FILE
- HI\$PTR\$MOVE, LOW\$PTR\$MOVE = WORD PAIR CONTAINING A 32-BIT UNSIGNED NUMBER

```
CALL RQ$SEEK (FILE$CONNECTION$TOKEN, MODE, PTR$MOVE, RESP$MBOX, @STATUS);
```

<u>MODE</u>	<u>ACTION BY POINTER</u>
1	BACKWARD BY PTR\$MOVE (RELATIVE)
2	EQUAL TO PTR\$MOVE (ABSOLUTE)
3	FORWARD BY PTR\$MOVE (RELATIVE)
4	TO EOF MINUS PTR\$MOVE (ABSOLUTE)

1-12

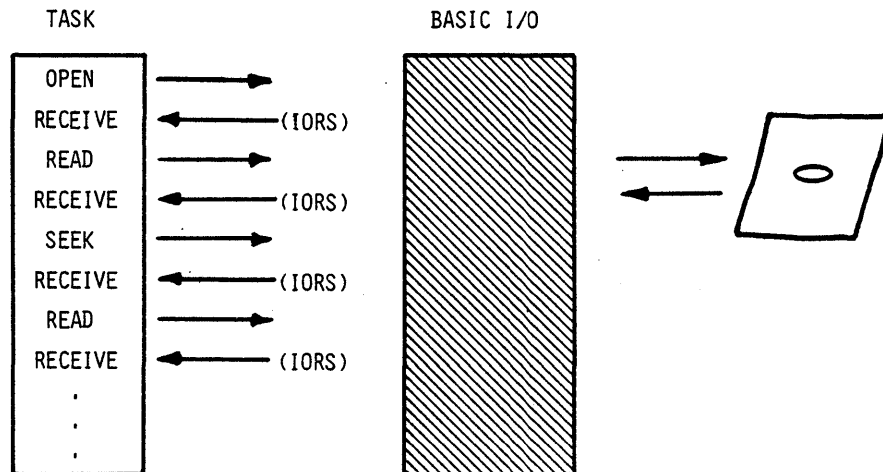
THE RQ\$A\$CLOSE SYSTEM CALL

- CLOSES AN OPEN FILE CONNECTION
- A FILE CONNECTION IS CLOSED BY THE APPLICATION PROGRAMMER
 - WHEN I/O OPERATIONS ARE COMPLETE
 - WHEN THE MODE OR SHARED STATUS IS TO BE CHANGED

```
CALL RQ$A$CLOSE (FILE$CONNECTION$TOKEN,RESP$MBOX, EXCEP$PTR);
```

BIOS I/O ASYNCHRONOUS

- EACH I/O OPERATION (OPEN, READ, WRITE, SEEK, CLOSE) SHOULD BE FOLLOWED BY A "STATUS CHECK"(HANDSHAKE)



RESULT OF I/O OPERATION CALLS

- THE PROGRAM MAY RECEIVE AN I/O RESULT SEGMENT* (IORS) AFTER A FILE ACCESS CALL.
*SEE BASIC I/O REFERENCE MANUAL FOR A DESCRIPTION OF THE IORS STRUCTURE.
- THE PROGRAM WAITS AT THE RESPONSE MAILBOX SPECIFIED IN THE CALL.
- AFTER EXAMINING THE STATUS FIELD IN THE IORS THE PROGRAMMER MUST DELETE THE SEGMENT.
- IF THE RESPONSE MAILBOX PARAMETER IN THE CALL EQUALS Ø THEN NO IORS WILL BE RETURNED BY THE I/O SYSTEM. (NOT RECOMMENDED)

1-15

EXAMPLE ACCESS CALL

```
CALL RQ$READ (FILE$CONNECTION$TOKEN, @BUFFER, 80, RSP$MBOX, @STATUS);  
IF STATUS<>E$OK THEN CALL ERROR; /*SYNCHRONOUS PART*/  
.  
.  
OVERLAPPED PROCESSING  
.  
.  
IORS$TOKEN = RQ$RECEIVE$MESSAGE (RSP$MBOX, . . . ., @STATUS);  
IF STATUS<>E$OK THEN CALL ERROR; /*SYNCHRONOUS PART*/  
  
IF IORS.STATUS<>E$OK THEN CALL ERROR: /*ASYNCHRONOUS PART*/  
CALL RQ$DELETE$SEGMENT (IORS$TOKEN, @STATUS);
```

1-16

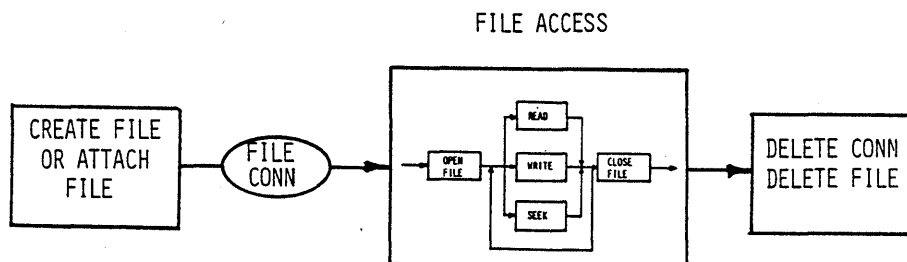
THE EASY WAY!

- FOR READ, WRITE, AND SEEK WE MAY USE THE RQ\$WAIT\$IO SYSTEM CALL
- THE FORM OF THE CALL IS

```
ACTUAL = RQ$WAIT$IO(CONN$T,RSPMBOX,TIMELIMIT,@STATUS);
```
- BASIC I/O DEALS WITH IORS'S DIRECTLY
 - EFFICIENT BECAUST IT KEEPS A SUPPLY OF IORS'S AVAILABLE
 - USER TASKS DO NOT HAVE TO DELETE THE IORS

1-17

FILE CONNECTION



1-18

FILE ATTACH

- IF THE FILE ALREADY EXISTS THEN THE USER MAKES AN "ATTACH\$FILE" SYSTEM CALL.

```
CALL RQ$A$ATTACH$FILE (USER$TOKEN, PREFIX, SUBPATH, . . . , @STATUS);
```

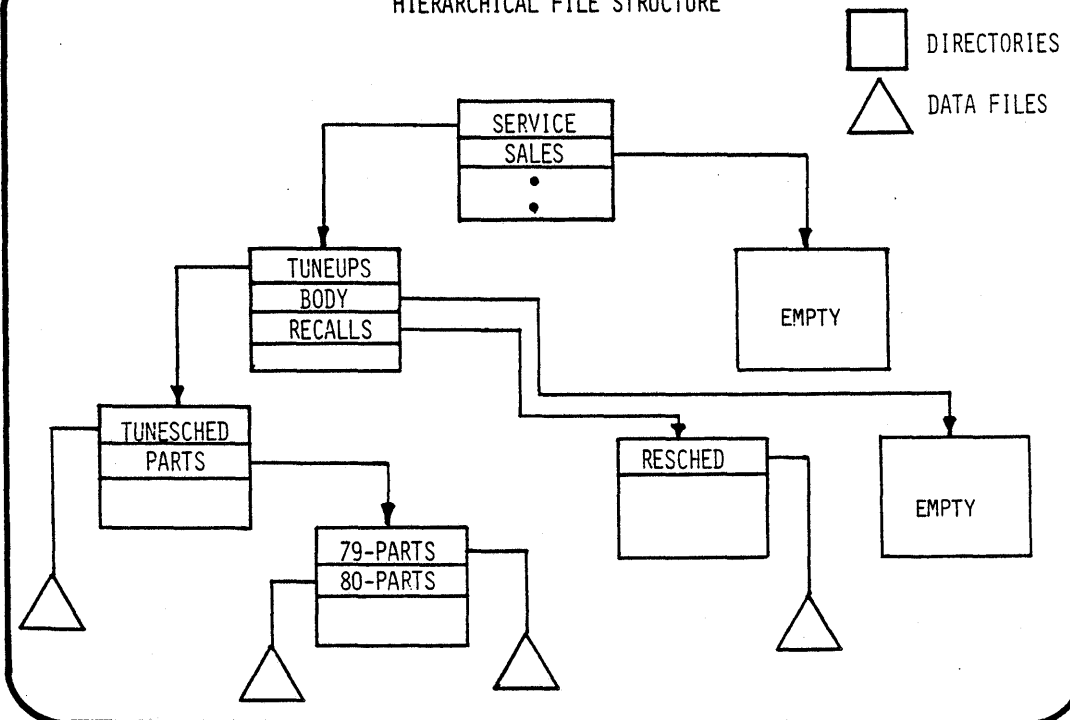
SEE BASIC I/O REFERENCE MANUAL FOR DETAILS.

```
CALL RQ$A$ATTACH$FILE (USER$OBJECT, DEVICE$CONNECTION$TOKEN,  
a(25, 'SERVICE/TUNEUP/TUNESCHED'), RSP$MBOX, @STATUS);
```

- THE USER\$TOKEN WILL BE DISCUSSED IN NEXT CHAPTER.

1-19

HIERARCHICAL FILE STRUCTURE



1-20

RESULT OF FILE CONNECTION

- THE PROGRAMMER MUST WAIT AT THE RESPONSE MAILBOX SPECIFIED IN THE CREATE\$FILE OR ATTACH\$FILE SYSTEM CALL.

```
CALL RQ$A$ATTACH$FILE ( , , , RSP$MBOX, @STATUS);
```

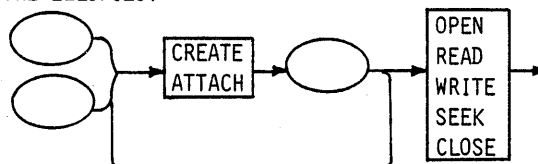
```
FILE$CONNECTION$TOKEN = RQ$RECEIVE$MESSAGE (RSP$MBOX, , , @STATUS);
```

- SUCCESSFUL CONNECTION RETURNS FILE CONNECTION TOKEN (TYPE = 101H)
- UNSUCCESSFUL CONNECTION RETURNS SEGMENT TOKEN (TYPE = 6)
 - THE SEGMENT RETURNED IS AN IORS
 - THE PROGRAMMER MUST DELETE THE IORS AFTER EXAMINING THE STATUS FIELD

1-23

EXERCISE (OPEN MANUAL)

- HOW LONG CAN AN ASCII NAME IN A SUBPATH BE?
- DESCRIBE PICTORIALLY AN IORS.
- FILL IN THE ELLIPSES:



- A PREFIX CAN BE A _____ OR A _____.

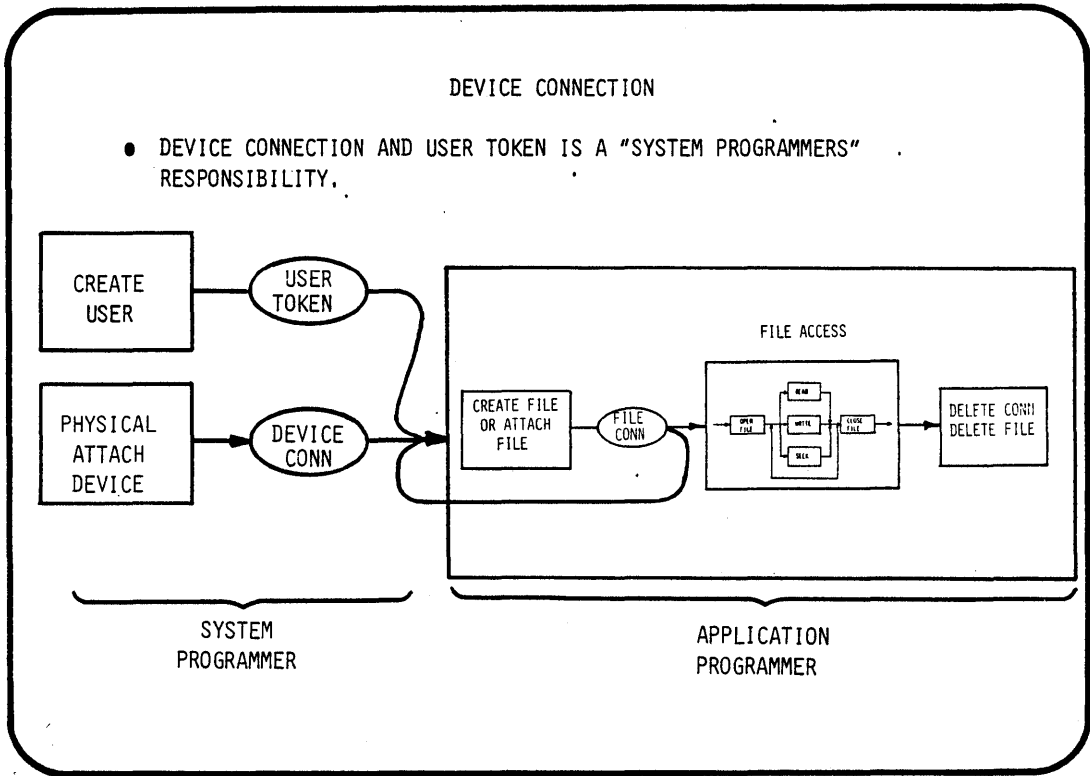
1-24

CHAPTER 2

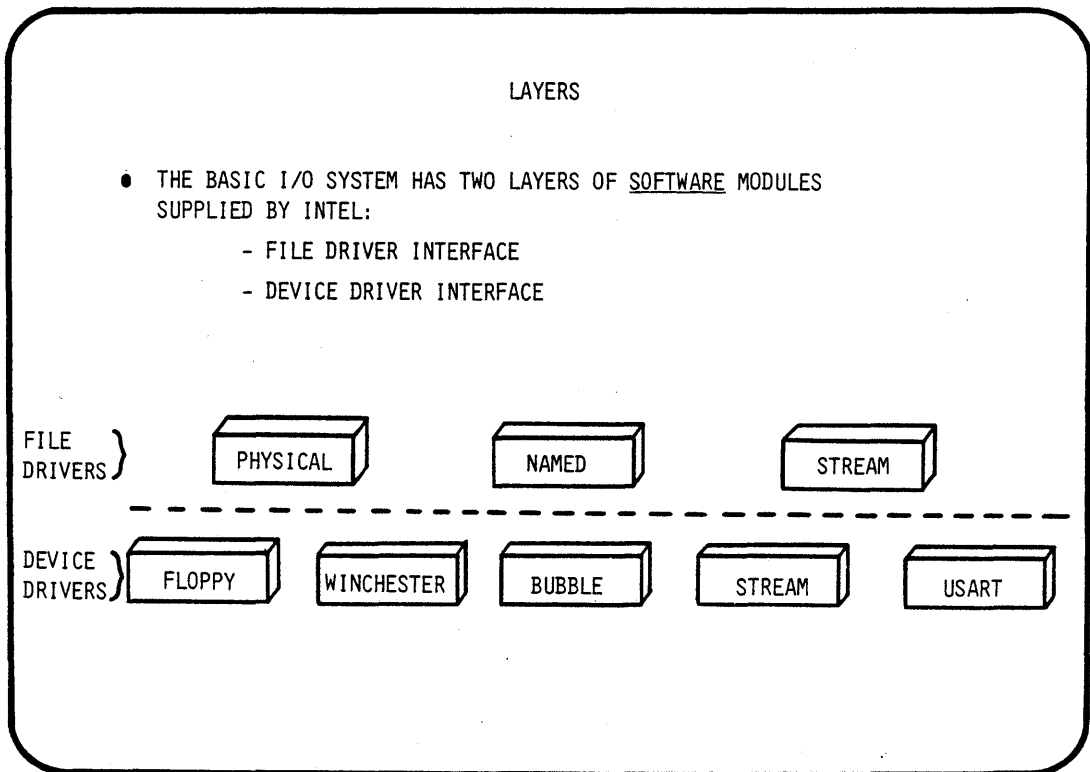
RMX 86 BASIC I/O SYSTEM

-A System Programmer's View

- DEVICE CONNECTION
- PHYSICAL ATTACHMENT
- USER OBJECT
- FILE ACCESS LIST
- SUMMARY



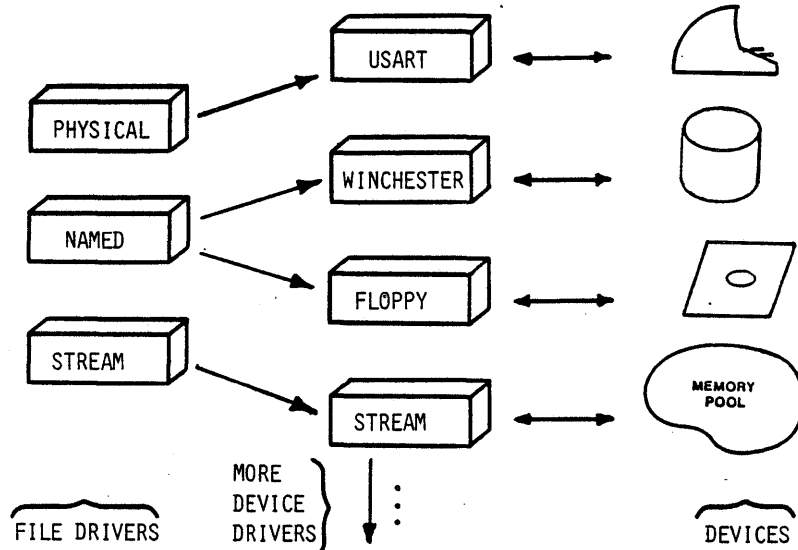
2-1



2-2

PHYSICAL ATTACHMENT

- AT RUN TIME THE FILE DRIVER IS "PHYSICALLY ATTACHED" TO THE DEVICE DRIVERS THROUGH I/O SYSTEM CALLS



INTEL SUPPLIED DEVICE DRIVERS

- THE BASIC I/O SYSTEM SUPPORTS SEVERAL DEVICES

- SOME EXAMPLES -

<u>DEVICE CONTROLLER</u>	<u>DEVICE DRIVER 'NAME'</u>
iSBC 204 SS/128 UNIT 0	'F0'
iSBC 204 SS/512 UNIT 0	'FX0'
iSBC 208 DS/256 UNIT 0	'AFDD0'
iSBC 215/218 PRIAM 3450 UNIT 0	'IW0'
iSBC 215/218 DS/256 FLOPPY UNIT 0	'WFDD0'
LINE PRINTER	'LP'
USART	'T0'
⋮	⋮
⋮	⋮

PHYSICAL ATTACHMENT

```
CALL RQ$A$PHYSICAL$ATTACH$DEVICE (DRV$NAME, FILE$DRIVER, RESP$MBOX, @STATUS);
```

- A SUCCESSFUL "PHYSICAL ATTACH DEVICE" RETURNS A DEVICE CONNECTION TOKEN (TYPE = 101H)
- AN UNSUCCESSFUL CONNECTION RETURNS A SEGMENT TOKEN (TYPE = 6)
 - THE STRUCTURE OF THE SEGMENT IS AN I/O REQUEST/RESULT SEGMENT (IORS)

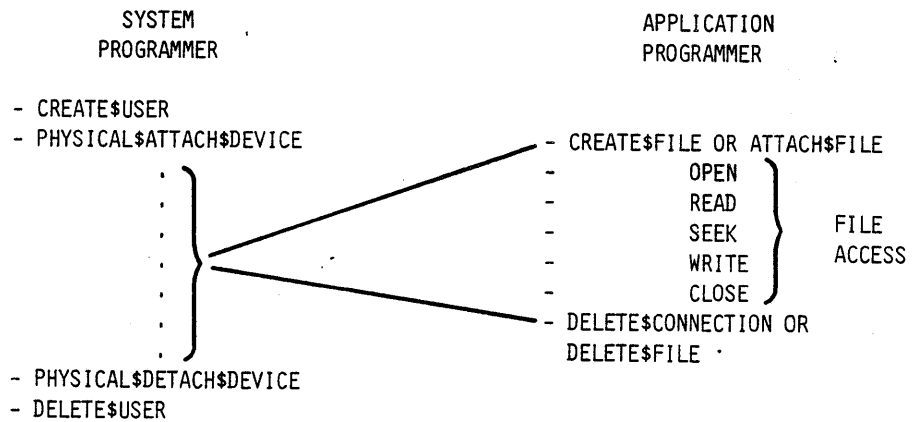
2-5

EXAMPLE

```
CALL RQ$A$PHYSICAL$ATTACH$DEVICE(@ (2, 'F0'), 4, RESP$MBOX, @STATUS);  
.  
● <OVERLAPPED PROCESSING MAY OCCUR HERE>  
.  
DEVICE$CONNECTION$TOKEN = RQ$RECEIVE$MESSAGE (RESP$MBOX, 0FFFFH, @RESP, @STATUS);  
/* TEST FOR VALID CONNECTION OBJECT */  
  
TYPE$TOKEN = RQ$GET$TYPE (DEVICE$CONNECTION$TOKEN, @STATUS);  
IF TYPE$TOKEN <> 101H THEN ERROR;
```

2-6

BASIC I/O SYSTEM CALLS FOR FILES



2-7

BASIC I/O EXERCISE

1) WRITE THE CODE NECESSARY TO WRITE A STRING OF DATA TO A FILE ON A SBC204 SD/SS FLOPPY

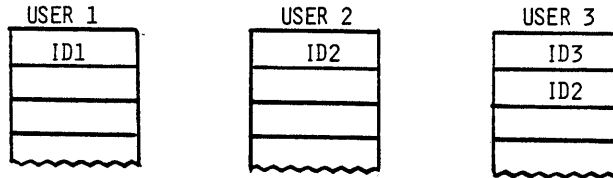
* GIVEN

- THE FLOPPY HAS ALREADY BEEN FORMATTED
- THE NAME OF THE DEVICE IS F0
- THE NAME OF THE FILE IS 'COMPANY/EMPLOYEE/PERSONAL'
- THE FILE ALREADY EXISTS
- THE DATA TO BE WRITTEN AT THE END OF THE FILE
- THE DATA IS 'L. JONES, 5050 MAIN DRAG, 3710217'

2-8

CONTROLLED ACCESS

- ONLY "NAMED FILES" PROVIDE CONTROLLED ACCESS TO FILES.
- CONTROL IS ACCOMPLISHED BY COMPARISON OF A USER ID\$STRUCTURE AND A FILE ACCESS LIST.



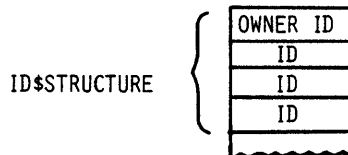
FILE ACCESS LIST

ID1	ALL
ID2	READ ONLY
ID3	WRITE ONLY

2-9

THE USER ID STRUCTURE

- IDENTIFYING INFORMATION ABOUT A USER (JOB OR HUMAN)
- EACH ID\$STRUCTURE CONTAINS AN ARRAY OF 16 BIT VALUES CALLED ID'S
- THE FIRST ID IN THE ARRAY IS CALLED THE OWNER ID
- THE REMAINING ID'S DEFINE THE GROUPS OF WHICH THE USER IS A MEMBER OF



2-10

USER TOKEN CREATION

- TO CREATE A USER TOKEN THE "SYSTEM PROGRAMMER" MAKES A CALL TO THE O.S. IN THE FORM

```
USER$TOKEN = RQ$CREATE$USER (@ID$STRUCT, @STATUS);
```

- E.G.

```
DECLARE ID$STRUCT STRUCTURE(LENGTH WORD,  
                             COUNT WORD,  
                             ID(*) WORD);
```

2-11

FILE ACCESS LIST

- A COLLECTION OF UP TO 3 PAIRS OF OWNER ID'S AND ACCESS MASKS
- THE ID'S REPRESENT USERS OR GROUP OF USERS
- THE ACCESS MASK REPRESENTS THE KINDS OF ACCESS TO THE FILE THAT THOSE USERS OR GROUPS OF USERS ARE ALLOWED

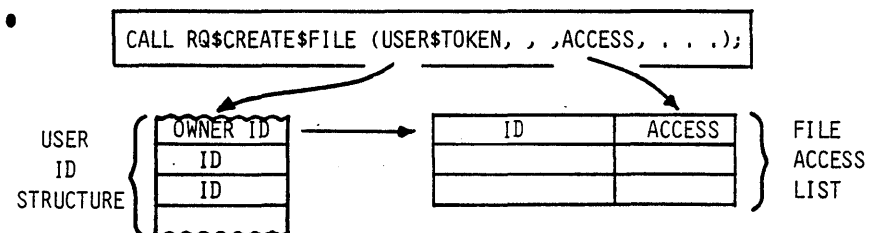
ACCESS LIST {	OWNER ID #1	ACCESS

- THE ACCESS LIST BELONGS TO THE FILE

2-12

ACCESS LIST CREATION

- TASKS CALLING CREATE\$FILE PASS AN ACCESS MASK AND A USER\$TOKEN
- THE I/O SYSTEM PAIRS THE "OWNER ID" AND "ACCESS MASK" AND APPENDS THE PAIR TO THE FILE ACCESS LIST



- ACCESS IS ALSO IMBEDDED IN THE FILE CONNECTION TOKEN RETURNED TO THE CALLING TASK VIA THE RESPONSE MAILBOX

2-13

ADDING ID'S TO THE ACCESS LIST

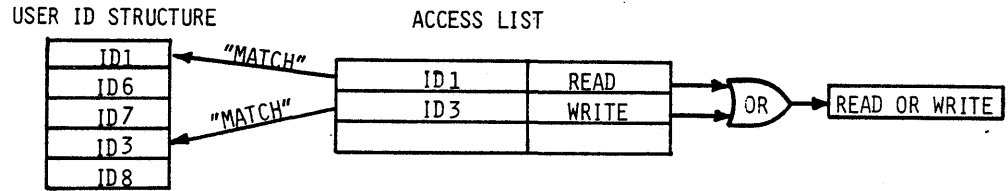
- THE RQ\$A\$CHANGE\$ACCESS SYSTEM CALL WILL CHANGE THE ACCESS RIGHTS TO A NAMED DATA OR DIRECTORY FILE.
- THE ID AND ACCESS SPECIFIED IN THE CALL WILL BE ADDED TO THE ACCESS LIST IF THE ID IS NOT FOUND IN THE LIST.
- THE FORM OF THE CALL IS

CALL RQ\$A\$CHANGE\$ACCESS (USER\$TOKEN, . . . , ID, ACCESS, . . .)

2-14

COMPUTATION OF ACCESS DURING ATTACH FILE

- THE I/O SYSTEM COMPARES THE ID'S IN THE USER ID STRUCTURE WITH THE ID'S IN THE ACCESS LIST.
- THE ACCESS MASKS CORRESPONDING TO MATCHING ID'S ARE LOGICALLY COMBINED, FORMING AN AGGREGATE MASK WITH COMBINED RIGHTS.



- THE COMPUTED ACCESS IS IMBEDDED IN THE FILE CONNECTION TOKEN RETURNED TO THE CALLING TASK VIA THE RESPONSE MAILBOX.

LABS

OBJECTIVES:

EXECUTE A STUDENT BASIC IO APPLICATION JOB IN AN RMX86 O.S. ENVIROMENT

INTRODUCE (BIOS) SYSTEM CALLS:

- RQ\$\$PHYSICAL\$ATTACH\$DEVICE
- RQ\$\$CREATE\$FILE
- RQ\$\$OPEN
- RQ\$\$READ
- RQ\$\$SEEK
- RQ\$\$WRITE
- RQ\$\$CLOSE
- RQ\$WAIT\$IO

CREATE SOURCE CODE:

- A SOURCE FILE NAMED START.P86
- A SOURCE FILE NAMED BIOLAB.P86

COMPILE (PLM86), LINK, AND LOCATE AN APPLICATION JOB, THAT WILL CALL UPON THE BIOS TO COMMUNICATE WITH A TERMINAL AND A FILE IN A FLOPPY

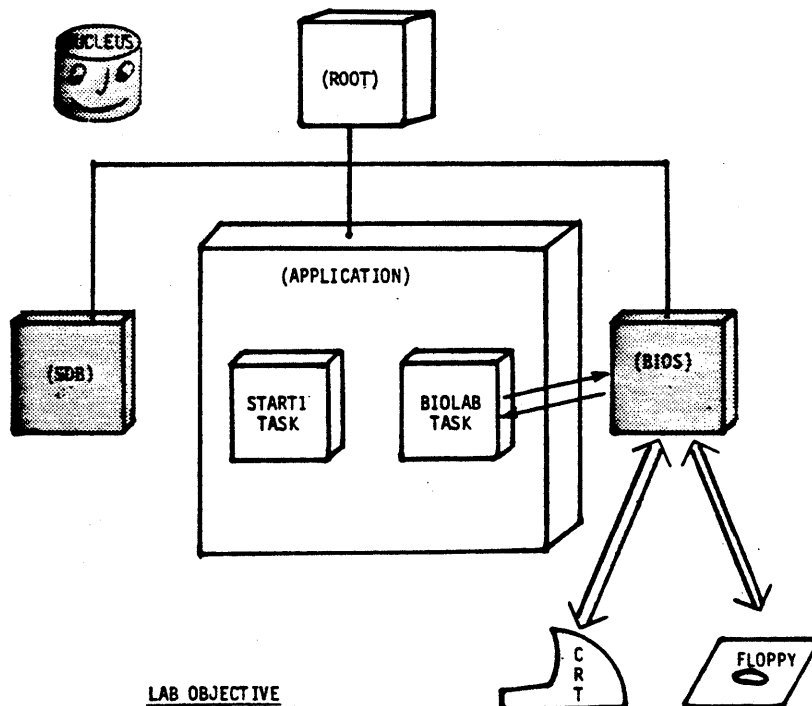
STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB1) AS THE LOGICAL NAME (:LAB:)

- AFILE /"TEAM NAME"/LAB1 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *

LAB 1



LAB OBJECTIVE

- Use Basic IO System Calls
- Read and write from/to a terminal & a floppy
- The student will be given the nucleus, BIOS and SDB

STEP2:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:BIOLAB.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:BIOLAB.P86

* THIS SOURCE FILE IS THE APPLICATION TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

```

-----
CREATE A BUFFER /* LENGTH=80
-----
CREATE A RESPONSE MAILBOX /* FIFO , HIGH PERFORMANCE
-----
PHYSICAL ATTACH TO DEVICE /* @(2,'TO') , PHYSICAL , *NOTE1
-----
PHYSICAL ATTACH TO DEVICE /* @(5,'WFDDO') , NAMED , *NOTE1
-----
CREATE FILE CONNECTION TO TERMINAL /* *NOTE1
-----
CREATE A USER TOKEN /* IDS=(2,2,0000H,OFFFH)
-----
CREATE FILE CONNECTION TO FLOPPY /* @(8,'LAB1DATA') , *NOTE1
-----
OPEN TERMINAL FILE /* (R/W) , SHARE ALL , *NOTE2
-----
OPEN FLOPPY FILE /* (R/W) , SHARE ALL , *NOTE2
-----
WRITE READY MESSAGE TO TERMINAL /* (USE RQ$WAIT$IO)
-----
ACTUAL = 80;
-----
DO WHILE ACTUAL GREATER THAN 2
-----
    READ FROM TERMINAL /* (USE RQ$WAIT$IO)
-----
    WRITE TO FLOPPY /* (USE RQ$WAIT$IO)
-----
    SEEK TO EOF MINUS ACTUAL /* (USE RQ$WAIT$IO)
-----
    READ FROM FLOPPY /* (USE RQ$WAIT$IO)
-----
    WRITE TO TERMINAL /* (USE RQ$WAIT$IO)
-----
CLOSE TERMINAL FILE /* *NOTE2
-----
CLOSE FLOPPY FILE /* *NOTE2
-----
** DELETE SELF **
-----

```

*NOTE1: WAIT FOR CONNECTION AND VALIDATE
 *NOTE2: WAIT FOR IORS , VALIDATE IORS.STATUS , AND DELETE SEGMENT

THE SOURCE CODE SUPPLIED DOES NOT VALIDATE CONNECTIONS OR IORS'S
 THE STUDENT MAY WISH TO IMPLEMENT THIS FUNCTIONALITY WHEN MODIFYING
 THE SOURCE CODE

STEP3:

- * ROOT JOBS ABSOLUTELY ADDRESS THE STARTING LOCATION OF THE STUDENT'S JOB CODE. THE ENTRY POINT MAY VARY IF INTERNAL PROCEDURES OR CHARACTER CONSTANTS ARE USED.
FOR THIS REASON IT IS ADVISABLE TO CREATE AND LINK A START TASK TO THE REST OF THE APPLICATION CODE TO FIX THE ENTRY POINT'S OFFSET INTO THE CODE
- * THIS APPLICATION JOB WILL BE A FIRST LEVEL JOB, THIS REQUIRES THAT A TASK WITHIN THIS JOB MAKE A CALL TO RQ\$END\$INIT\$TASK TO RESUME THE ROOT TASK
- * IN ORDER TO DEBUG OUR CODE BEFORE IT "CRASHES" WE MAY WISH TO INVOKE THE 957 MONITOR AT THE START OF OUR JOB'S EXECUTION. THIS CAN EASILY BE ACCOMPLISHED BY PLACING A "CAUSE\$INTERRUPT(3)" INSTRUCTION AT THE BEGINNING OF OUR CODE (IN OUR START TASK).

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:START.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:START.P86

- * THIS SOURCE FILE IS THE START TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

```
-----  
CALL RQ$END$INIT$TASK  
-----  
CAUSE$INTERRUPT(3)  
-----  
CREATE THE "COMMON$ENTRY" TASK */* PRI=155 , STACKSIZE = 512  
-----  
** DELETE SELF **  
-----
```


***** LAB ONE (BASIC IO SYSTEM) *****

STEP4:

COMPILE THE SOURCE FILES (START.P86 AND BIOLAB.P86)

- PLM86 :LAB:START.P86
- PLM86 :LAB:BIOLAB.P86

- * IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING
- * IF COMPILATION IS SUCCESSFUL THE COMPILER WILL CREATE FOR EACH OF THE SOURCE FILES:

- A LIST FILE NAMED ":LAB:(SOURCE).LST"
- AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 :LAB:START.OBJ,&
        :LAB:BIOLAB.OBJ,&
        /RMX5.0/DUTILS/EPIFL.LIB,&
        /RMX5.0/DUTILS/IPIFL.LIB,&
        /RMX5.0/DUTILS/RPIFL.LIB &
        TO :LAB:JOB.LNK &
        NOMAP
```

LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS

```
LOC86 :LAB:JOB.LNK &
        TO :LAB:LABJOB &
        SC(3) SEGSIZE(STACK(0)) &
        ORDER(CLASSES(CODE,DATA,STACK)) &
        ADDRESSES(CLASSES(CODE(1040H))) &
        NOINITCODE &
        OC(PURGE)
```

AND FINALLY ADD THE LOCATED MODULE TO THE OTHER PRECONFIGURED PARTS OF OUR SYSTEM

```
LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
EXIT
```

- * IN THE LINKING PROCESS OBSERVE THAT WE LINKED THE START MODULE FIRST
- * !!! NO WARNINGS OR ERRORS DURING LINK
- * !!! SOME WARNINGS ARE OK DURING LOCATE (SEE INSTRUCTOR)
- * :LAB:RMX86 IS A "GIVEN" FILE THAT CONTAINS:
 - A PRECONFIGURED NUCLEUS
 - A PRECONFIGURED BIOS
 - A PRECONFIGURED SDB
 - A PRECONFIGURED ROOT JOB
- * THE STUDENT MAY "OPTIONALLY" USE A "GIVEN" SUBMIT FILE THAT WILL COMPILE , LINK , LOCATE AND ADD THE FINAL MODULE TO THE SYSTEM

- SUBMIT :LAB:JOB.CSD

***** LAB ONE (BASIC IO SYSTEM) *****

STEP5:

* A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB1/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.
(COPY :LAB:RMX86 OVER :FDO:RMX86)

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NESSESARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

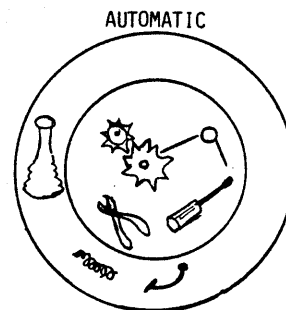
* GOOD LUCK...!

CHAPTER 3

RMX 86 EXTENDED I/O SYSTEM

-An Application Programmer's View

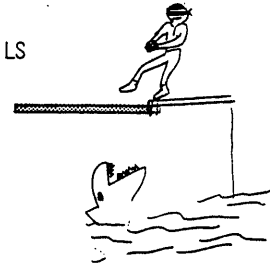
- WHY USE EIOS?
- ACCESSING NAMED FILES
- EIOS BUFFERING
- FILE CREATION



WHY USE THE EXTENDED I/O SYSTEM?



- REDUCE DEVELOPMENT COST
- AUTOMATIC BUFFERING
- SYNCHRONOUS SYSTEM CALLS
- FREES PROGRAMMER FROM TEDIOUS DETAILS



3-1

SYNCHRONOUS LEVEL I/O OPERATIONS (EIOS)

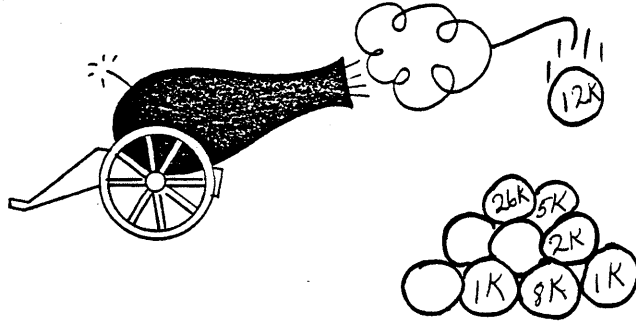
- PROGRAMMER DOES NOT HAVE TO USE RESPONSE MAILBOXES
- SYSTEM CALLS REQUIRE FEWER PARAMETERS
- NEED TO CHECK ONLY ONE STATUS AFTER THE CALL



3-2

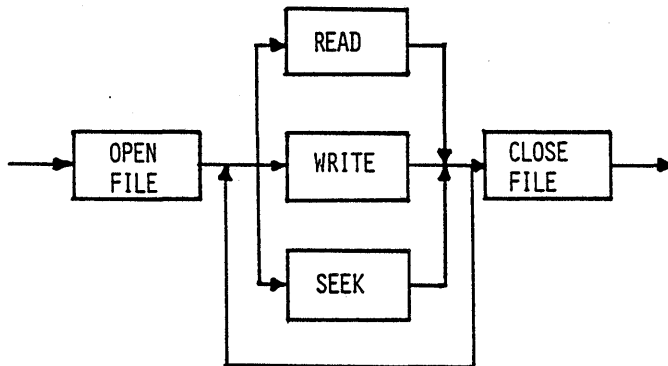
EIOS MEMORY REQUIREMENTS

- THE EIOS REQUIRES 12K BYTES ABOVE THE BIOS



3-3

ACCESSING NAMED FILES THROUGH EIOS



3-4

EIOS OPERATIONS

AFTER A FILE CONNECTION HAS BEEN ESTABLISHED,
THE RQ\$\$OPEN SYSTEM CALL OPENS A CONNECTION
FOR I/O OPERATIONS.

- THE RQ\$\$OPEN SPECIFIES
 - A FILE MAY BE
 - READ ONLY
 - WRITE ONLY
 - READ OR WRITE
 - NUMBER OF BUFFERS DESIRED
- COMMANDS ON OPEN FILES
 - READ
 - WRITE
 - SEED

3-5

OPENING A FILE

CALL RQ\$\$OPEN (FILE\$CONNECTION\$TOKEN, MODE, NUM\$BUF, @STATUS);

MODE: MODE OF ACCESS DESIRED

<u>VALUE</u>	<u>MODE</u>
1	OPEN FOR READING
2	OPEN FOR WRITING
3	OPEN FOR READING AND WRITING

- CONTROL IS RETURNED ONLY AFTER I/O HAS BEEN PERFORMED.

3-6

THE RQ\$\$READ\$MOVE SYSTEM CALL

- ALLOWS READING FROM AN OPEN FILE
- COUNT BYTES ARE READ STARTING AT FILE POINTER

ACTUAL = RQ\$\$READ\$MOVE (FILE\$CONNECTION\$TOKEN, @BUFFER, COUNT, @STATUS);

3-7

THE RQ\$\$WRITE\$MOVE SYSTEM CALL

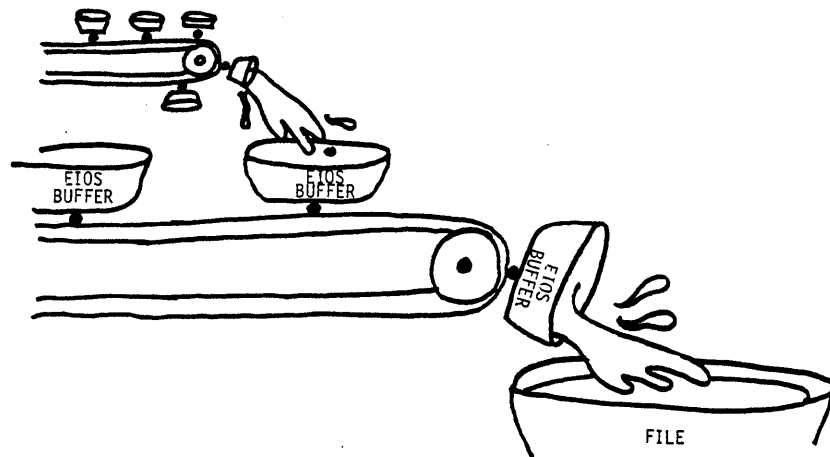
- ENABLES ANY NUMBER OF BYTES TO BE WRITTEN FROM A USER BUFFER INTO AN OPEN FILE
- THE DATA IS WRITTEN BEGINNING AT THE CURRENT SETTING OF THE FILE POINTER

ACTUAL = RQ\$\$WRITE\$MOVE (FILE\$CONNECTION\$TOKEN, @BUFFER, COUNT, @STATUS);

3-8

EIOS BUFFERING

- THE EIOS PROVIDES AUTOMATIC BUFFERING OF I/O OPERATIONS



3-9

BUFFERING METHODS

- ONE BUFFER ONLY
 - THE EIOS WILL WRITE OR READ INFORMATION ONE BUFFER AT A TIME (BLOCKING).
- TWO OR MORE BUFFERS
 - ALLOWS BLOCKING AND OVERLAPPED I/O BY USING READ-AHEAD, WRITE-BEHIND ALGORITHMS.
- ZERO BUFFERS
 - THE EIOS WILL ACCESS THE FILE EACH TIME THE APPLICATION READS OR WRITES TO THE FILE.

3-10

THE RQ\$\$SEEK SYSTEM CALL

- MOVES THE FILE POINTER FOR AN OPENED FILE TO ANY BYTE POSITION IN THE FILE
- HI\$PTR\$MOVE, LOW\$PTR\$MOVE = WORD PAIR CONTAINING A 32-BIT UNSIGNED NUMBER

CALL RQ\$\$SEEK (FILE\$CONNECTION\$TOKEN, MODE, HI\$PTR\$MOVE, LOW\$PTR\$MOVE, @STATUS);

<u>MODE</u>	<u>ACTION BY POINTER</u>
1	BACKWARD BY PTR\$MOVE (RELATIVE)
2	EQUAL TO PTR\$MOVE (ABSOLUTE)
3	FORWARD BY PTR\$MOVE (RELATIVE)
4	TO EOF MINUS PTR\$MOVE (ABSOLUTE)

3-11

THE RQ\$\$CLOSE SYSTEM CALL

- CLOSING AN OPEN FILE CONNECTION
- A FILE CONNECTION IS CLOSED BY THE PROGRAMMER
 - IF I/O OPERATIONS ARE COMPLETE
 - IF THE OPEN MODE OR SHARED STATUS IS TO BE CHANGED

CALL RQ\$\$CLOSE (FILE\$CONNECTION\$TOKEN, EXCEPT\$PTR);

3-12

EXAMPLE

```
ACTUAL = RQ$$READ$MOVE (FILE$CONNECTION$TOKEN, @BUFFER, 80, @STATUS);
```

```
IF STATUS < > 0 THEN CALL ERROR;
```

```
·  
·  
·
```

- THE USER ONLY NEEDS TO CHECK STATUS
- AFTER RETURNING FROM THE CALL THE BUFFER WILL CONTAIN THE INFORMATION

3-13

EIOS FILE CREATION

- TO CREATE A FILE THE USER MAKES A "CREATE\$FILE" SYSTEM CALL

```
FILE$CONNECTION$TOKEN = RQ$$CREATE$FILE (PATH$PTR, @STATUS);
```

SEE EIOS REFERENCE MANUAL FOR DETAILS ON PARAMETERS

3-14

EIOS FILE ATTACH

- IF THE FILE ALREADY EXISTS THEN THE USER MAKES AN "ATTACH\$FILE" SYSTEM CALL.

```
FILE$CONNECTION$TOKEN = RQ$$ATTACH$FILE (PATH$PTR, @STATUS);
```

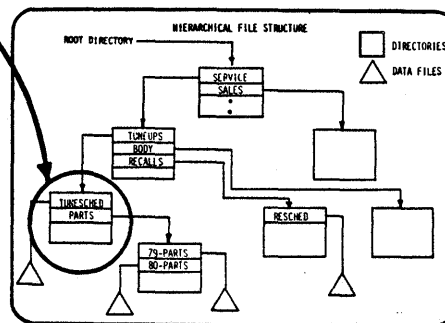
SEE EIOS REFERENCE MANUAL FOR DETAILS.

3-15

EXAMPLE

```
F$CONN = RQ$$ATTACH$FILE (@(19,':FO:SERVICE/TUNEUPS'), @STATUS);
```

- I NOW HAVE A CONNECTION TO THE DIRECTORY "TUNEUPS"



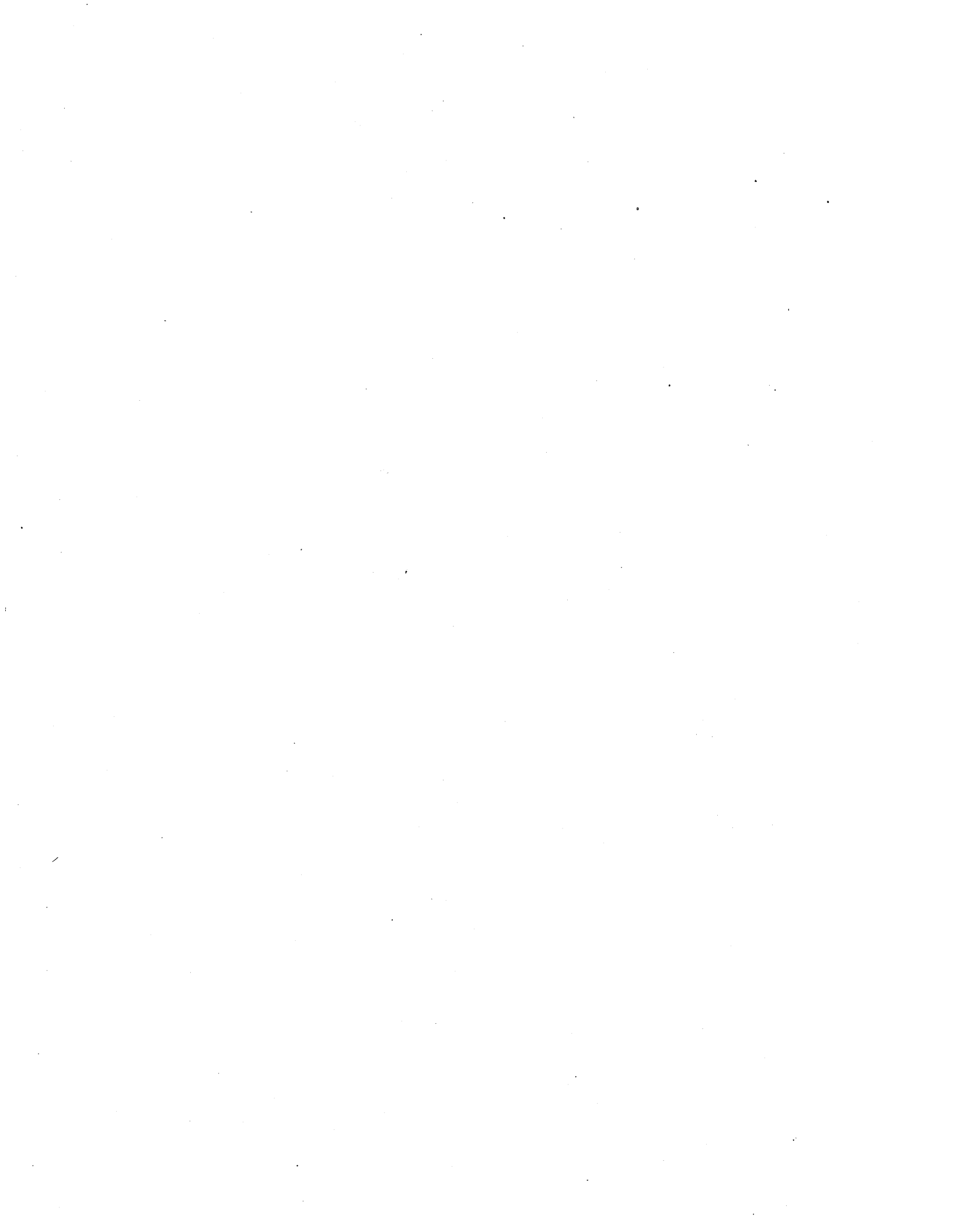
3-16

CHAPTER 4

RMX 86 EXTENDED I/O SYSTEM

-A System Programmer's View

- LOGICAL NAMES
- DEVICE CONNECTIONS
- IO JOBS
- DEFAULT TOKENS



Q: WHAT IS A LOGICAL NAME

- IF I CATALOG THE DEVICE CONNECTION TOKEN IN MY JOB'S DIRECTORY UNDER AN ASCII NAME, THEN THAT NAME WILL BE KNOWN TO THE EIOS AS THE LOGICAL NAME FOR THAT FILE CONNECTION.

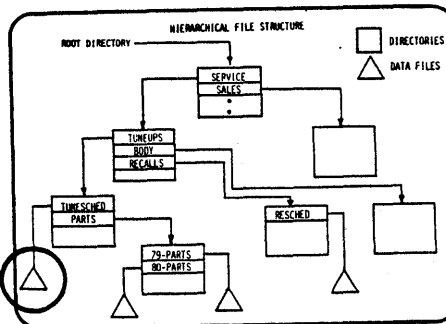
CALL RQ\$CATALOG\$OBJECT(Ø, F\$TOKEN, @ (4, 'AUTO'), @STATUS)

JOB DIRECTORY	
ASCII NAME	OBJECT TOKEN
'INTE\$6\$TASK'	8C58
'AUTO'	945C
⋮	⋮

EIOS AND LOGICAL NAMES

- PLACING COLONS AROUND AN ASCII STRING IDENTIFIES A LOGICAL NAME TO THE EIOS

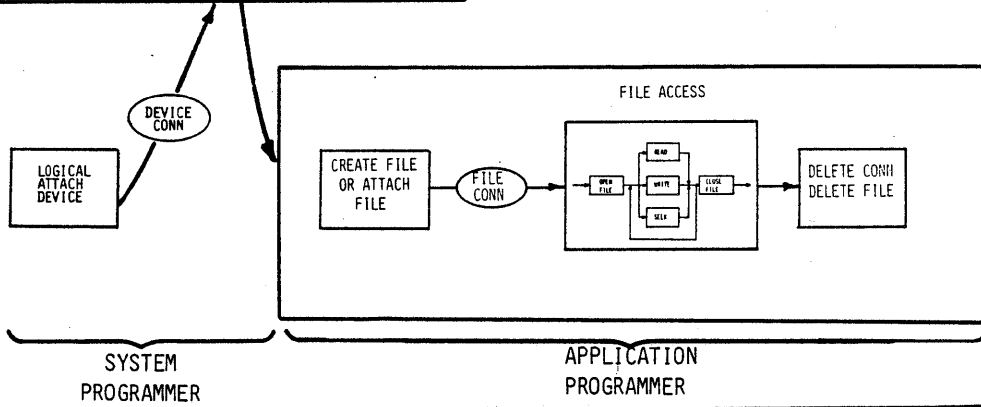
F\$CONN\$1 = RQ\$\$ATTACH\$FILE(@ (15, ':AUTO:TUNESCHED', @STATUS);



THE EIOS DEVICE CONNECTION

ROOT JOB DIRECTORY	
ASCII NAME	OBJECT TOKEN
'INTE\$6\$TASK'	8C58
'MBX1'	945C
'F0' :	846C :
:	:

- THE EIOS PHYSICALLY ATTACHES TO THE DEVICE
- THEN CATALOGS THE DEVICE CONNECTION TOKEN UNDER A LOGICAL NAME



4-3

EIOS LOGICAL ATTACHMENT

- LOGICAL ATTACH IS ACCOMPLISHED THROUGH

```
CALL RQ$$LOGICAL$ATTACH$DEVICE (LOG$NAME, DRV$NAME, FILE$DRIVER, @STATUS);
```

E. G.

```
CALL RQ$A$LOGICAL$ATTACH$DEVICE (a(4:'F0:'), a(2,'F0'), 4, @STATUS);
```

4-4

THE IO JOB

- THE IO JOB DIFFERS FROM A NORMAL JOB
 - THERE ARE THREE ENTRIES IN THE OBJECT DIRECTORY OF THE JOB UNDER THE NAMES "RQGLOBAL", "\$", AND "R?USER"

JOB DIRECTORY	
ASCII NAME	OBJECT TOKEN
'RQGLOBAL'	8C58
'\$'	945C
'R?USER'	9818
:	:
:	:

4-5

IO JOB CREATION

TO CREATE AN IO JOB

```
JOB$TOKEN = RQ$CREATE$IO$JOB (POOL$MIN, POOL$MAX,  
                                EXCEPT$HANDLER, JOB$FLAGS,  
                                TASK$PRIORITY, START$ADDRESS,  
                                DATA$SEG, STACK$PTR,  
                                STACK$SIZE, TASK$FLAGS,  
                                MSG$MBOX, @STATUS);
```

REFER TO EXTENDED IO SYSTEM REFERENCE MANUAL.

4-6

I/O JOB DELETION

- TO DELETE AN I/O JOB

```
CALL RQ$EXIT$I0$JOB (USER$FAULT$CODE,  
                    RETURN$DATA$PTR,  
                    @STATUS);
```

REFER TO EXTENDED IO SYSTEM REFERENCE MANUAL.

4-7

"CATCH 22"

- AN I/O JOB CAN ONLY BE CREATED BY AN I/O JOB
- THE SYSTEM PROGRAMMER DEFINES I/O JOBS DURING CONFIGURATION OF THE EXTENDED I/O SYSTEM (TO BE DISCUSSED LATER)
- THESE I/O JOBS WILL BE CHILD JOBS OF THE EIOS

4-8

EIOS LOGICAL NAME SEARCH SEQUENCE

- THE EIOS SEARCHES THREE OBJECT DIRECTORIES FOR THE LOGICAL NAME
 - FIRST, THE OBJECT DIRECTORY OF THE LOCAL JOB
 - SECOND, THE OBJECT DIRECTORY OF THE GLOBAL JOB
 - A JOB THAT HAS MORE "SCOPE" THAN THE LOCAL JOB BUT LESS "SCOPE" THAN THE ROOT JOB
 - THIRD, THE OBJECT DIRECTORY OF THE ROOT JOB

4-9

THE DEFAULT PREFIX

- THE DEFAULT PREFIX IS A DEVICE OR FILE CONNECTION TOKEN
- IT IS CATALOGED IN THE LOCAL JOB DIRECTORY UNDER THE ASCII NAME "\$"
- IF A TASK FAILS TO SPECIFY A CONNECTION DURING A SYSTEM CALL THAT REQUIRES IT, THE EIOS USES THE DEFAULT PREFIX

4-10

THE DEFAULT USER

- THE DEFAULT USER IS A USER TOKEN
- IT IS CATALOGUED IN THE LOCAL JOB DIRECTORY UNDER THE ASCII NAME "R?USER"
- THE EIOS PERFORMS ALL OF THE I/O OPERATIONS WITHIN A JOB ON BEHALF OF ONE USER TOKEN

4-11

EIOS EXERCISE

1) WRITE THE CODE NECESSARY TO WRITE A STRING OF DATA TO A FILE ON A SBC204 SD/SS FLOPPY

- * GIVEN
 - THE FLOPPY HAS ALREADY BEEN FORMATTED
 - THE NAME OF THE DEVICE IS FØ
 - THE NAME OF THE FILE IS 'COMPANY/EMPLOYEE/PERSONAL'
 - THE FILE ALREADY EXISTS
 - THE DATA TO BE WRITTEN AT THE END OF THE FILE
 - THE DATA IS 'L. JONES, 5050 MAIN DRAG, 3710217'

4-12

OBJECTIVES:

EXECUTE A STUDENT EXTENDED IO APPLICATION JOB IN AN RMX86 O.S. ENVIRONMENT

INTRODUCE (EIOS) SYSTEM CALLS:

- RQ\$\$\$CREATE\$FILE
- RQ\$\$\$OPEN
- RQ\$\$\$READ\$MOVE
- RQ\$\$\$SEEK
- RQ\$\$\$WRITE\$MOVE
- RQ\$\$\$CLOSE
- RQ\$EXIT\$IO\$JOB

CREATE SOURCE CODE:

- A SOURCE FILE NAMED START.P86
- A SOURCE FILE NAMED EIOLAB.P86

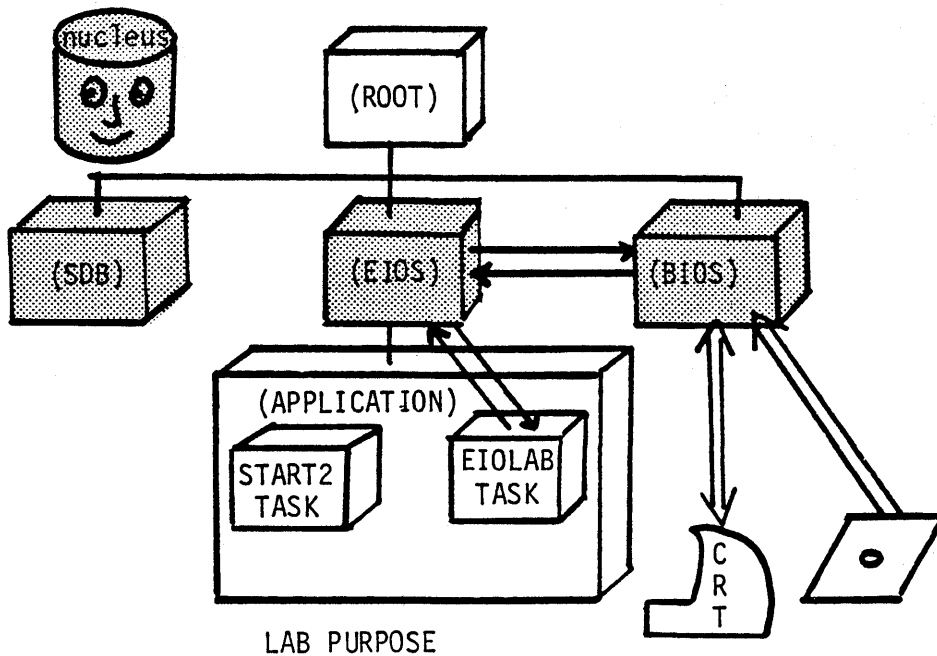
COMPILE (PLM86), LINK, AND LOCATE AN APPLICATION JOB, THAT WILL CALL UPON THE EIOS TO COMMUNICATE WITH A TERMINAL AND A FILE IN A FLOPPY

STEP 1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED (/ "TEAM NAME" /LAB2) AS THE LOGICAL NAME (:LAB:)

- AFILE / "TEAM NAME" /LAB2 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



- Use extended io system calls
- Read and write to/from a terminal and a floppy
- The student will be given the nucleus, BIOS, EIOS AND SDB
- The student will supply the LAB2JOB and LAB2RJB

STEP2:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:EIOLAB.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:EIOLAB.P86

* THIS SOURCE FILE IS THE APPLICATION TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

```

-----
CREATE A BUFFER /* LENGTH=80
-----
CREATE FILE CONNECTION TO TERMINAL /* @(4,':TO:') , *NOTE1
-----
CREATE FILE CONNECTION TO FLOPPY /* @(13,':FDO:LAB2DATA') , *NOTE1
-----
OPEN TERMINAL FILE /* (R/W) , SHARE ALL , *NOTE1
-----
OPEN FLOPPY FILE /* (R/W) , SHARE ALL , *NOTE1
-----
WRITE READY MESSAGE TO TERMINAL /*
-----
ACTUAL = 80;
-----
DO WHILE ACTUAL GREATER THAN 2
-----
    READ FROM TERMINAL /*
-----
    WRITE TO FLOPPY /*
-----
    READ FROM FLOPPY /*
-----
    SEEK TO EOF MINUS ACTUAL /*
-----
    WRITE TO TERMINAL /*
-----
CLOSE TERMINAL FILE /* *NOTE1
-----
CLOSE FLOPPY FILE /* *NOTE1
-----
** DELETE SELF ** CALL EXIT$IO$JOB
-----
*NOTE1: VALIDATE BY CHECKING STATUS = E$OK

THE SOURCE CODE SUPPLIED DOES NOT VALIDATE CONNECTIONS
THE STUDENT MAY WISH TO IMPLEMENT THIS FUNCTIONALITY WHEN MODIFYING
THE SOURCE CODE
    
```


STEP3:

- * ROOT JOBS ABSOLUTELY ADDRESS THE STARTING LOCATION OF THE STUDENT'S JOB CODE. THE ENTRY POINT MAY VARY IF INTERNAL PROCEDURES OR CHARACTER CONSTANTS ARE USED.
FOR THIS REASON IT IS ADVISABLE TO CREATE AND LINK A START TASK TO THE REST OF THE APPLICATION CODE TO FIX THE ENTRY POINT'S OFFSET INTO THE CODE
- * THIS APPLICATION JOB WILL BE A SECOND LEVEL JOB. A TASK WITHIN THIS JOB IS NOT REQUIRED TO MAKE A CALL TO RQ\$END\$INIT\$TASK, THE EIOS CODE SUPPLIES A TASK THAT CALLS RQ\$END\$INIT\$TASK
- * IN ORDER TO DEBUG OUR CODE BEFORE IT "CRASHES" WE MAY WISH TO INVOKE THE 957 MONITOR AT THE START OF OUR JOB'S EXECUTION.
THIS CAN EASILY BE ACCOMPLISHED BY PLACING A "CAUSE\$INTERRUPT(3)" INSTRUCTION AT THE BEGINNING OF OUR CODE (IN OUR START TASK).

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:START.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:START.P86

- * THIS SOURCE FILE IS THE START TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

CAUSE\$INTERRUPT(3)

CREATE THE "COMMON\$ENTRY" TASK */* PRI=155 , STACKSIZE = 512

** DELETE SELF **

***** LAB TWO (EXTENDED IO SYSTEM) *****

STEP4:

COMPILE THE SOURCE FILES (START.P86 AND EIOLAB.P86)

- PLM86 :LAB:START.P86
- PLM86 :LAB:EIOLAB.P86

- * IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING
- * IF COMPILATION IS SUCCESSFUL THE COMPILER WILL CREATE FOR EACH OF THE SOURCE FILES:

- A LIST FILE NAMED ":LAB:(SOURCE).LST"
- AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 :LAB:START.OBJ,&
        :LAB:EIOLAB.OBJ,&
        /RMX5.0/DUTILS/EPIFL.LIB,&
        /RMX5.0/DUTILS/IPIFL.LIB,&
        /RMX5.0/DUTILS/RPIFL.LIB &
        TO :LAB:JOB.LNK &
        NOMAP
```

LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS

```
LOC86 :LAB:JOB.LNK &
        TO :LAB:LABJOB &
        SC(3) SEGSIZE(STACK(0)) &
        ORDER(CLASSES(CODE,DATA,STACK)) &
        ADDRESSES(CLASSES(CODE(1040H))) &
        NOINITCODE &
        OC(PURGE)
```

AND FINALLY ADD THE LOCATED MODULE TO THE OTHER PRECONFIGURED PARTS OF OUR SYSTEM

```
LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
EXIT
```

- * IN THE LINKING PROCESS OBSERVE THAT WE LINKED THE START MODULE FIRST
- * !!! NO WARNINGS OR ERRORS DURING LINK
- * !!! SOME WARNINGS ARE OK DURING LOCATE (SEE INSTRUCTOR)
- * :LAB:RMX86 IS A "GIVEN" FILE THAT CONTAINS:
 - A PRECONFIGURED NUCLEUS
 - A PRECONFIGURED BIOS
 - A PRECONFIGURED EIOS
 - A PRECONFIGURED SDB
 - A PRECONFIGURED ROOT JOB
- * THE STUDENT MAY "OPTIONALLY" USE A "GIVEN" SUBMIT FILE THAT WILL COMPILE , LINK , LOCATE AND ADD THE FINAL MODULE TO THE SYSTEM

- SUBMIT :LAB:JOB.CSD

***** LAB TWO (EXTENDED IO SYSTEM) *****

STEP5:

* A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB2/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.
(COPY :LAB:RMX86 OVER :FDO:RMX86)

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NESSESARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

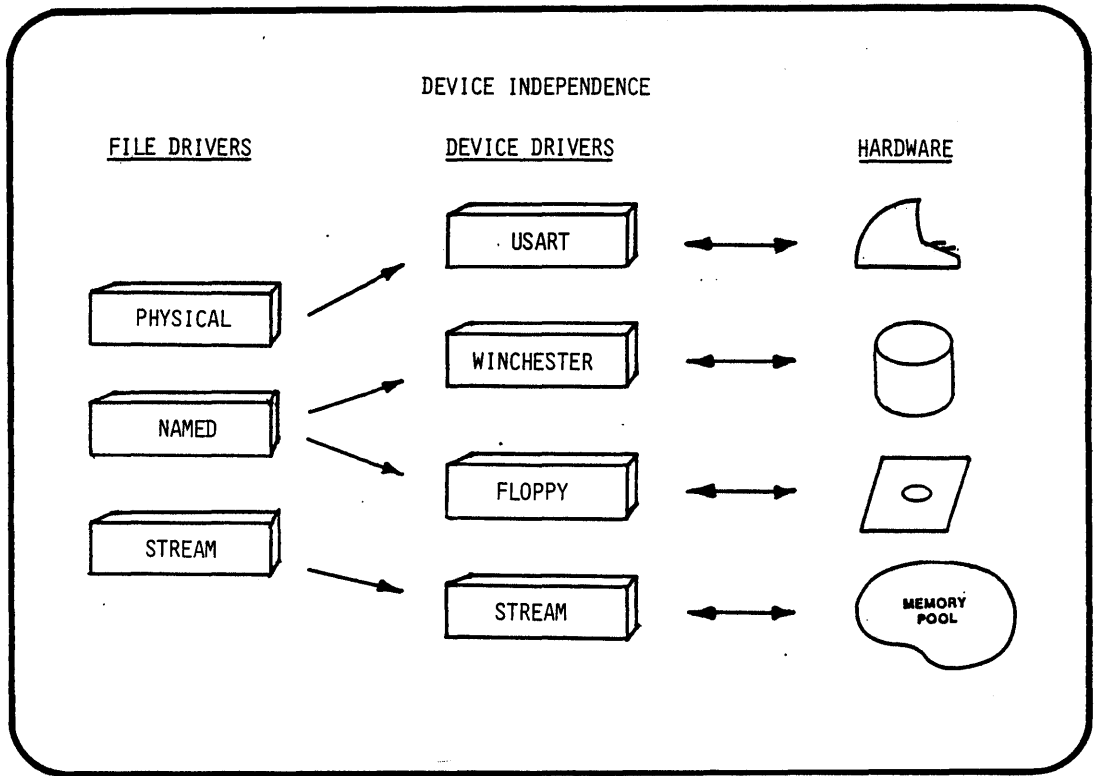
* GOOD LUCK...!

CHAPTER 5

WRITING DEVICE DRIVERS

-Generalities

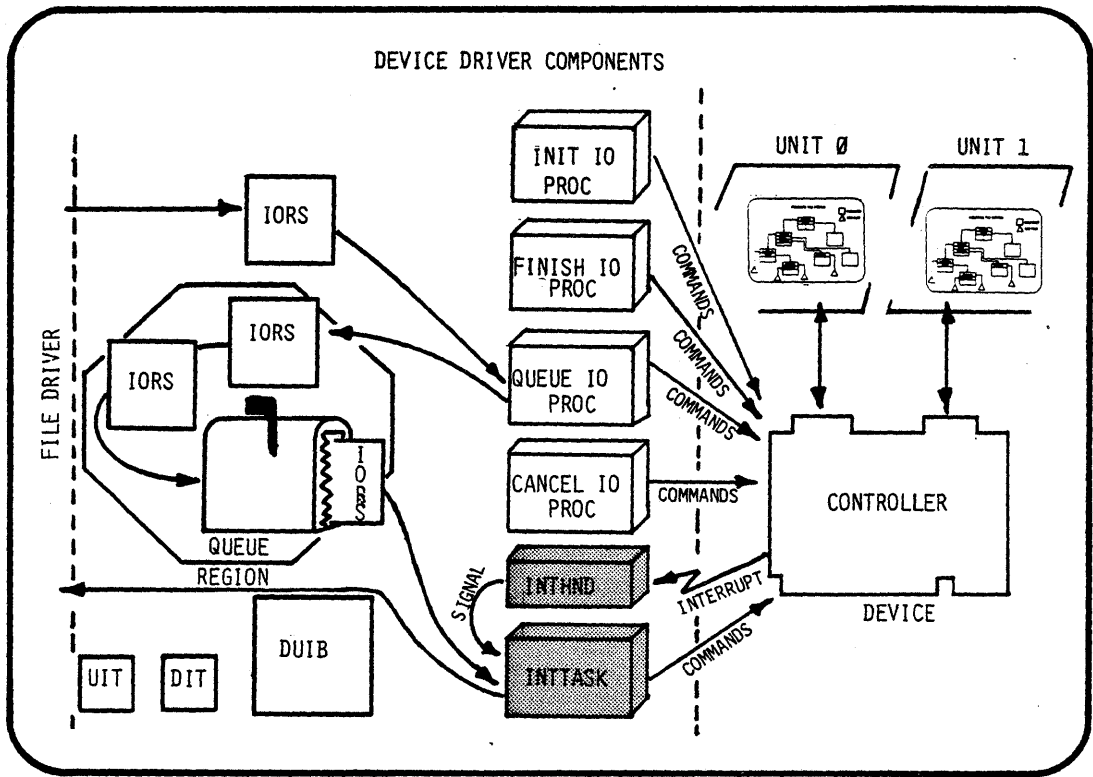
- INDEPENDENT IO
- DRIVER COMPONENTS.
- THE DUIB
- THE I/O REQUEST
- DRIVER FUNCTIONS



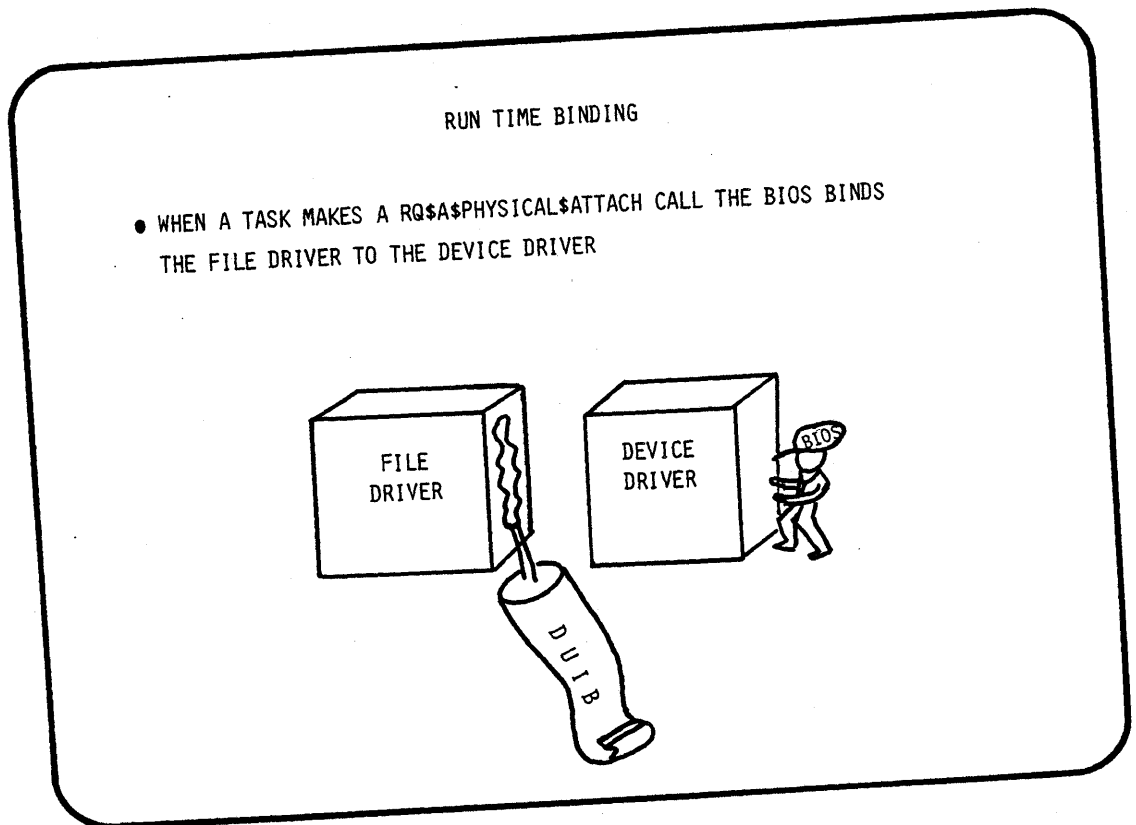
- DEVICE INDEPENDENT I/O**
- APPLICATION TASKS COMMUNICATE WITH FILE DRIVERS
 - THIS ALLOWS TASKS TO MANIPULATE ALL FILES IN THE SAME MANNER

 - FILE DRIVERS COMMUNICATE WITH DEVICE DRIVERS
 - THEY PROVIDE THE INTERFACE BETWEEN SOFTWARE AND HARDWARE

 - THIS STANDARD INTERFACE HAS ADVANTAGES
 - THE HARDWARE CAN BE CHANGED WITHOUT EXTENSIVE MODIFICATION TO THE SOFTWARE
 - THE I/O SYSTEM CAN SUPPORT A GREATER RANGE OF DEVICES.
- 5-2

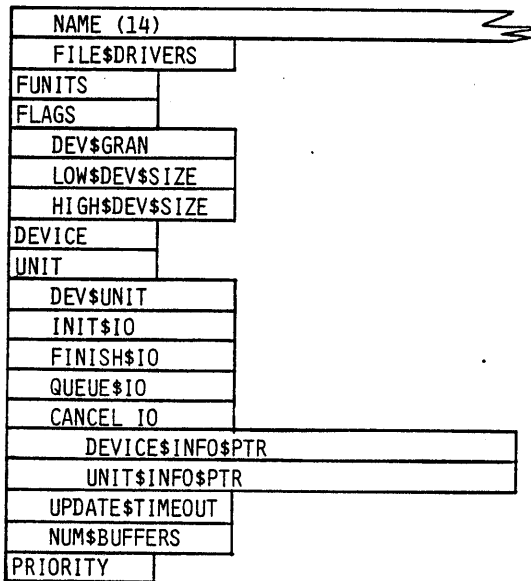


5-3



5-4

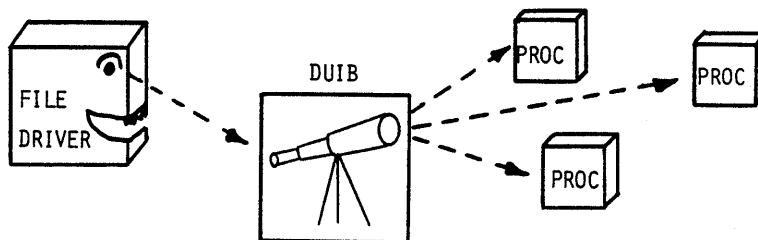
THE DUIB STRUCTURE
(DEVICE UNIT INFORMATION BLOCK)



*REFER TO "GUIDE TO WRITING DEVICE DRIVERS" REFERENCE MANUAL.

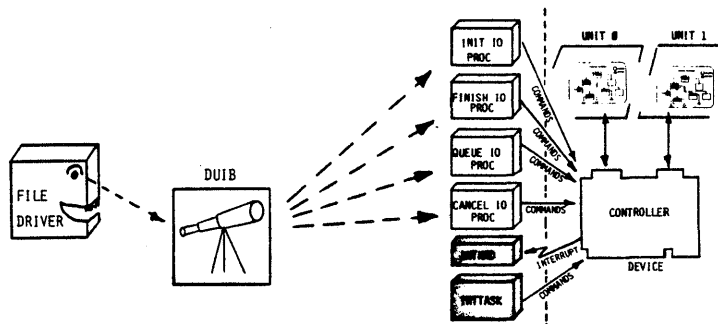
THE DUIB

- THE DEVICE UNIT INFORMATION BLCK IS A TABLE OF VALUES DESCRIBING THE COMPONENTS OF A DEVICE DRIVER.
- IT IS CREATED BY THE SYSTEM PROGRAMMER DURING CONFIGURATION OF THE BASIC I/O.
- THE DUIB BINDS THE FILE DRIVER TO THE DEVICE DRIVER BY DESCRIBING POINTERS TO THE DEVICE DRIVER PROCEDURES.



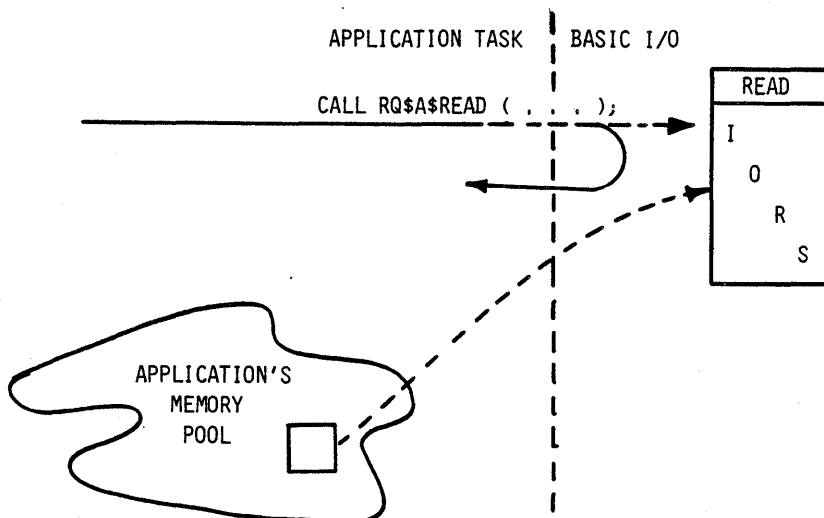
THE DEVICE DRIVER "PROCEDURES"

- THE DEVICE DRIVER PROCEDURES ARE CALLED BY THE FILE DRIVER
- THESE PROCEDURES
 - CONTAIN THE CODE NECESSARY TO COMMUNICATE WITH THE HARDWARE
 - MAINTAIN THE QUEUE
- THE ADDRESSES OF THESE PROCEDURES ARE IMBEDDED IN THE DEVICE UNIT INFORMATION BLOCK (DUIB)



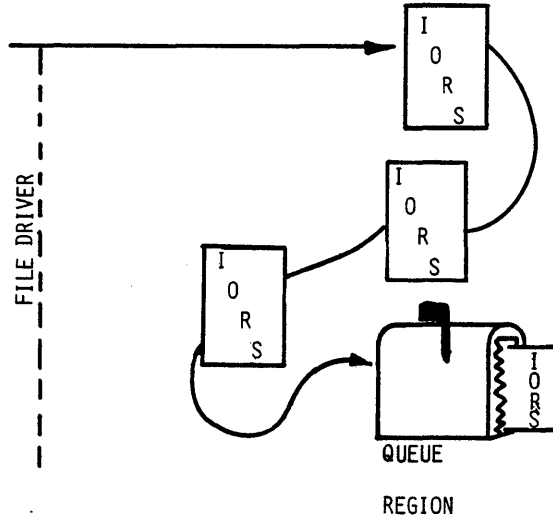
THE I/O REQUEST

- WHEN THE APPLICATION TASK CALLS THE BASIC I/O SYSTEM AN IORS IS GENERATED (I/O REQUEST SEGMENT)



THE REQUEST QUEUE

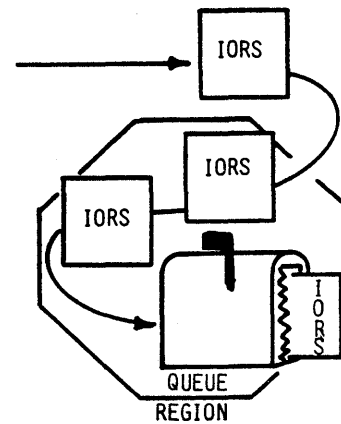
- THE FILE DRIVER SENDS IORS TO THE DEVICE DRIVER (QUEUE IO PROC)
- IF THE DEVICE DRIVER IS "BUSY", THE IORS IS PLACED AT THE END OF THE QUEUE



5-9

THE REGION

- PROTECTION OF THE QUEUE IS ACCOMPLISHED THROUGH A REGION
 - THE TASK CALLS THE IRMX O.S. TO GAIN ACCESS TO THE QUEUE
 - THE TASK MANIPULATES OBJECTS IN THE QUEUE
 - THE TASK THEN CALLS THE IRMX O.S. TO RELEASE ACCESS TO THE QUEUE

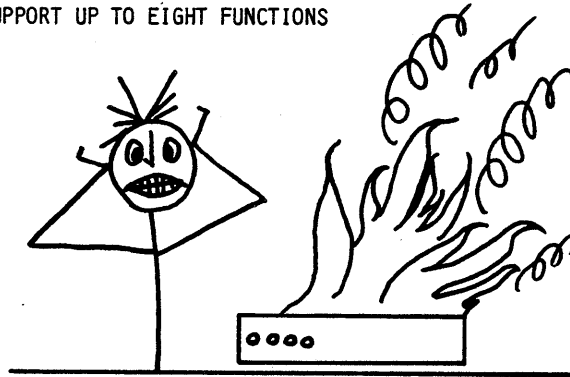


5-10

DEVICE DRIVER FUNCTIONS

- A DEVICE DRIVER MAY SUPPORT UP TO EIGHT FUNCTIONS

- READ
- WRITE
- SEEK
- SPECIAL
- ATTACH
- DETACH
- OPEN
- CLOSE

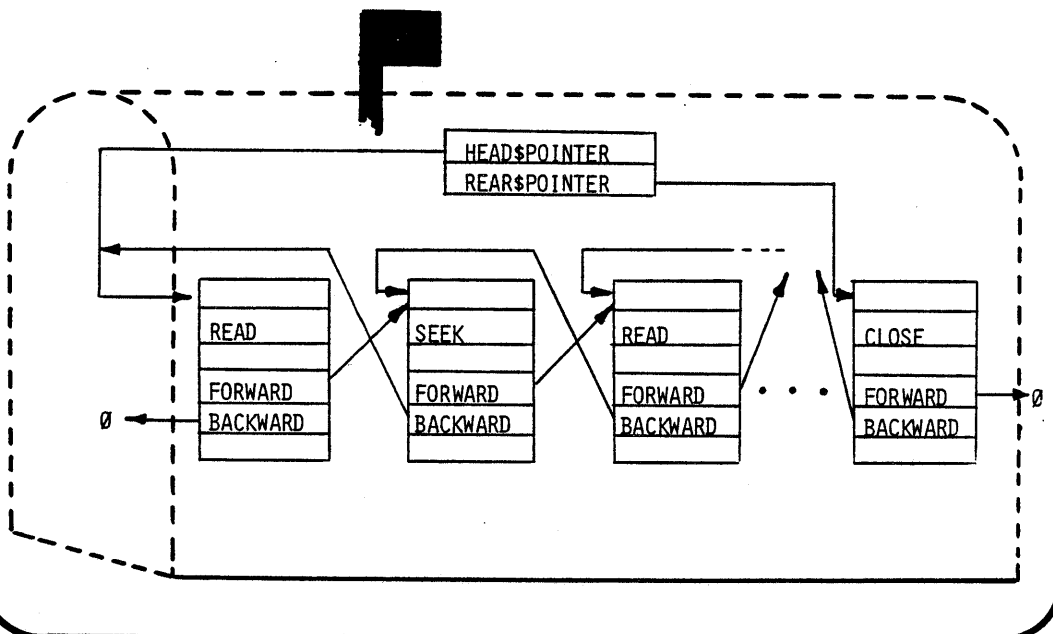


SPECIAL FUNCTION = SCF
(STOP & CATCH FIRE!)

5-11

QUEUE IMPLEMENTATION

- THE IORS STRUCTURE CONTAINS FORWARD AND BACKWARD POINTER FIELDS



5-12

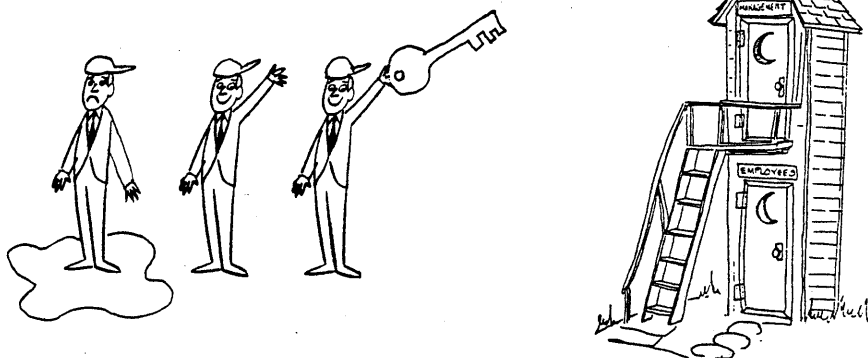
THE IORS STRUCTURE
(INPUT OUTPUT REQUEST SEGMENT)

STATUS	
UNIT\$STATUS	
ACTUAL	
ACTUAL\$FILL	
DEVICE	
UNIT	
FUNCT	
SUB\$FUNCT	
LOW\$DEV\$LOC	
HIGH\$DEV\$LOC	
BUF\$PTR	
COUNT	
COUNT\$FILL	
AUX\$PTR	
LINK\$FOR	
LINK\$BACK	
RESP\$MBOX	
DONE	
FILL	
CANCEL ID	

*REFER TO "GUIDE TO WRITING DEVICE DRIVERS" REFERENCE MANUAL.

QUEUE PROTECTION

- THE QUEUE IS A SHARED RESOURCE OF OTHER DEVICE DRIVER COMPONENTS
- THE QUEUE CAN ONLY BE ACCESSED BY ONE OF THE DEVICE DRIVER COMPONENTS AT A TIME
- THAT COMPONENT MUST HAVE A KEY TO USE THE QUEUE



SOME FACTS ABOUT REGIONS

- ONCE A TASK GAINS ACCESS TO A REGION
 - THE PRIORITY OF THE TASK MAY BE TEMPORARILY RAISED
 - THIS OCCURS AUTOMATICALLY IF THE REGION IS PRIORITY BASED, AND THERE IS A TASK OF HIGHER PRIORITY WAITING TO USE THE REGION
 - THE TASK CANNOT BE SUSPENDED OR DELETED UNTIL IT SURRENDERS ACCESS TO THE REGION

5-15

SYSTEM CALLS FOR REGIONS

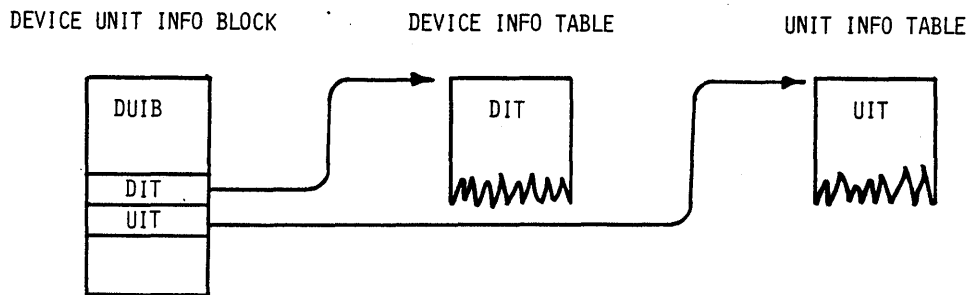
- REGION = RQ\$CREATE\$REGION (FLAGS, @STATUS);
- CALL RQ\$SEND\$CONTROL (@STATUS);
- CALL RQ\$RECEIVE\$CONTROL (REGION,@STATUS);
- CALL RQ\$ACCEPT\$CONTROL (REGION,@STATUS);
- CALL RQ\$DELETE\$REGION (REGION,@STATUS);

REFER TO SYSTEM PROGRAMMER'S REFERENCE MANUAL

5-16

THE (DIT) AND (UIT)

- ALL DUIB'S ARE FIXED LENGTH
- SOME DEVICE DRIVERS NEED MORE INFORMATION
- THE SYSTEM PROGRAMMER MAY "OPTIONALLY" PROVIDE TWO EXTRA TABLES OF UNBOUND LENGTH



- THE DUIB CONTAINS POINTERS TO THESE TABLES

5-17

EXERCISE

- WHY ARE THE ADDRESS FIELDS IN THE DUIB, FOR THE DRIVER PROCEDURES, ONLY WORD FIELDS AND NOT POINTER FIELDS?
- NAME SOME GENERAL DEVICE DRIVER COMPONENTS. _____

- NAME ONE ADVANTAGE OF A REGION vs. SEMAPHORE FOR RESOURCE PROTECTION.

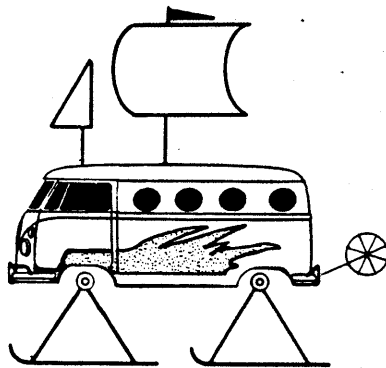
5-18

DEVICE DRIVER TYPES

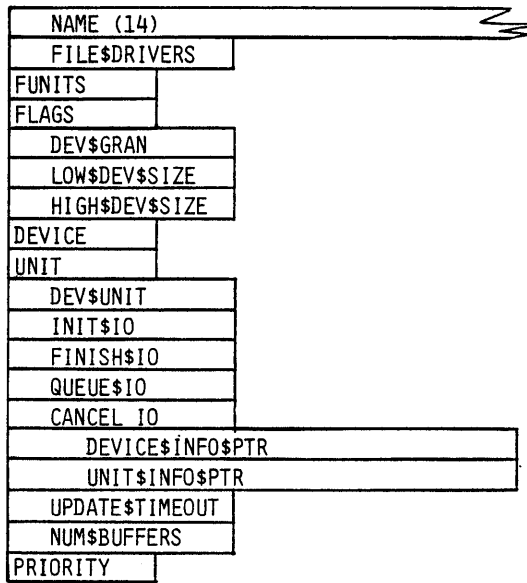
- THERE ARE THREE TYPES OF DEVICE DRIVERS IN THE iRMX ENVIRONMENT
 - COMMON DEVICE DRIVER
 - RANDOM ACCESS DEVICE DRIVER
 - CUSTOM DEVICE DRIVER

CHAPTER 6

THE CUSTOM DEVICE DRIVER



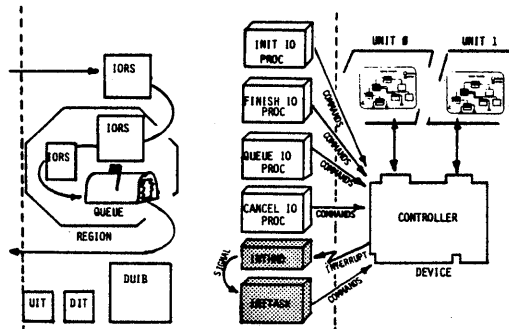
THE DUIB STRUCTURE
(DEVICE UNIT INFORMATION BLOCK)



*REFER TO "GUIDE TO WRITING DEVICE DRIVERS" REFERENCE MANUAL.

THE CUSTOM DEVICE DRIVER

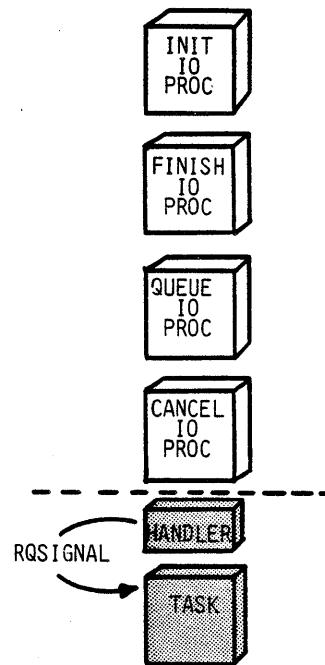
- THE CUSTOM DEVICE DRIVER IS NEEDED IF
 - TWO OR MORE INTERRUPT LEVELS PER DEVICE ARE REQUIRED
 - PRIORITY ORDERED REQUEST QUEUE IS REQUIRED
 - THE DEVICE DOES NOT FIT INTO THE COMMON RANDOM ACCESS DEVICE CATEGORY (TO BE DISCUSSED LATER)



CUSTOM DEVICE DRIVER COMPONENTS

● TO WRITE A CUSTOM DEVICE DRIVER YOU MUST PROVIDE

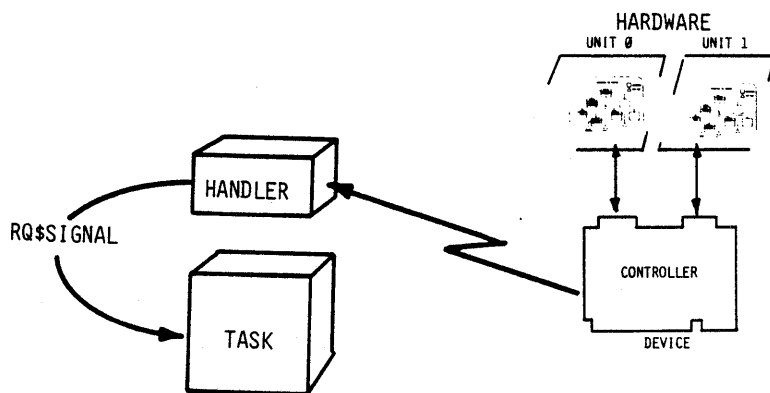
- AN INITIALIZE I/O PROCEDURE
- A FINISH I/O PROCEDURE
- A QUEUE I/O PROCEDURE
- A CANCEL I/O PROCEDURE
- A DEVICE INTERRUPT TASK AND HANDLER



6-3

THE DEVICE INTERRUPT HANDLER

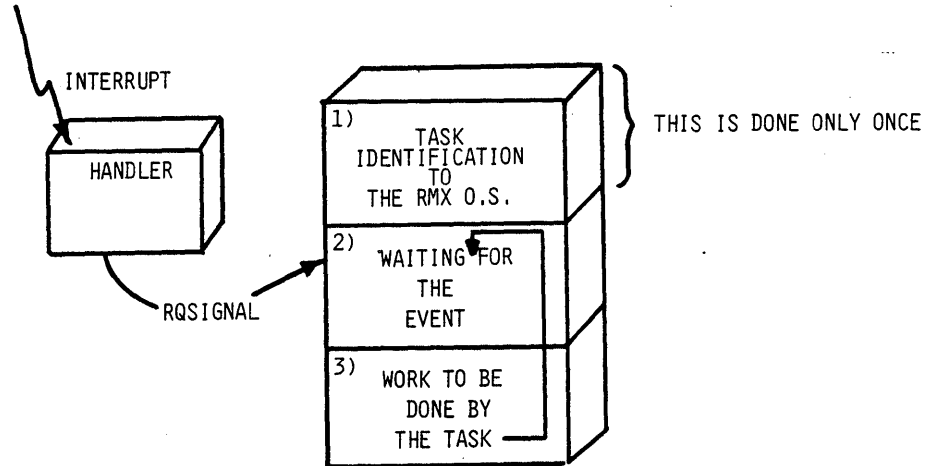
- EXAMPLE INTERRUPT HANDLER
 - WHEN THE DEVICE CONTROLLER FINISHES A REQUEST (READ,SEEK,...,WRITE) IT GENERATES AN INTERRUPT.
 - THE INTERRUPT HANDLER THEN SIGNALS THE INTERRUPT TASK



6-4

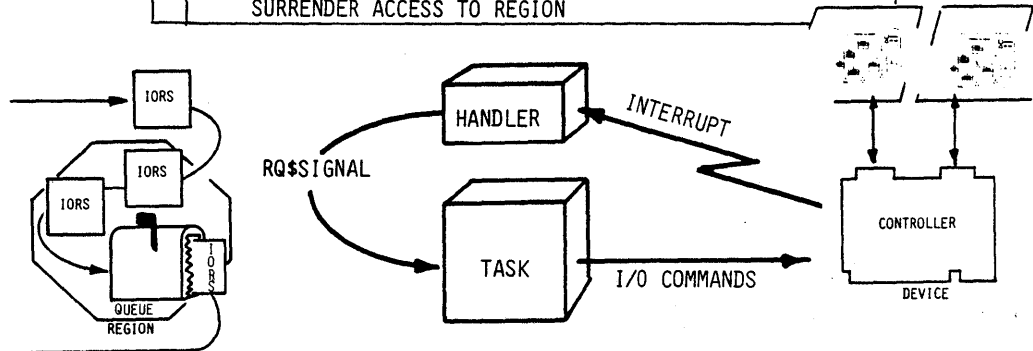
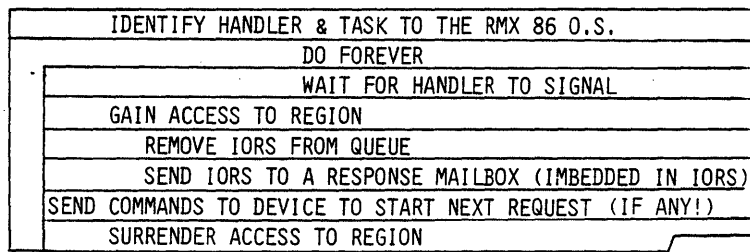
INTERRUPT TASK

- THERE ARE 3 PARTS TO AN INTERRUPT TASK



6-5

EXAMPLE DEVICE INTERRUPT TASK



Q: HOW DOES THE FIRST REQUEST GET STARTED?

6-6

THE QUEUE I/O PROCEDURE

- THE BASIC I/O SYSTEM CALLS THE QUEUE I/O PROCEDURE IN THE FOLLOWING MANNER.

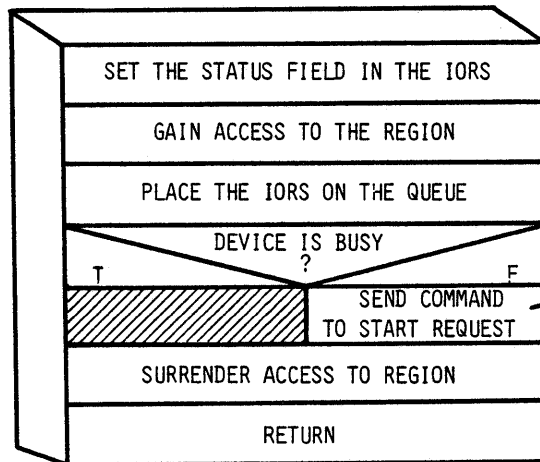
CALL QUEUE\$I0(IORS\$T,DUIB\$P,D\$DATA\$T);

WHERE:

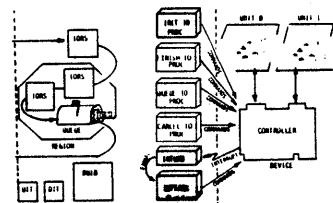
- IORS\$T IS THE TOKEN FOR THE I/O REQUEST SEGMENT
- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK OF THE DEVICE
- D\$DATA\$T IS A TOKEN FOR A DATA STORAGE AREA (CREATED BY THE INIT I/O PROCEDURE), CONTAINS (HEAD OF QUEUE, REGION TOKEN, . . . INTERRUPT TASK TOKEN, . . . ETC)

6-7

EXAMPLE - QUEUE I/O PROCEDURE



THIS IS WHERE THE FIRST REQUEST GETS STARTED



6-8

THE INIT I/O PROCEDURE

- THE BASIC I/O SYSTEM CALLS THE INIT I/O PROCEDURE IN THE FOLLOWING MANNER:

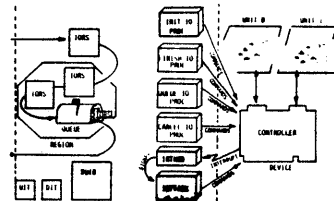
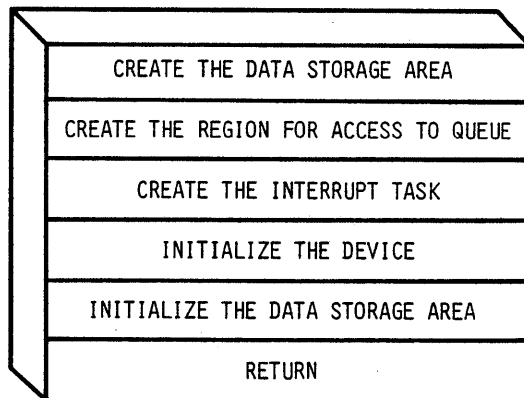
CALL INIT\$I/O(DUIB\$P,D\$DATA\$P,STATUS\$P);

WHERE:

- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK OF THE DEVICE
- D\$DATA\$P IS A POINTER TO A WORD WHERE THE INIT I/O PROCEDURE PLACES THE TOKEN FOR A DATA STORAGE AREA
- STATUS\$P IS A POINTER TO A WORD WHERE THE INIT I/O PROCEDURE PLACES A STATUS OF THE INITIALIZE OPERATION (SUCCESSFUL OR NOT SUCCESSFUL)

6-9

EXAMPLE INIT I/O PROCEDURE



6-10

THE FINISH I/O PROCEDURE

- THE BASIC I/O CALLS THE FINISH I/O PROCEDURE IN THE FOLLOWING MANNER:

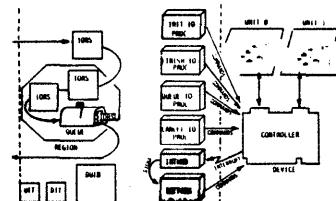
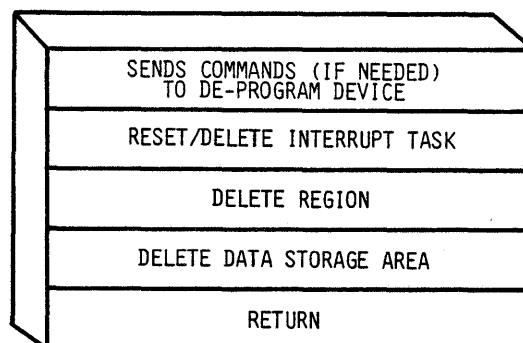
CALL FINISH\$I/O(DUIB\$P,D\$DATA\$T);

WHERE:

- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK OF THE DEVICE
- D\$DATA\$T IS A TOKEN FOR THE DATA STORAGE AREA

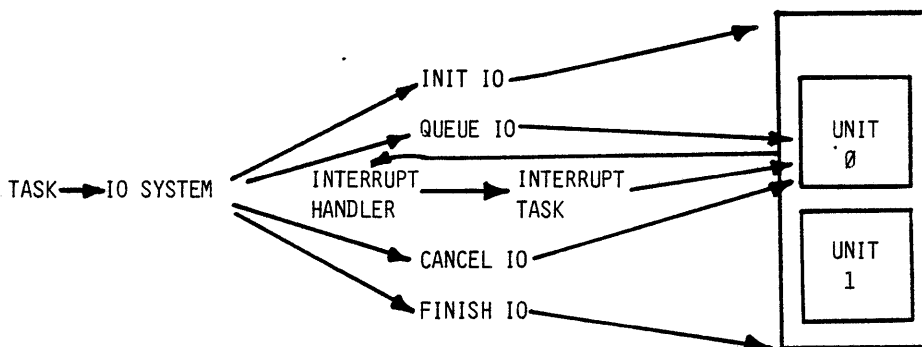
6-11

EXAMPLE FINISH I/O PROCEDURE



6-12

RELATIONSHIP BETWEEN CALLS



6-15

EXERCISE

- WRITE THE QUEUE\$I/O PROCEDURE HEADING AND DECLARATIONS IN PL/M 86.
- WHAT IS A DATA STORAGE AREA?
- WHEN IS THE INIT\$I/O PROCEDURE CALLED?
- WHEN IS CANCEL\$I/O CALLED?
- HOW DOES THE INTERRUPT HANDLER KNOW WHERE THE DATA STORAGE AREA IS LOCATED?

6-16

CHAPTER 7

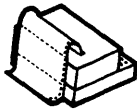
DEVICE DRIVERS

(Random Access and Common Device Drivers)

- COMPONENTS
- THE INTERRUPT PROCEDURE
- THE START PROCEDURE
- DEVICE INFORMATION TABLE

THE COMMON DEVICE DRIVER

- SIMPLE DEVICES
E.G. LINE PRINTERS, USART
- FIFO MECHANISM FOR QUEUING REQUESTS IS SUFFICIENT
- ONLY ONE INTERRUPT LEVEL IS NEEDED TO SERVICE A DEVICE
- DATA READ OR WRITTEN BY THESE DEVICES DOES NOT NEED TO BE BROKEN UP INTO BLOCKS.



7-1

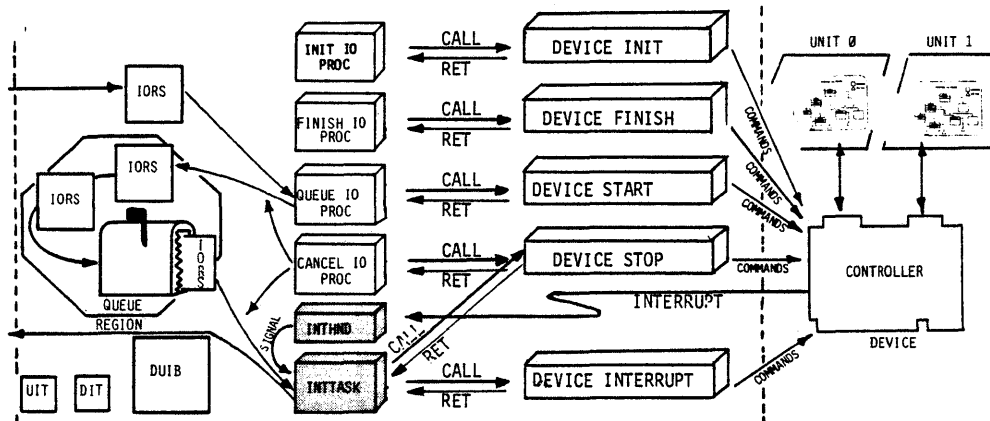
THE RANDOM ACCESS DEVICE DRIVER

- COMPLEX DEVICES
E.G. HARD DISK, BUBBLE MEMORY, FLOPPY
- FIFO MECHANISM FOR QUEUING REQUESTS IS SUFFICIENT
- ONLY ONE INTERRUPT LEVEL IS NEEDED TO SERVICE A DEVICE
- I/O REQUESTS MUST BE BROKEN UP INTO BLOCKS OF SPECIFIC LENGTH
- THE DEVICE SUPPORTS RANDOM ACCESS SEEK OPERATIONS



7-2

RANDOM/Common DEVICE DRIVER COMPONENTS



7-3

I/O SYSTEM SUPPLIED PROCS

- THE BASIC IO SYSTEM PROVIDES PROCEDURES THAT COMPRISE THE BULK OF WRITING A RANDOM ACCESS OR COMMON DEVICE DRIVER.

- PROCEDURE NAMES -



- "RAD\$INIT\$IO" OR "INIT\$IO"
- "RAD\$FINISH\$IO OR "FINISH\$IO"
- "RAD\$QUEUE\$IO" OR "QUEUE\$IO"
- "RAD\$CANCEL\$IO" OR "CANCEL\$IO"

7-4

THE DEVICE\$INTERRUPT PROCEDURE

- THE INTERRUPT TASK, SUPPLIED BY THE IO SYSTEM, CALLS THE INTERRUPT PROCEDURE IN THE FOLLOWING MANNER:

CALL DEVICE\$INTERRUPT(IORS\$P, DUIB\$P, DDATA\$P);

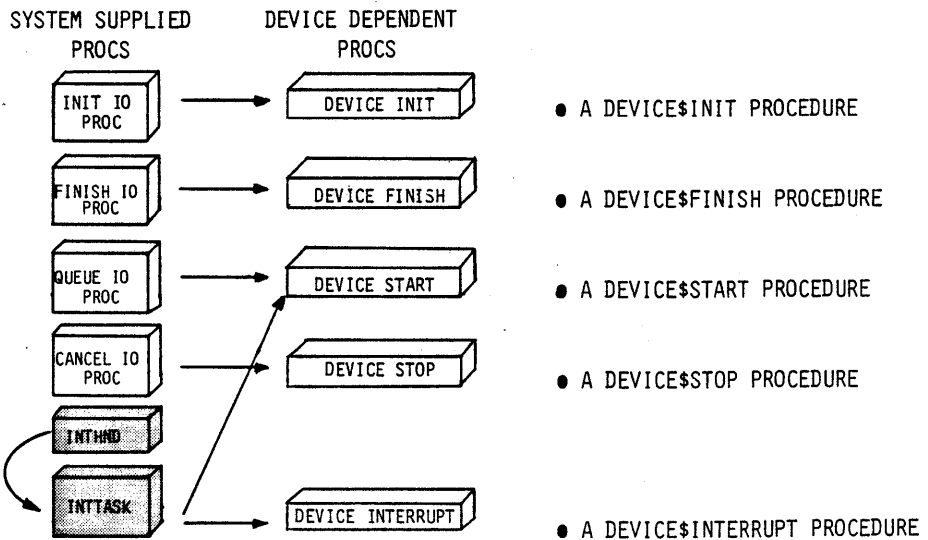
WHERE:

- IORS\$P IS A POINTER TO THE I/O REQUEST SEGMENT
- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK
- DDATA\$P IS A POINTER TO A USER REQUESTED RAM SEGMENT (SCRATCH PAD AREA)

7-5

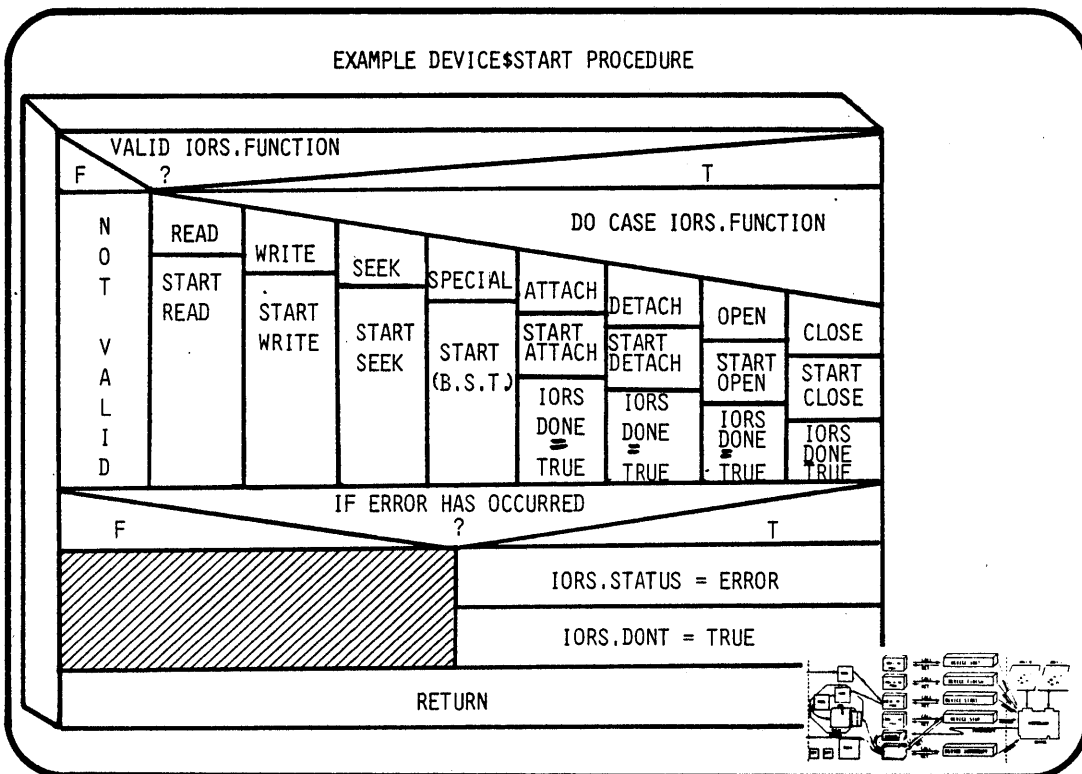
DEVICE DEPENDENT PROCS

- TO WRITE A RANDOM ACCESS OR COMMON DEVICE DRIVER YOU MUST PROVIDE:



7-6

EXAMPLE DEVICE\$START PROCEDURE



THE DEVICE\$INIT PROCEDURE

- THE INIT\$IO PROCEDURE, SUPPLIED BY THE IO SYSTEM, CALLS THE DEVICE\$INIT PROCEDURE IN THE FOLLOWING MANNER:

CALL DEVICE\$INIT(DUIB\$P,DDATA\$P,STATUS\$P);

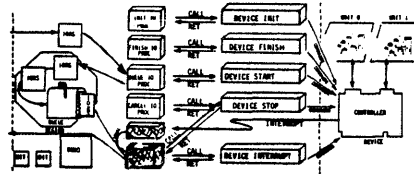
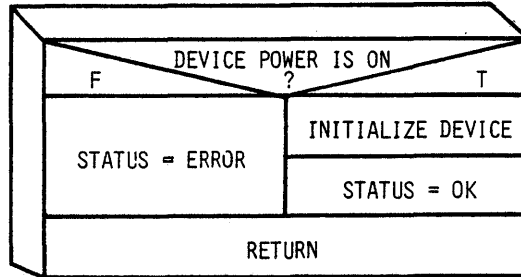
WHERE:

-DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK

-DDATA\$P IS A POINTER TO A USER REQUESTED RAM SEGMENT

-STATUS\$P IS A POINTER TO A WORD WHERE THE RESULT OF THE CALL WILL BE STORED BY THE USER

EXAMPLE DEVICE&INIT PROCEDURE



7-11

THE DEVICE\$FINISH PROCEDURE

- THE FINISH \$IO PROCEDURE, SUPPLIED BY THE IO SYSTEM, CALLS THE DEVICE\$FINISH PROCEDURE IN THE FOLLOWING MANNER:

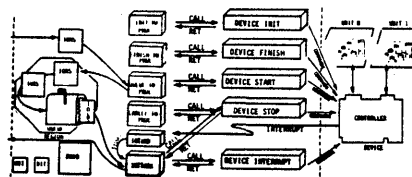
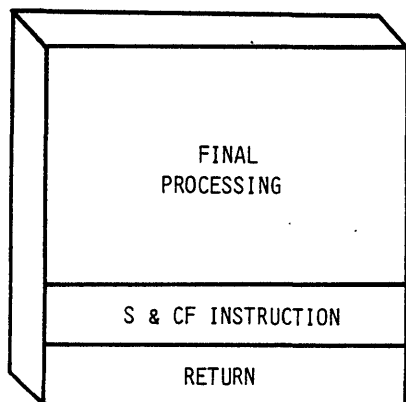
CALL DEVICE\$FINISH(DUIB\$P, DDATA\$P);

WHERE:

- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK
- DDATA\$P IS A POINTER TO A USER REQUESTED RAM SEGMENT

7-12

EXAMPLE DEVICE\$FINISH PROCEDURE



7-13

THE DEVICE\$STOP PROCEDURE

- THE CANCEL\$IO PROCEDURE, SUPPLIED BY THE IO SYSTEM, CALLS THE DEVICE\$STOP PROCEDURE IN THE FOLLOWING MANNER:

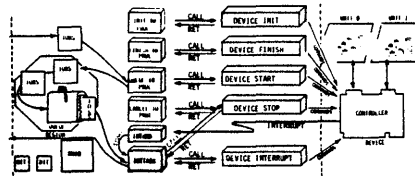
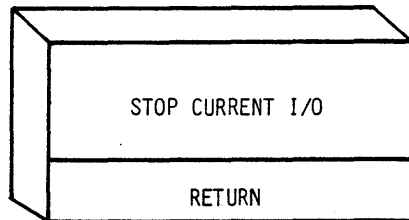
```
CALL DEVICE$STOP(IORS$P, DUIB$P, DDATA$P);
```

WHERE:

- IORS\$P IS A POINTER TO THE I/O REQUEST SEGMENT
- DUIB\$P IS A POINTER TO THE DEVICE UNIT INFORMATION BLOCK
- DDATA\$P IS A POINTER TO A USER REQUESTED RAM SEGMENT

7-14

EXAMPLE DEVICE\$STOP PROCEDURE



7-15

THE DEFAULT PROCEDURES

- THE I/O SYSTEM PROVIDES THREE NULL PROCEDURES THAT THE USER MAY USE.

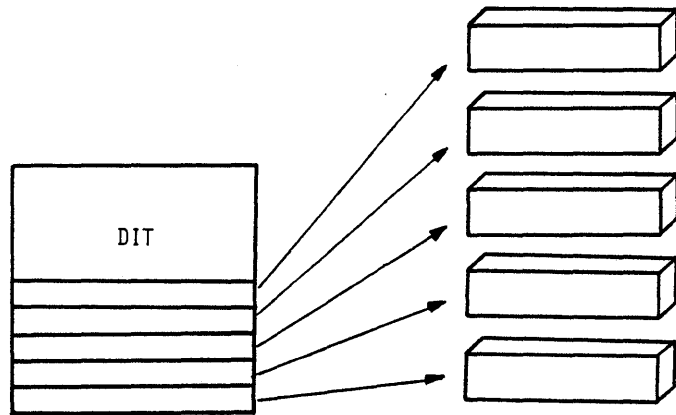
THEY ARE:

- "DEFAULT\$INIT"
- "DEFAULT\$FINISH"
- "DEFAULT\$STOP"

7-16

DIT'S

- COMMON AND RANDOM ACCESS DEVICE INFORMATION TABLES ARE CREATED BY THE SYSTEM'S PROGRAMMER DURING CONFIGURATION OF THE BASIC I/O SYSTEM.
- THE DIT BINDS THE I/O SYSTEM SUPPLIED PROCEDURE TO THE DEVICE DEPENDENT PROCEDURES BY DESCRIBING POINTERS TO THEM.



7-17

THE DIT STRUCTURE (DEVICE INFORMATION TABLE)

LEVEL	
PRIORITY	
STACK\$SIZE	
DATA\$SIZE	
NUM\$UNITS	
DEVICE\$INIT	
DEVICE\$FINISH	
DEVICE\$START	
DEVICE\$STOP	
DEVICE\$INTERRUPT	
~~~~~	

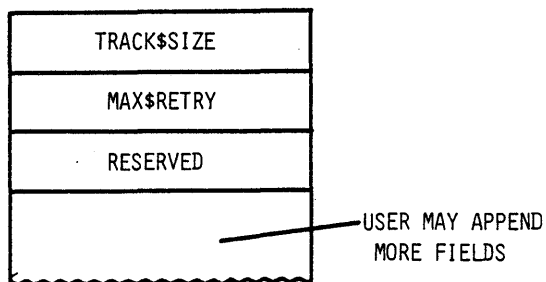
USER MAY  
APPEND MORE FIELDS

- REFER TO "GUIDE TO WRITING DEVICE DRIVERS" REFERENCE MANUAL.

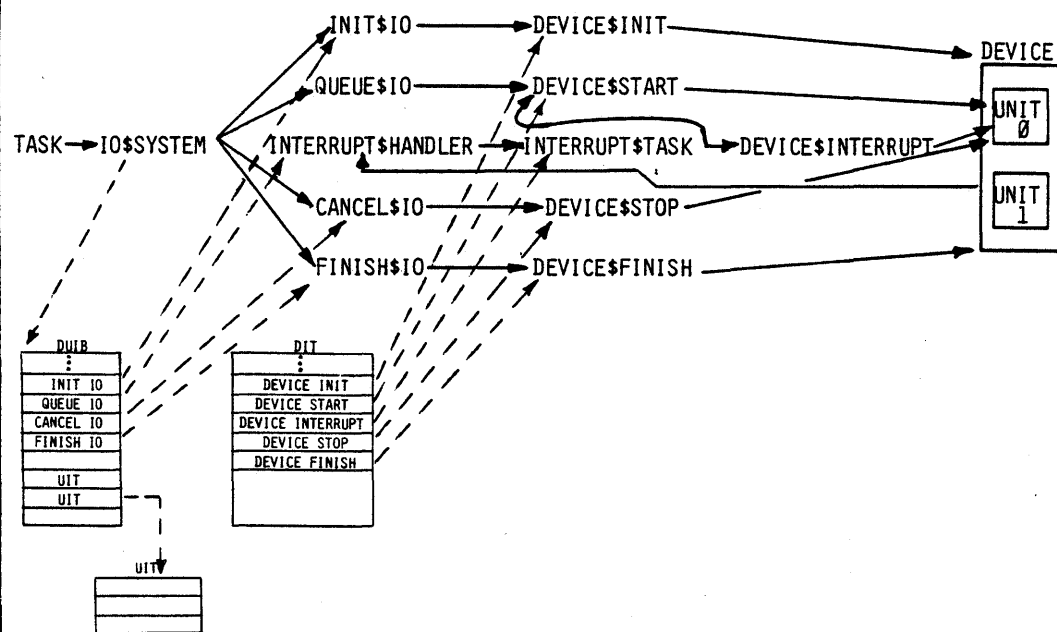
7-18

### UIT'S

- UNIT INFORMATION TABLES ARE CREATED BY THE SYSTEM PROGRAMMER DURING CONFIGURATION OF THE BASIC IO SYSTEM.



### TABLES AND CALLS



OBJECTIVES:

EXECUTE A GIVEN APPLICATION JOB, THAT WILL CALL UPON THE BIOS TO COMMUNICATE WITH A LIGHT BOX

THE STUDENT WILL WRITE A COMMON DEVICE DRIVER TO SUPPORT THE LIGHT BOX HARDWARE

CREATE SOURCE CODE:

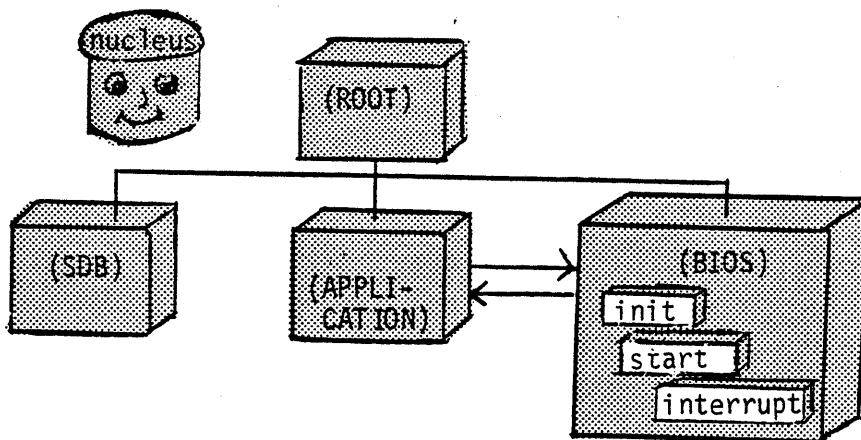
- A SOURCE FILE NAMED DEVDRV.P86
- A SOURCE FILE NAMED LBOXDUIB.SRC
- A SOURCE FILE NAMED LBOXDUIT.SRC

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB3) AS THE LOGICAL NAME (:LAB:)

- AFILE "/TEAM NAME"/LAB3 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



- Write a common device driver for the light box
- The student will be given the nucleus, SDB, application and root
- The student will supply init, start and interrupt procedures for the driver
- The student will build a BIOS with preconfigured ITABLE.A86 and IDEVCF.A86

STEP2:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:LBOXDUIB.SRC WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:LBOXDUIB.SRC

* THIS SOURCE FILE CONTAIN PARTIALLY WRITTEN SOURCE CODE TO THE DUIB TABLES NEEDED TO SUPPORT THE LIGHT BOX DRIVER

STEP3:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:LBOXDUII.SRC WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:LBOXDUII.SRC

* THIS SOURCE FILE CONTAIN PARTIALLY WRITTEN SOURCE CODE TO THE DIT TABLES NEEDED TO SUPPORT THE LIGHT BOX DRIVER

STEP4:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:DEVDRV.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:DEVDRV.P86

* THIS SOURCE FILE CONTAIN PARTIALLY WRITTEN SOURCE CODE TO THE FOLLOWING PROCEDURES:

- A LIGHT BOX DEVICE INTERRUPT PROCEDURE
- A LIGHT BOX DEVICE START PROCEDURE
- A LIGHT BOX DEVICE INIT PROCEDURE



***** LAB THREE (COMMON DEVICE DRIVER) *****

STEP5:

COMPILE THE SOURCE FILE DEVDRV.P86

- PLM86 :LAB:DEVDRV.P86

- * IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING
- * IF COMPILATION IS SUCCESSFUL THE COMPILER WILL CREATE FOR THE SOURCE FILE:
  - A LIST FILE NAMED ":LAB:(SOURCE).LST"
  - AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

STEP6:

- * WE MUST NOW ADD THE OBJECT CODE THAT WE HAVE GENERATED TO THE BIOS SYSTEM
- * THEN WE MUST BUILD A LOADABLE SYSTEM THAT INCLUDES:
  - THE NUCLEUS
  - THE BIOS ( YOU SUPPLY THE LIGHT BOX DRIVER )
  - THE SDB
  - THE APPLICATION JOB (SUPPLIED) THAT CALLS YOUR DRIVER
  - THE ROOT JOB
- * THIS IS ACCOMPLISHED THROUGH SEVERAL LINKS, LOCATES AND FINALLY USING THE LIB86 UTILITY
- * A SUBMIT FILE IS SUPPLIED
  - SUBMIT :LAB:ICU.CSD

STEP7:

- * A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT REMEMBER THE MAP FILE IS :LAB:IOS.MP2

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL  
-BOOT THE NEW SYSTEM  
.B /"TEAM NAME"/LAB2/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )  
  
-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL  
-BOOT THE NEW SYSTEM  
.B /RMX86

- * THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NECESSARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL



## **CHAPTER 8**

### **BASIC I/O SYSTEM**

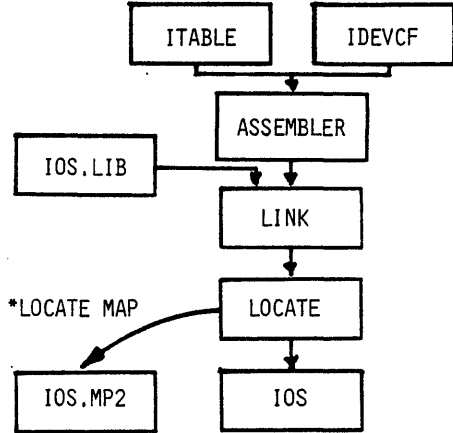
#### **Configuration**

- TABLES
- ICU86



STEPS IN BUILDING THE BASIC I/O SYSTEM

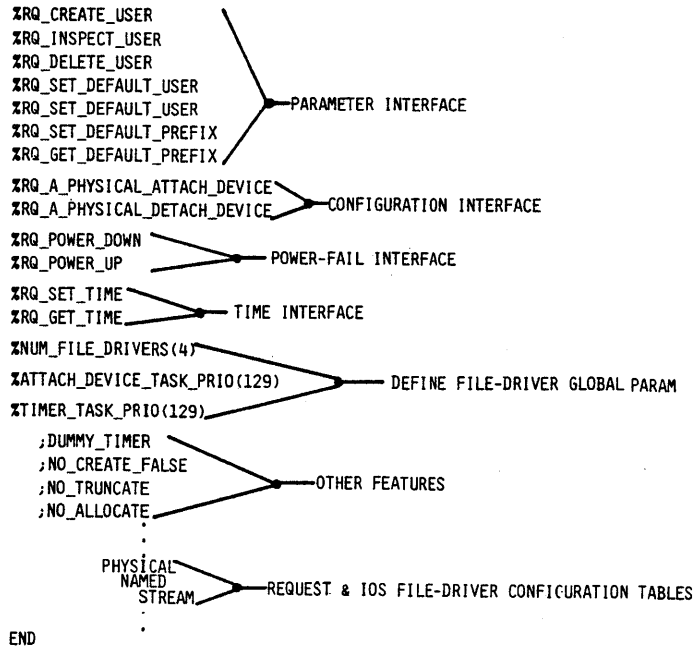
- 1) TWO CONFIGURATION FILES ARE NEEDED (ITABLE.A86 AND IDEVCF.A86)
- 2) ASSEMBLE EACH CONFIGURATION FILE
- 3) LINK AND LOCATE THE BIOS



} ALTER I . . . .A86

- ITABLE CONTAINS SYSTEM CALLS AND FILE DRIVERS AND TABLES
- IDEVCF CONTAINS DEVICE DRIVERS INTO TABLES (DUIBS, DITS, UITS)
- REFER TO CONFIGURATION MANUAL THROUGH THE REST OF THIS CHAPTER

ITABLE.A86



IDEVCF.A86

```

      .
      .
EXTRN  INITIO: NEAR
EXTRN  FINISHIO: NEAR
EXTRN  QUEUEIO: NEAR

EXTRN  1204INIT: NEAR
EXTRN  1204START: NEAR
EXTRN  1204INTERRUPT: NEAR
      .
      .

```

EXTERNAL PROCEDURES  
DEFINITIONS

IDEVCF.A86 (CONTINUED)

```

; SPC 204 CONTROLLER UNIT 0
  DEFINE_DUIB
& 'F0',      ; NAME (14)
& 00BH,     ; FILEDRIVERS
& 0FFH,     ; FUNCTS
& 00H,      ; FLAGS
& 128,      ; DEV GRANULARITY
& 0E900H,03H, ; DEV SIZE = 256256
& 0,        ; DEVICE #
& 0,        ; UNIT #
& 0,        ; DEVICE-UNIT #
& INITIO,   ; DRIVER PROCS
& FINISHIO , ;
& QUEUEIO,  ;
& CANCELIO, ;
& DINFO_204, ; DIT
& VINFO_SHUGART, ; UIT
& 100,      ; UPDATE TIMEOUT
& 6,        ; NUMBER OF BUFFERS
& 129      ; INITIAL TASK PRIORITY
& >

```

DUIB  
DEFINITIONS

IDEVCF.A86 (CONTINUED)

```
DINFO_204    RADEV_DEV_INFO<
& 58H,      ; LEVEL
& 81H,      ; PRIORITY
& 209,      ; STACK SIZE
& 127,      ; DATA SIZE
& 4,        ; NUM UNITS
& I204INIT, ; DRIVER PROCS
& DEFAULTFINISH, ;
& I204START, ;
& DEFAULTSTOP, ;
& I204INTERRUPT ;
& >
```

DEVICE INFO TABLE

DW 0A0H ; BASE ADDRESS

EXTRA INFO MAY BE APPENDED HERE

IDEVCF.A86 (CONTINUED)

```
UINFO_SHUGART  RADDEV_UNIT_INFO<
& 26 * 128,   ; TRACK SIZE
& 9,          ; MAX RETRY
& 0
& >
```

UNIT INFO TABLE

; 204 SPECIFIC

DW 4 ; RESERVED

DB 035H,0DH ; FIXED INIT VALUES

DB 8 ; STEP RATE

DB 8 ; SETTLE

DB 39H ; ENT LOAD

EXTRA INFO IS APPENDED HERE

THE SUBMIT FILE (86/330 STYLE)

```

IOS.CSD
;ASSEMBLE THE TABLES
ASM86 /RMX/DBIOS/ITABLEA86 WORKFILES(:WORK:,:WORK:)
ASM86 /RMX/DBIOS/IDEVCF.A86 WORKFILES(:WORK:,:WORK:)
;LINK THE BIOS
LINK 86 /RMX/DBIOS/IOS.LIB(ISTART),&
        /RMX/DBIOS/ITABLE.OBJ      ,&
        /RMX/DBIOS/IDEVCF.OBJ      ,&
        /RMX/DBIOS/ITDR.LIB        ,&
        /RMX/DHI/HI.LIB(HCONTC)    ,&
        /RMX/DBIOS/100PT1.LIB      ,&
        /RMX/DBIOS/IOS.LIB         ,&
        /RMX/DNUCLUS/RPIFC.LIB     ,&
TO      /RMX/DBIOS/IOS.LNK          &
        NO PUBLICS EXCEPT(RQAIOSINITASK,RQAIOSINITERROR)
;LOCATE THE BIOS
LOCATE  /RMXDBIOS/IOS.LNK           &
TO      /RMX/DBIOS/IOS.LNK         &
        SEGSIZE(STACK(0))           &
        ORDER(CLASSES(CODE,DATA))  &
        ADDRESSES(CLASSES(CODE(%1))) &
NOINITCODE

```

FOUND IN MISCELLANEOUS DISKETTE

COMPACT!

ADDRESS OF BIOS IS PASSED AS A PARAMETER

8-7

LOCATE ADDRESSES

- THE LOCATE PROGRAM GENERATES A MAP FILE CALLED /RMX/DBIOS/IOS.M12
- EXAMINING THE MAP WE OBTAIN THE ENDING ADDRESS OF THE BIOS MEMORY MAP OF MODULE ISTART

SEGMENT MAP

START	START	LENGTH	ALIGN	NAME	CLASS
00200H	00216H	0017H	A	(ABSOLUTE)	
07190H	14E85H	DCF6H	W	CODE	CODE
14E86H	14E93E	000EF	W	PARAM_SEG	CODE
14E94H	14E97H	0004H	W	CONFIG_SEG	CODE
14E98H	14B9BH	0004H	W	POWER_SEG	CODE
14E9CH	14E9FH	0004H	W	TIME_SEG	CODE
14EA0H	14EBBH	001CH	W	FILE_DRIVER_IN	CODE
				-FO_SEG	
14EBCH	14F7BH	00C0H	W	REQ_TABLE	CODE
14F7CH	1504BH	00D0H	W	IOS_TABLE	CODE
1504CH	1509DH	0052H	W	DATA	DATA
150A0H	150AFH	0010H	G	??SEG	
150B0H	150B0H	0000H	W	STACK	STACK
150B0H	150B0H	0000H	W	MEMORY	MEMORY

8-8



REMEMBER!

- ADD A %JOB MACRO TO YOUR ROOT JOB

```
; IOS JOB
;
%JOB(0,          %' OBJECT DIRECTORY SIZE
0600H, 0FFFFJ,  %' POOL SIZE (MIN, MAX)
0FFFFH, 0FFFFH, %' MAX OBJECTS AND TASKS
0,           %' MAX JOB PRIORITY
0:0, 0       %' EXCEPTION HANDLER ADDR, MODE
0,           %' JOB FLAGS
130,         %' INIT TASK PRIORITY
719:0       %' INIT TASK ENTRY ADDRESS
0,           %' INIT TASK DATA SEGMENT ADDRESS
0:0, 200H   %' INIT TASK STACK ADDRESS, STACK SIZE
0)          %' INIT TASK FLAGS
```

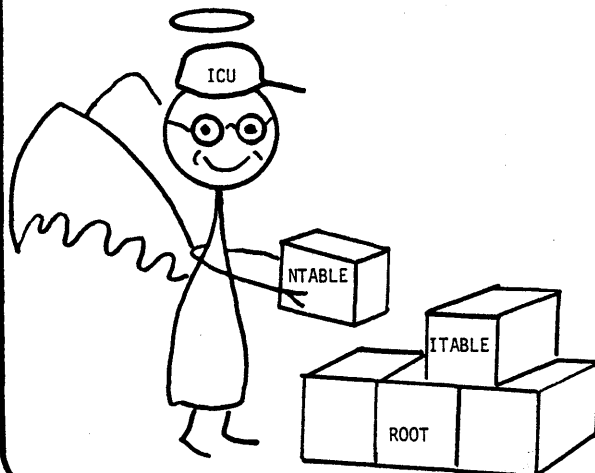


THAT'S ALL FOLKS!

8-9

THE GOOD NEWS!

- ICU WILL
  - CREATE ITABLE AND IDEVCF
  - CREATE A SUBMIT FILE TO ASSEMBLE LINK AND LOCATE THE BIOS
  - ADD A JOB MACRO TO THE ROOT JOB
  - COMPUTE THE STARTING ADDRESS FOR THE BIOS



8-10



OBJECTIVES:

EXECUTE A GIVEN APPLICATION JOB, THAT WILL CALL UPON THE BIOS TO COMMUNICATE WITH A LIGHT BOX

THE STUDENT WILL BUILD UPON LAB THREE TO CONFIGURE THE ALL THE PARTS NESSESARY TO EXECUTE THAT LAB

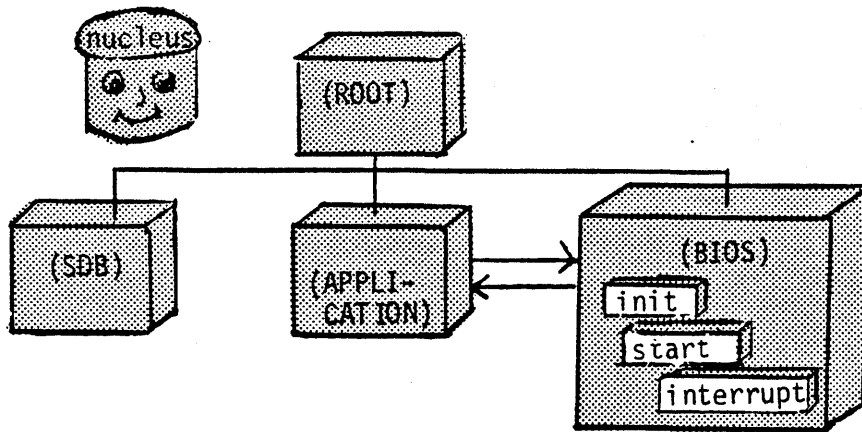
THE STUDENT WILL USE THE INTERACTIVE CONFIGURATION UTILITY (ICU)

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB3) AS THE LOGICAL NAME (:LAB:)

- AFILE "/TEAM NAME"/LAB3 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



***** LAB FOUR (BIOS CONFIG THROUGH ICU) *****

STEP2:

THE (ICU) IS INVOKED BY TYPING THE FOLLOWING  
-ICU86 :LAB:ICU.DEF

* WHERE ICU.DEF IS THE NAME OF THE FILE WE HAVE CHOSEN TO CONTAIN  
THE INFORMATION NEEDED TO CONFIGURE OUR O.S.

WHEN THE ICU SIGN ON ENTER THE COMMAND C , TO MODIFY THE SCREENS

STEP3:

* TRY FILLING THE SCREENS WITHOUT LOOKING AT THESE FIRST,  
THEN MATCH YOUR ENTRIES TO THE ONES GIVEN HERE

* IF YOU DO NOT UNDERSTAND AN ENTRY TYPE ?  
- E.G. OSP? <cr>

* THE SCREEN FOR OUR LAB THREE CONFIGURATION FOLLOW

Hardware

(OSP)	80130 Operating System Extension (Yes/No)	No
(OTU)	80130 Timer Used (Yes/No)	No
(OPU)	80130 PIC Used (Yes/No)	No
(OCD)	80130 Copyright = 1981 (Yes/No)	Yes
(BL)	80130 Base Address Location (40h-0FFFFh)	0000H
(BP)	80130 Base Port Address (0-0FFFFH)	0000H
(MP)	8259A Master Port (0-0FFFFH)	00C0H
(MPS)	Master PIC Port Separation (0-0FFH)	0002H
(SIL)	Slave Interrupt Levels (1-7/None)	None
(LSS)	Level Sensitive Slaves (1-7/None)	None
(LSP)	Local Slave PICS (1-7/None)	None
(TP)	8253 Timer Port (0-0FFFFH)	00D0H
(CIL)	Clock Interrupt Level (0-7)	0002H
(CN)	Timer Counter Number (0,1,2)	0000H
(CI)	Clock Interval (0-0FFFFH msec)	000AH
(CF)	Clock Frequency (0-0FFFFH khz)	04CDH
(TPS)	Timer Port Separation (0-0FFH)	0002H
(NPX)	Numeric Processor Extension (Yes/No)	Yes
(NIL)	NPX Interrupt Level (Encoded)	0008H

Memory

Type : RAM = low, high  
Type : ROM = low, high  
Type : RAM = 0500H, F7FFH

***** LAB FOUR (BIOS CONFIG THROUGH ICU) *****

Sub-systems

(UDI) Universal Development Interface (Yes/No)	No
(HI) Human Interface (Yes/No)	No
(AL) Application Loader (Yes/No)	No
(EIO) Extended I/O System (Yes/No)	No
(BIO) Basic I/O System (Yes/No)	Yes
(DB) Debugger (Yes/No)	No
(TH) Terminal Handler (Yes/No)	No
(CA) Crash Analyzer (Yes/No)	No
(UIR) UDI in ROM (Yes/No)	No
(CAR) Crash Analyzer in ROM (Yes/No)	No
(RIR) Root Job in ROM (Yes/No)	No

BIOS

(ASC) All Sys Calls in BIOS (Yes/No)	Yes
(ADP) Attach Device Task Priority (1-OFFH)	0081H
(TF) Timing Facilities Required (Yes/No)	Yes
(TTP) Timer Task Priority (0-OFFH)	0081H
(CON) Connection Job Delete Priority (0-OFFH)	0082H
(ACE) Ability to Create Existing Files (Yes/No)	Yes
(SMI) System Manager ID (Yes/No)	Yes
(CUT) Common Update Timeout (0-OFFFFH)	03E8H
(CST) Control-Sequence Translation (Yes/No)	Yes
(PMI) BIOS Pool Minimum (0-OFFFFH)	0C00H
(PMA) BIOS Pool Maximum (0-OFFFFH)	0C00H
(BIR) Basic I/O System in ROM (Yes/No)	No

User Devices

(OPN) Object Code Path Name (1-45 characters)	
	:LAB:DEVDRV.OBJ
(DPN) Duib Source Code Path Name (1-45 characters)	
	:LAB:LBOXDUIB.SRC
(DUP) Device and Unit Source Code Path Name (1-45 characters)	
	:LAB:LBOXDUNIT.SRC
(ND) Number of User Defined Devices (0-OFFH)	0001H
(NDU) Number of User Defined Device-Units (0-OFFH)	0002H

***** LAB FOUR (BIOS CONFIG THROUGH ICU) *****

Nucleus

(ASC) All Sys Calls (Yes/No)	Yes
(PV) Parameter Validation (Yes/No)	Yes
(ROD) Root Object Directory Size (0 - 0FF0h)	0028H
(MTS) Minimum Transfer Size (0-0FFFFH)	0040H
(DEH) Default Exception Handler (Yes/No/Deb/Use)	Yes
(NEH) Name of Ex Handler Object Module (1-32chs)	
(EM) Exception Mode (Never/Program/Environ/All)	Never
(NR) Nucleus in ROM (Yes/No)	No

***** THIS JOB IS SUPPLIED TO CALL THE (BIOS) *****

User Jobs

(ODS) Object Directory Size (0-0FF0H)	0032H
(PMI) Pool Minimum (20H - 0FFFFH)	0100H
(PMA) Pool Maximum (20H - 0FFFFH)	FFFFH
(MOB) Maximum Objects (1 - 0FFFFH)	FFFFH
(MTK) Maximum Tasks (1 - 0FFFFH)	FFFFH
(MPR) Maximum Priority (0 - 0FFH)	0000H
(AEH) Address of Exception Handler (CS:IP)	0000H:0000H
(EM) Exception Mode (Never/Prog/Environ/All)	Never
(PV) Parameter Validation (Yes/No)	Yes
(TP) Task Priority (0-0FFH)	009BH
(TSA) Task Start Address (CS:IP)	0104H:0002H
(DSB) Data Segment Base (0-0FFFFH)	0000H
(SSA) Stack Segment Address (SS:SP)	0000H:0000H
(SS) Stack Size (0-0FFFFH)	0200H
(NPX) Numeric Processor Extension Used (Yes/No)	No

***** THIS JOB SUPPLIES THE INTERRUPT FOR THE MONITOR *****

User Jobs

(ODS) Object Directory Size (0-0FF0H)	000AH
(PMI) Pool Minimum (20H - 0FFFFH)	0030H
(PMA) Pool Maximum (20H - 0FFFFH)	FFFFH
(MOB) Maximum Objects (1 - 0FFFFH)	FFFFH
(MTK) Maximum Tasks (1 - 0FFFFH)	FFFFH
(MPR) Maximum Priority (0 - 0FFH)	0000H
(AEH) Address of Exception Handler (CS:IP)	0000H:0000H
(EM) Exception Mode (Never/Prog/Environ/All)	Never
(PV) Parameter Validation (Yes/No)	Yes
(TP) Task Priority (0-0FFH)	0000H
(TSA) Task Start Address (CS:IP)	0080H:0002H
(DSB) Data Segment Base (0-0FFFFH)	0000H
(SSA) Stack Segment Address (SS:SP)	0000H:0000H
(SS) Stack Size (0-0FFFFH)	0200H
(NPX) Numeric Processor Extension Used (Yes/No)	No

***** LAB FOUR (BIOS CONFIG THROUGH ICU) *****

Includes and Libraries

Path Name (1-45 Characters)

(UDF) UDI Includes and Libs

/RMX5.0/DUDI/

(HIF) Human Interface Includes and Libs

/RMX5.0/DINCLSLIBS/

(EIF) Extended I/O System Includes and Libs

/RMX5.0/DINCLSLIBS/

(ALF) Application Loader Includes and Libs

/RMX5.0/DLOADER/

(BIF) Basic I/O System Includes and Libs

/RMX5.0/DINCLSLIBS/

(THF) Terminal Handler and Debugger Includes and Libs

/RMX5.0/DDEBTH/

(NUF) Nucleus and Root Job Includes and Libs

/RMX5.0/DNUCLUS/

(ILF) Interface Libraries

/RMX5.0/DUTILS/

(CAF) Crash Analyzer Includes and Libs

/RMX5.0/DUDI/

(DTF) Development Tools Path Names

/LANG/

Generate File Names

File Name (1-55 Characters)

(ROF) ROM Code File Name

:LAB:NONE

(RAF) RAM Code File Name

:LAB:RMX86

STEP4:

AFTER YOU ENTER ALL OF THE SCREENS ENTER G TO GENERATE

EXIT THE ICU

SUBMIT THE ICU.CSD FILE TO GENERATE YOUR SYSTEM

-SUBMIT :LAB:ICU.CSD

***** LAB FOUR (BIOS CONFIG THROUGH ICU) *****

STEP5:

YOU MUST NOW ADD THE USER JOB , AND THE SDB TO THE SYSTEM,  
USING THE LIB86 UTILITY

```
-LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
DELETE :LAB:RMX86(INT3TASKMOD)
ADD /DINT3/INT3JOB to :LAB:RMX86
EXIT
```

STEP6:

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION  
THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB3/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION  
THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG  
YOUR CODE IF NECESSARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL



**CHAPTER 9**

**EXTENDED I/O SYSTEM**

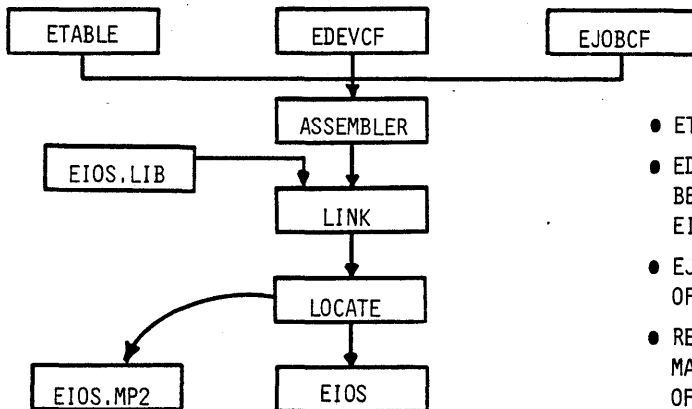
**Configuration**

- TABLES
- ICU86



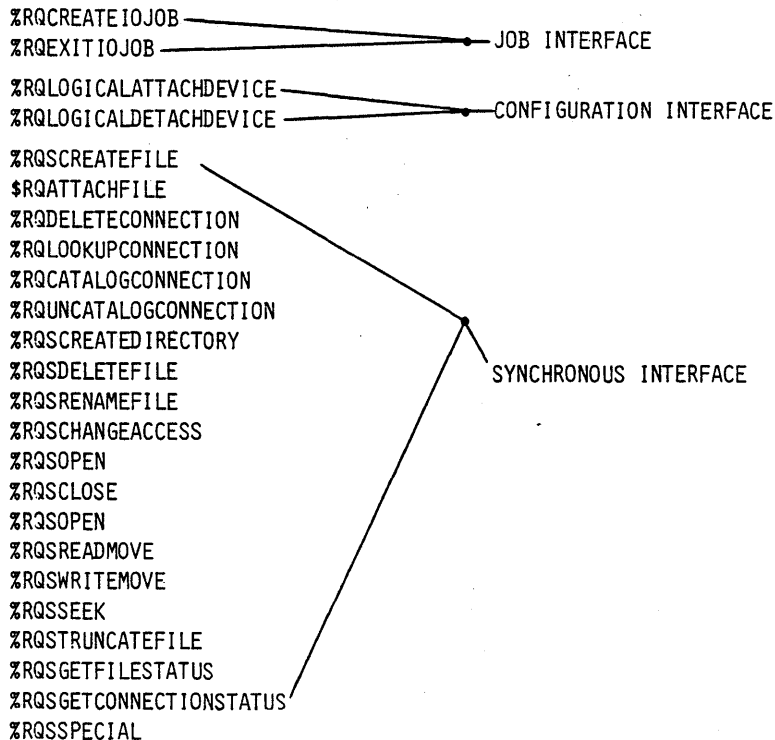
STEPS IN BUILDING THE EXTENDED I/O SYSTEM

- 1) THREE CONFIGURATION FILES ARE NEEDED (ETABLE.A86,EDEVCF.A86, EJOBBCF.A86)
- 2) ASSEMBLE EACH CONFIGURATION TABLE
- 3) LINK AND LOCATE THE EIOS



- ETABLE CONTAINS SYSTEM CALLS
- EDEVCF CONTAINS DEVICES TO BE LOGICALLY ATTACHED BY EIOS
- EJOBBCF CONTAINS DESCRIPTION OF CHILD IO JOBS
- REFER TO CONFIGURATION MANUAL THROUGH THE REST OF THIS CHAPTER

ETABLE.A86



EDEVCF.A86

```
; BYTE-BUCKET
  %DEV_INFO_BLOCK('BB','BB',PHYSICAL)
; TERMINAL
  %DEV_INFO_BLOCK('TO','TO',PHYSICAL)
; 215 WINCHESTER - PR1AM, UNIT 0, DRIVE 0
  %DEV_INFO_BLOCK('WD0','IW0',NAMED) — LOGICALLY ATTACH
; 215 WINCHESTER FLOPPY DS/DD, UNIT 0, DRIVE 0   IW0 AS :WD0:
  %DEV-INFO-BLOCK('FD0','WFDD0',NAMED)
; STREAM
  %DEV-INFO-BLOCK('STREAM','STREAM',STREAM)
; LP
  %DEV-INFO-BLOCK('STREAM','STREAM',STREAM)
  %END_DEV_CONFIG(1024)
```

EJOB CF.A86

USER 'WORLD' DEFINITION

```
%IO_USER('WORLD', 0FFFFH) — OWNER ID
```

EIOS TEST JOB

```
%IO_JOB('TO', 'WORLD', 260H, 0FFFFH, 0:0, 3, 155, 2000:2, 0, 0:0, 1200, 0)
```

```
%END_IO_JOB_CONFIG(40)
```

—  
—  
YOU PROVIDE ADDRESS

### LOCATE ADDRESSES

- THE LOCATE PROGRAM GENERATES A MAP FILE CALLED /RMX/DEIOS/EIOS.MP2
- EXAMINING THE MAP WE OBTAIN THE ENDING ADDRESS OF THE EIOS (USED FOR LOCATING THE NEXT JOB)

#### MEMORY MAP OF MODULE EIOENT

##### SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
00200H	00216H	0017H	A	(ABSOLUTE)	
182B0H	1D0D7H	2E28H	W	CODE	CODE
1D0D8H	1D0DFH	0008H	W	JOB_SEG	CODE
1D0E0H	1D117H	0038H	W	SYNCHRONOUS_SE	CODE
				-G	
1D118H	1D11BH	0004H	W	CONFIGURATION_	CODE
				-SEG	
1D11CH	1D129H	000EH	W	GROUPUSER_SEG	CODE
1D12AH	1D133H	000AH	W	ALLOCATION_SEG	CODE
1D134H	1D143H	0010H	W	DATA	DATA
1D150H	1D150H	0000H	G	??SEG	
1D150H	1D150H	0000H	W	STACK	STACK
1D150H	1D150H	0000H	W	MEMORY	MEMORY

9-5

### THE SUBMIT FILE (86/330 STYLE)

```

ASM86 /RMX/DEIOS/ETABLE330.A86 PRINT(/RMX/DEIOS/ETABLE330.LST) &
      WORKFILES(:WORK:,:WORK:) OBJECT(/RMX/DEIOS/ETABLE.OBJ)
ASM86 /RMX/DEIOS/EDEVCF330.A86 PRINT(/RMX/DEIOS/EDEVCF330.LST) &
      WORKFILES(:WORK:,:WORK:) OBJECT(/RMX/DEIOS/EDEVCF.OBJ)
ASM86 /RMX/DEIOS/EJOBFCF330.A86 PRINT(/RMX/DEIOS/EJOBFCF330.LST) &
      WORKFILES(:WORK:,:WORK:) OBJECT(/RMX/DEIOS/EJOBFCF.OBJ)
;
; LINK AND LOCATE EIOS INITIALIZATION CODE AND SYSTEM CALLS
;
LINK86
      &
      /RMX/DEIOS/EIOS.LIB(EIOENT), &
      /RMX/DEIOS/ETABLE.OBJ, &
      /RMX/DEIOS/EDEVCF.OBJ, &
      /RMX/DEIOS/EJOBFCF.OBJ, &
      /RMX/DEIOS/EIOS.LIB, &
      /RMX/DEIOS/EPIFC.LIB, &
      /RMX/DBIOS/IPIFC.LIB, &
      /RMX/DNUCLUS/RPIFC.LIB &
      TO /RMX/DEIOS/EIOS.LNK &
      MAP PRINT(/RMX/DEIOS/EIOS.MP1)&
      NOPUBLICS EXCEPT(RQEIOSINITASK, RQEIOSINITERROR)
;
LOC86
      &
      /RMX/DEIOS/EIOS.LNK TO /RMX/DEIOS/EIOS &
      MAP PRINT(/RMX/DEIOS/EIOS.MP2) &
      NOLINES NOCOMMENTS NOSYMBOLS &
      SEGSIZE(STACK(0)) &
      ORDER(CLASSES(CODE, DATA)) &
      ADDRESSES(CLASSES(CODE(0x1)))

```

9-6

## ROOT JOB MACRO

- REMEMBER TO ADD A %JOB MACRO TO YOUR ROOT JOB

### EIOS JOB

```
%JOB(10,           %'OBJECT DIRECTORY SIZE
      0150H, 0FFFFH, %'POOL SIZE (MIN , MAX)
      0FFFFH, 0FFFFH, %'MAX OBJECTS AND TASKS
      130,          %'MAX JOB PRIORITY
      0:0, 3,       %'EXCEPTION HANDLER ADDR, MODE
      0,            %'JOB FLAGS
      150,          %'INIT TASK PRIORITY
      1A2B:0,       %'INIT TASK ENTRY ADDRESS
      0,            %'INIT DATA SEGMENT ADDRESS
      0:0, 250H,    %'INIT TASK STACK ADDRESS, STACK SIZE
      0)            %'INIT TASK FLAGS
```

9-7

## ICU 86

- ICU86 WILL
  - CREATE ETABLE, EDEVCF AND EJOB CF
  - CREATE A SUBMIT FILE THAT INCLUDES ASSEMBLY LINKING AND LOCATION OF THE EIOS
  - ADD A JOB MACRO TO THE ROOT JOB
  - COMPUTE THE STARTING ADDRESS OF THE EIOS

9-8

OBJECTIVES:

EXECUTE A GIVEN APPLICATION JOB, THAT WILL CALL UPON THE EIOS TO COMMUNICATE WITH A TERMINAL AND A FILE IN A FLOPPY

THE STUDENT WILL BUILD UPON LAB TWO TO CONFIGURE ALL THE PARTS NECESSARY TO EXECUTE THAT LAB

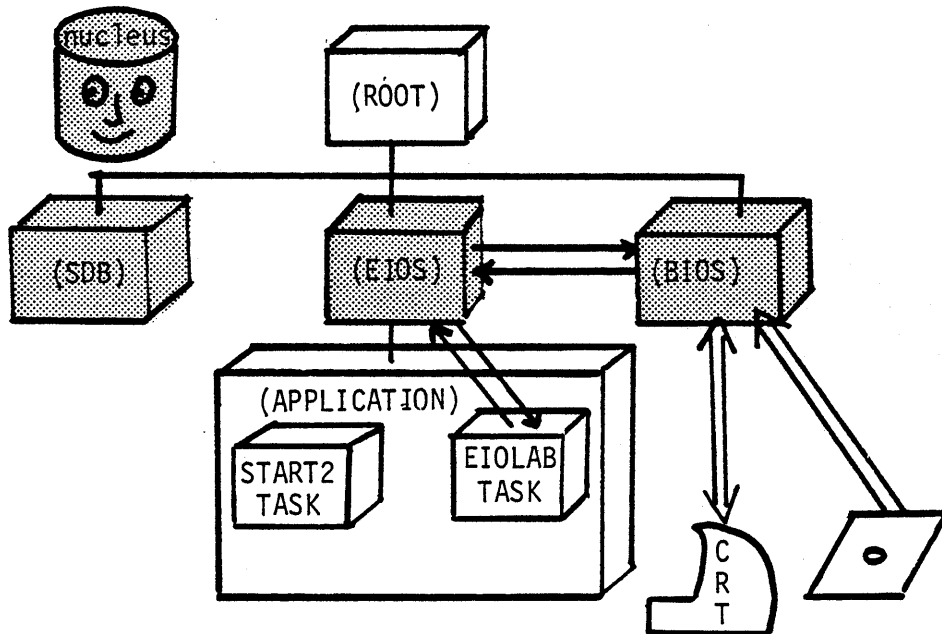
THE STUDENT WILL USE THE INTERACTIVE CONFIGURATION UTILITY (ICU)

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB2) AS THE LOGICAL NAME (:LAB:)

- AFILE "/TEAM NAME"/LAB2 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

STEP2:

THE (ICU) IS INVOKED BY TYPING THE FOLLOWING  
-ICU86 :LAB:ICU.DEF

* WHERE ICU.DEF IS THE NAME OF THE FILE WE HAVE CHOSEN TO CONTAIN  
THE INFORMATION NEEDED TO CONFIGURE OUR O.S.

WHEN THE ICU SIGN ON ENTER THE COMMAND C , TO MODIFY THE SCREENS

STEP3:

* TRY FILLING THE SCREENS WITHOUT LOOKING AT THESE FIRST,  
THEN MATCH YOUR ENTRIES TO THE ONES GIVEN HERE

* IF YOU DO NOT UNDERSTAND AN ENTRY TYPE ?  
- E.G. OSP? <er>

* THE SCREEN FOR OUR LAB TWO CONFIGURATION FOLLOW

Hardware

(OSP)	80130	Operating System Extension (Yes/No)	No
(OTU)	80130	Timer Used (Yes/No)	No
(OPU)	80130	PIC Used (Yes/No)	No
(OCD)	80130	Copyright = 1981 (Yes/No)	Yes
(BL)	80130	Base Address Location (40h-0FFFFh)	0000H
(BP)	80130	Base Port Address (0-0FFFFH)	0000H
(MP)	8259A	Master Port (0-0FFFFH)	00C0H
(MPS)		Master PIC Port Separation (0-OFFH)	0002H
(SIL)		Slave Interrupt Levels (1-7/None)	None
(LSS)		Level Sensitive Slaves (1-7/None)	None
(LSP)		Local Slave PICS (1-7/None)	None
(TP)	8253	Timer Port (0-0FFFFH)	00D0H
(CIL)		Clock Interrupt Level (0-7)	0002H
(CN)		Timer Counter Number (0,1,2)	0000H
(CI)		Clock Interval (0-0FFFFH msec)	000AH
(CF)		Clock Frequency (0-0FFFFH khz)	04CDH
(TPS)		Timer Port Separation (0-OFFH)	0002H
(NPX)		Numeric Processor Extension (Yes/No)	Yes
(NIL)		NPX Interrupt Level (Encoded)	0008H



***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

Memory

Type : RAM = low, high  
 Type : ROM = low, high  
 Type : RAM = 0500H, F7FFH

Sub-systems

(UDI) Universal Development Interface (Yes/No)	No
(HI) Human Interface (Yes/No)	No
(AL) Application Loader (Yes/No)	No
(EIO) Extended I/O System (Yes/No)	Yes
(BIO) Basic I/O System (Yes/No)	Req
(DB) Debugger (Yes/No)	No
(TH) Terminal Handler (Yes/No)	No
(CA) Crash Analyzer (Yes/No)	No
(UIR) UDI in ROM (Yes/No)	No
(CAR) Crash Analyzer in ROM (Yes/No)	No
(RIR) Root Job in ROM (Yes/No)	No

EIOS

(ASC) All Sys Calls in EIOS	Yes
(ABR) Automatic Boot Device Recognition (Yes/No)	No
(DLN) Default System Device Logical Name (1-12 characters)	
(DPN) Default System Device Physical Name (1-12 characters)	
(DFD) Default System Device File Driver (Phys/Str/Named)	Named
(DO) Default System Device Owners ID (0-0FFFFH)	0000H
(EBS) Internal Buffer Size (0-0FFFFh)	0400H
(DDS) Default IO Job Directory Size (5-0FF0h)	0020H
(ITP) Internal EIOS Task's Priorities (0-0FFH)	0083H
(PMI) EIOS Pool Minimum (0-0FFFFH)	0180H
(PMA) EIOS Pool Maximum (0-0FFFFH)	FFFFH
(EIR) Extended I/O System in ROM (Yes/No)	No

***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

I/O Users

User : user name, Owner-ID (, ID, ID, ID, ID)  
 User : LAB2, 0000H, FFFFH

Logical Names

Logical Name : logical_name, device_name, file_driver, owners-id  
 (1-12 Chars , 1-14 Chars , Physical/Stream/Named, 0-OFFFFH)  
 Logical Name : BB, BB, Physical, 0000H  
 Logical Name : STREAM, STREAM, Stream, 0000H  
 Logical Name : TO, TO, Physical, 0000H  
 Logical Name : FDO, WFDDO, Named, 0000H

I/O Jobs

(IJD) I/O Job Default Prefix (Logical Name)	TO
(DU) Default User (I/O User)	LAB2
(PMI) Pool Minimum (20H - OFFFFH)	0260H
(PMA) Pool Maximum (20H - OFFFFH)	FFFFH
(AEH) Address of Exception Handler (CS:IP)	0000H:0000H
(EM) Exception Mode (Never/Prog/Environ/All)	Never
(PV) Parameter Validation (Yes/No)	Yes
(TP) Task Priority (0-OFFH)	009BH
(TSA) Task Start Address (CS:IP)	0104H:0002H
(DSB) Data Segment Base (0-OFFFFH)	0000H
(SSA) Stack Segment Address (SS:SP)	0000H:0000H
(SS) Stack Size (0-OFFFFH)	0300H
(NPX) Numeric Processor Extension Used (Yes/No)	No

BIOS

(ASC) All Sys Calls in BIOS (Yes/No)	Yes
(ADP) Attach Device Task Priority (1-OFFH)	0081H
(TF) Timing Facilities Required (Yes/No)	Yes
(TTP) Timer Task Priority (0-OFFH)	0081H
(CON) Connection Job Delete Priority (0-OFFH)	0082H
(ACE) Ability to Create Existing Files (Yes/No)	Yes
(SMI) System Manager ID (Yes/No)	Yes
(CUT) Common Update Timeout (0-OFFFFH)	03E8H
(CST) Control-Sequence Translation (Yes/No)	Yes
(PMI) BIOS Pool Minimum (0-OFFFFH)	0C00H
(PMA) BIOS Pool Maximum (0-OFFFFH)	0C00H
(BIR) Basic I/O System in ROM (Yes/No)	No

***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

Intel Terminal Driver	
(IIL) Input Interrupt Level (Encoded)	0068H
(OIL) Output Interrupt Level (Encoded)	0078H
(UDP) USART Data Port (0-OFFFFH)	00D8H
(USP) USART Status Port (0-OFFFFH)	00DAH
(IRP) 8253 Input Rate Port (0-OFFFFH)	00D4H
(ICP) 8253 Input Control Port (0-OFFFFH)	00D6H
(IRC) 8253 Input Counter Number (0-2)	0002H
(IRM) Input Rate Maximum (0-OFFFFFFFFH)	00012C00H
(ORP) 8253 Output Rate Port (0-OFFFFH)	0000H
(OCP) 8253 Output Control Port (0-OFFFFH)	0000H
(ORC) 8253 Output Counter Number (0-2)	0000H
(ORM) Output Rate Maximum (0-OFFFFFFFFH)	00000000H

Intel Terminal Driver Unit Information	
(NAM) Unit Info Name (1-17 Chars)	t0_uinfo
(LEM) Line Edit Mode (Trans/Normal/Flush)	Normal
(ECH) Echo Mode (Yes/No)	Yes
(IPC) Input Parity Control (Yes/No)	No
(OPC) Output Parity Control (Yes/No)	No
(OCC) Output Control in Input (Yes/No)	Yes
(OSC) OSC Controls (Both/In/Out/Neither)	Both
(DUP) Duplex Mode (Full/Half)	Full
(TRM) Terminal Type (CRT/Hard Copy)	CRT
(MC) Modem Control (Yes/No)	No
(RPC) Read Parity Checking (See Help/0-3)	0000H
(WPC) Write Parity Checking (See Help/0-4)	0000H
(BR) Baud Rate (0-OFFFFH)	2580H
(SN) Scroll Number (0-OFFFFH)	0012H

Intel Terminal Driver Device-Unit Information	
(NAM) Device-Unit Name (1-13 chars)	T0
(UN) Unit Number on this Device (0-OFFH)	0000H
(UIN) Unit Info Name (1-17 Chars)	t0_uinfo
(MB) Max Buffers (0-OFFH)	0000H

***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

Intel iSBC 215/218 Driver

(IL) Interrupt Level (Encoded Level)	0058H
(ITP) Interrupt Task Priority (0-OFFH)	0082H
(WIP) Wakeup I/O Port (0-OFFFFH)	0100H

Intel iSBC 215/218 Unit Information

(NAM) Unit Info Name (1-17 Chars)	uinfo_215fd
(MR) Maximum Retries (0-OFFFFH)	0009H
(CS) Cylinder Size (0-OFFFFH)	0000H
(NC) Number of Cylinders (0-OFFFFH)	004DH
(NFH) Number of Fixed Platters/Disk (0-OFFH)	0000H
(NRH) Number of Remove Platters/Disk (0-OFFH)	0002H
(NS) Number of Sectors/Track (0-OFFFFH)	001AH
(NAC) Number of Aux. Cylinders (0-OFFH)	0000H
(SSN) Starting Sector Number (0-OFFFFFFFFH)	00000000H
(BTI) Bad Track Information (Yes/No)	Yes

Intel iSBC 215/iSBX 218 Device-Unit Information

(NAM) Device-Unit Name (1-13 chars)	WFDDO
(PFD) Physical File Driver Required (Yes/No)	Yes
(NFD) Named File Driver Required (Yes/No)	Yes
(SDD) Single or Double Density Disks (Single/Double)	Double
(SDS) Single or Double Sided Disks (Single/Double)	Double
(EFI) 8 or 5 Inch Disks (8/5)	8
(GRA) Granularity (0-OFFFFH)	0100H
(DSZ) Device Size (0-OFFFFFFFFH)	000F9700H
(UN) Unit Number on this Device (0-OFFH)	0008H
(UIN) Unit Info Name (1-17 Chars)	uinfo_215fd
(UDT) Update Timeout (0-OFFFFH)	0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFH)	0006H
(FUP) Fixed Update (True/False)	True
(MB) Max Buffers (0-OFFH)	00FFH

Nucleus

(ASC) All Sys Calls (Yes/No)	Yes
(PV) Parameter Validation (Yes/No)	Yes
(ROD) Root Object Directory Size (0 - OFF0h)	0028H
(MTS) Minimum Transfer Size (0-OFFFFH)	0040H
(DEH) Default Exception Handler (Yes/No/Deb/Use)	Yes
(NEH) Name of Ex Handler Object Module (1-32chs)	
(EM) Exception Mode (Never/Program/Environ/All)	Never
(NR) Nucleus in ROM (Yes/No)	No

***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

User Jobs

ODS) Object Directory Size (0-OFF0H)	000AH
PMI) Pool Minimum (20H - OFFF0H)	0030H
PMA) Pool Maximum (20H - OFFF0H)	FFFFH
MOB) Maximum Objects (1 - OFFF0H)	FFFFH
MTK) Maximum Tasks (1 - OFFF0H)	FFFFH
MPR) Maximum Priority (0 - OFFH)	0000H
AEH) Address of Exception Handler (CS:IP)	0000H:0000H
EM) Exception Mode (Never/Prog/Environ/All)	Never
PV) Parameter Validation (Yes/No)	Yes
TP) Task Priority (0-OFFH)	0000H
TSA) Task Start Address (CS:IP)	0080H:0002H
DSB) Data Segment Base (0-OFFFFH)	0000H
SSA) Stack Segment Address (SS:SP)	0000H:0000H
SS) Stack Size (0-OFFFFH)	0200H
NPX) Numeric Processor Extension Used (Yes/No)	No

Includes and Libraries

Path Name (1-45 Characters)

UDF) UDI Includes and Libs	/RMX5.0/DUDI/
HIF) Human Interface Includes and Libs	/RMX5.0/DINCLSLIBS/
EIF) Extended I/O System Includes and Libs	/RMX5.0/DINCLSLIBS/
ALF) Application Loader Includes and Libs	/RMX5.0/DLOADER/
BIF) Basic I/O System Includes and Libs	/RMX5.0/DINCLSLIBS/
THF) Terminal Handler and Debugger Includes and Libs	/RMX5.0/DDEBTH/
NUF) Nucleus and Root Job Includes and Libs	/RMX5.0/DNUCLUS/
ILF) Interface Libraries	/RMX5.0/DUTILS/
CAF) Crash Analyzer Includes and Libs	/RMX5.0/DUDI/
DTF) Development Tools Path Names	/LANG/

Generate File Names

File Name (1-55 Characters)

ROF) ROM Code File Name	:LAB:NONE
RAF) RAM Code File Name	:LAB:RMX86

***** LAB FIVE (EIOS CONFIG THROUGH ICU) *****

STEP4:

AFTER YOU ENTER ALL OF THE SCREENS ENTER G TO GENERATE  
EXIT THE ICU

SUBMIT THE ICU.CSD FILE TO GENERATE YOUR SYSTEM

-SUBMIT :LAB:ICU.CSD

STEP5:

YOU MUST NOW ADD THE USER JOB AND THE SDB TO THE SYSTEM,  
USING THE LIB86 UTILITY

```
-LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
DELETE :LAB:RMX86(INT3TASKMOD)
ADD /DINT3/INT3JOB to :LAB:RMX86
EXIT
```

STEP6:

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION  
THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB2/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION  
THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG  
YOUR CODE IF NECESSARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

## CHAPTER 10

### THE IRMX 86 APPLICATION LOADER

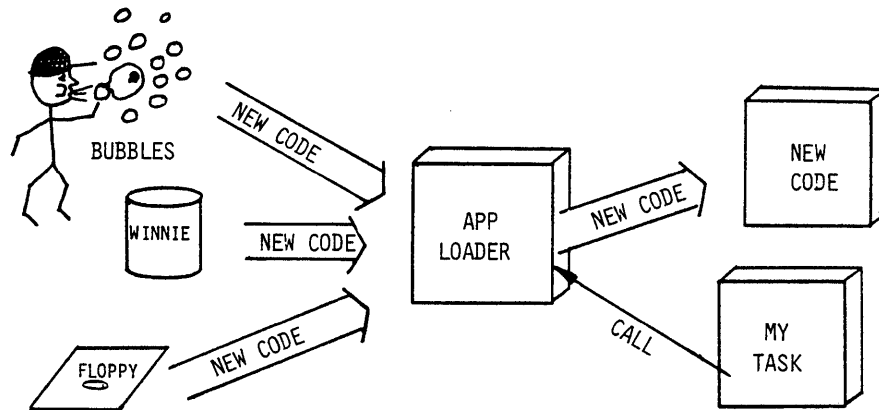
- LOADER FUNCTION
- TYPES OF LOADABLE CODE
- SYSTEMS WITHOUT THE EIOS
- LOADER RESULT SEGMENT
- SYSTEMS WITH THE EIOS





### APP LOADER FUNCTION

- THE APP LOADER MOVES CODE FROM SECONDARY STORAGE INTO RAM



10-1

### NAMED FILES

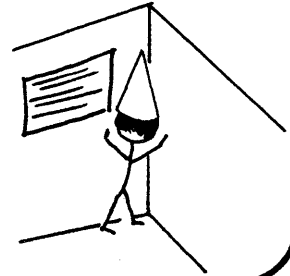
- THE APP LOADER CAN LOAD OBJECT CODE FROM ANY DEVICE THAT SUPPORTS iRMX 86 NAMED FILES
- THE iRMX 86 O.S. IS CURRENTLY DELIVERED WITH SUPPORT FOR THE FOLLOWING DEVICES
  - ISBC 204      ISBC 218
  - ISBC 206      ISBC 220
  - ISBC 215      ISBC 254
- IT WILL ALSO SUPPORT CUSTOM DEVICES, FOR WHICH YOU HAVE WRITTEN A DEVICE DRIVER

10-2

## TYPES OF LOADABLE CODE

- ABSOLUTE (ABS)

- CODE IS LOCATED AT AN ABSOLUTE LOCATION IN MEMORY WITH THE LOC86 UTILITY PROGRAM
- THE USER "MUST" HAVE THIS LOCATION RESERVED AT CONFIGURATION
- THE LOADER ALWAYS LOADS THE CODE AT THE SAME ABSOLUTE LOCATION
- PL/M MODELS MAY BE MEDIUM, LARGE OR COMPACT

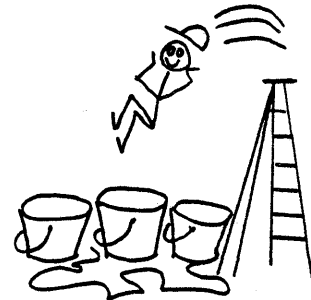


10-3

## TYPES OF LOADABLE CODE (CONTINUED)

- POSITION INDEPENDENT CODE (PIC)

- CODE IS NEVER LOCATED, INSTEAD THE BIND OPTION IS USED WHEN LINKING WITH THE LINK86 UTILITY PROGRAM
- PIC CODE CAN BE LOADED INTO ANY MEMORY LOCATION
- THE LOADER OBTAINS 1RMX 86 SEGMENTS "RUNTIME" AND LOADS PIC INTO THE SEGMENTS
- PIC IS RESTRICTED TO USE BY TASKS THAT HAVE ONLY ONE CODE SEGMENT AND ONE DATA SEGMENT
- PL/M MODEL IS COMPACT ONLY



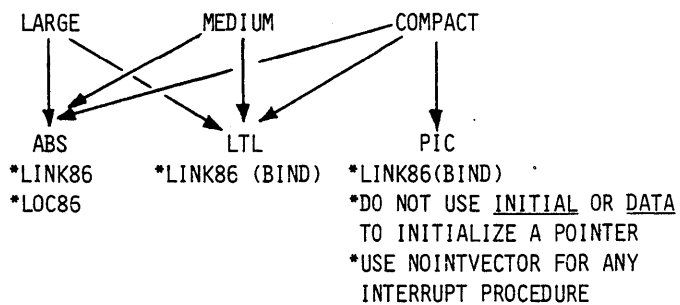
10-4

### TYPES OF LOADABLE CODE (CONTINUED)

- LOAD TIME LOCATABLE (LTL)
  - SIMILAR TO PIC CODE, LINK WITH BIND, CODE MAY BE LOADED ANYWHERE IN MEMORY
  - CAN BE USED BY TASKS HAVING MORE THAN ONE CODE SEGMENT OR MORE THAN ONE DATA SEGMENT
  - PL/M MODELS MAY BE MEDIUM, LARGE OR COMPACT

10-5

### PL/M MODELS AND TYPES OF CODE

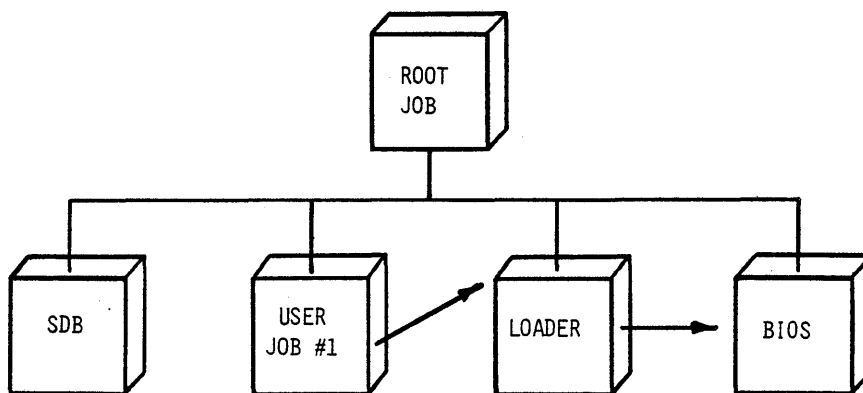


IF YOU DON'T HAVE AN 86-BASED DEVELOPMENT SYSTEM YOU CAN GENERATE ONLY ABS CODE.

10-6

### SYSTEMS WITHOUT THE EIOS

- THE APPLICATION LOADER LIVES IN YOUR SYSTEM AS A FIRST LEVEL JOB.



- YOU CALL THE LOADER THROUGH RQALOAD
- THE LOADER CALLS THE BIOS TO LOAD THE CODE

10-7

### LOADER SYSTEM CALLS

- RQALOAD SYSTEM CALL
  - A SYNCHRONOUS
  - LOADS FROM A FILE INTO MEMORY
  - THE FORM OF THE CALL IS

```
CALL RQALOAD (FILE$CONN, RSP$MBOX, @STATUS);
```

10-8

#### THE LOADER RESULT SEGMENT

- THE LDRS IS RETURNED TO THE RESPONSE MAILBOX AFTER THE LOADER HAS COMPLETED THE LOAD FUNCTION.

EXCEPT\$CODE
RECORD\$COUNT
REC\$TYPE
NUMBER\$UNDEFINED\$REFS
INIT\$IP
CODE\$SEG\$BASE
STACK\$SEG\$BASE
STACK\$OFFSET
STACK\$SEG\$BASE
STACK\$SIZE
CATA\$SEG\$BASE

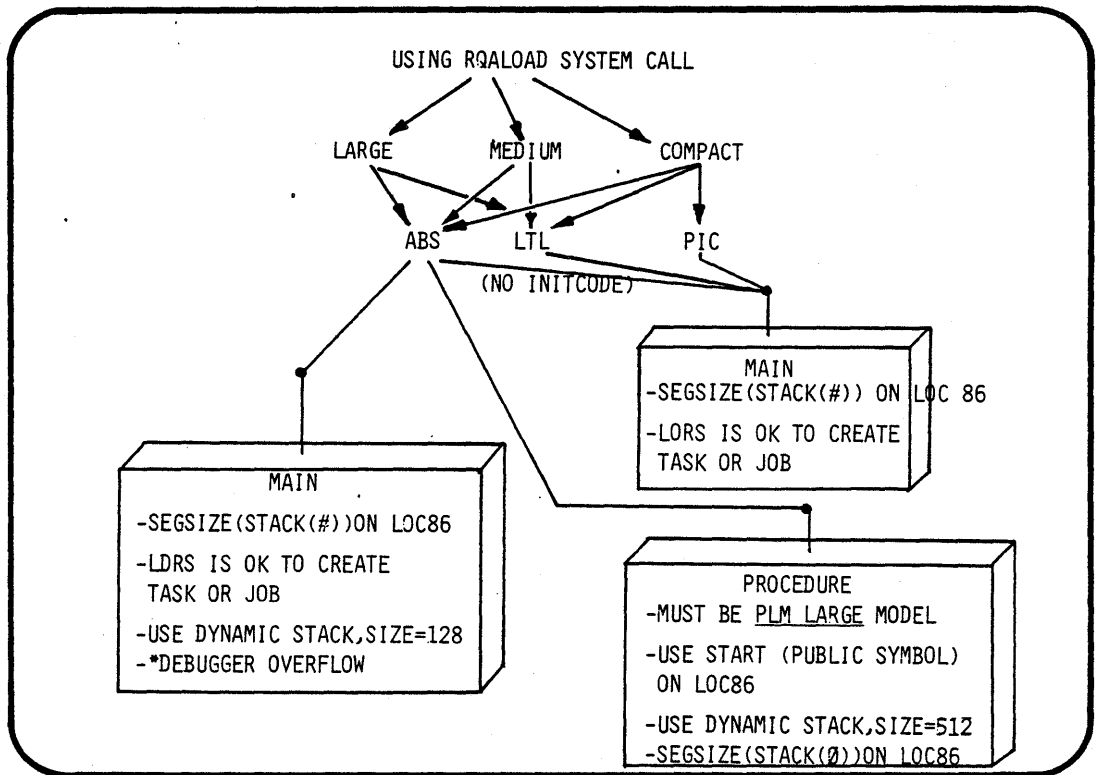
- THE LOADING TASK USES THE INFORMATION IN THE LDRS TO CREATE A TASK OR CREATE A JOB.

10-9

#### CODE ENTRY POINTS

- THE ENTRY POINT IS WHERE EXECUTION BEGINS AFTER THE CODE IS LOADED.
- THE LOADER MUST GET THIS ENTRY POINT FROM THE LOADABLE CODE.
- THERE IS ONE CIRCUMSTANCE IN WHICH THE LOADER DOES NOT REQUIRE AN ENTRY POINT.
  - THE LOADABLE CODE IS ABSOLUTE, AND
  - THE LOADING TASK KNOWS THE ENTRY POINT, AND
  - THE LOADING TASK USES THE RQALOAD SYSTEM CALL

10-10



10-11

LOADABLE PROCEDURES

- WRITE SOURCE CODE AS A PROCEDURE ONLY IF:
  - CODE IS (ABS), WITHOUT USING NOINITCODE IN LOC86
  - YOU ARE LOADING A TASK(S) AND NOT A JOB
  - YOU ARE USING THE RQALOAD SYSTEM CALL
- PROCEDURE EXAMPLE
 

```

FLASHMOD:DO;
$INCLUDE (/RMX/DNUCLUS/NUCLUS.EXT)
DECLARE STATUS WORD;
  FLASHTASK: PROCEDURE PUBLIC;
    DO FOREVER;
      OUTPUT (9CH) = 0FFH;
      CALL RQSLEEP(50, @STATUS);
      OUTPUT (9CH) = 0;
      CALL RQSLEEP(50, @STATUS);
    END; /*END OF FOREVER*/
  END; /*END OF TASK*/
END; /*END OF MODULE*/

```

10-12

### LOADABLE MAIN MODULES

- IF CODE IS LTL OR PIC THEN CODE MUST BE A MAIN MODULE
- A MAIN MODULE CONTAINS EXECUTABLE STATEMENTS AT THE 'OUTERMOST LAYER OF THE MODULE
- WHEN LINKING OR LOCATING USE "SEGSIZE(STACK(###))" TO ASSIGN THE APPROPRIATE STACKSIZE

- MAIN MODULE EXAMPLE

```
FLASHMOD: DO;  
  $INCLUDE(/RMX/DNUCLUS/NUCLUS.EXT)  
  DECLARE STATUS WORD;  
  DO FOREVER;  
    OUTPUT(09CH) = 0FFH;  
    CALL RQSLEEP(25, @STATUS);  
    OUTPUT(09CH) = 0;  
    CALL RQSLEEP(25, @STATUS);  
  END; /*END OF FOREVER*/  
END; /*END OF MODULE*/
```

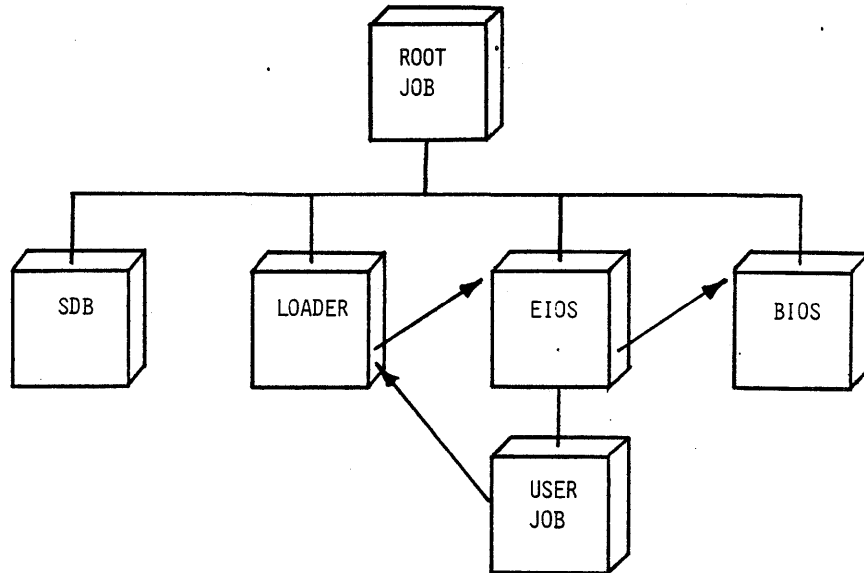
10-13

### QUIZ!

- 1) NAME TYPES OF CODE _____
- 2) WHERE DOES THE LDRS COME FROM?
- 3) NAME ONE ADVANTAGE OF USING A PROCEDURE OVER A MAIN MODULE?

10-14

SYSTEMS WITH THE EIOS (THE EASY WAY)



- THE USER CALLS THE LOADER THROUGH RQALOADIOJOB OR RQSLOADIOJOB
- THE LOADER CALLS THE EIOS ON YOUR BEHALF (YOU MUST BE AN IOJOB), TO LOAD THE CODE

10-15

SYNCHRONOUS LOADER SYSTEM CALL

- RQSLOADIOJOB
  - SYNCHRONOUS
  - LOADS CODE INTO MEMORY, STARTS NEW IO JOB
  - THE FORM OF THE CALL IS

```

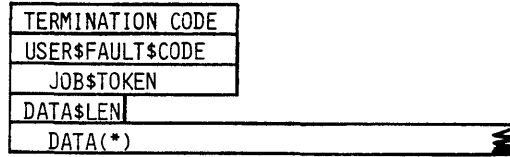
JOB$TOKEN = RQSLOADIOJOB (a(10,':FD0:FLASH'), /*PATH PTR*/
                          0,0FFFFH, /*POOL LOWER,UPPER*/
                          0, /*EXCEPT HANDLER*/
                          0, /*JOB FLAGS*/
                          130, /*TASK PRIORITY*/
                          0, /*TASK FLAGS*/
                          RSPMBOX, /*RESPONSE MAILBOX*/
                          aSTATUS);
  
```

10-16



### THE TERMINATION MESSAGE

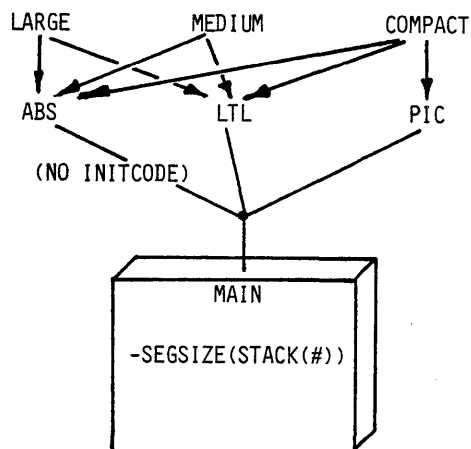
- WHEN THE NEWLY CREATED IO JOB MAKES AN RQEXITIOJOB CALL
  - THE PARENT JOB RECEIVES A TERMINATION MESSAGE
  - THE PARENT JOB WAITS AT THE RSPMBOX FOR THIS MESSAGE
  - THE FORMAT OF THE MESSAGE IS



- REFER TO RQCREATEIOJOB IN EIOS REFERENCE MANUAL.

10-17

### RQSLOADIOJOB



10-18



## **CHAPTER 11**

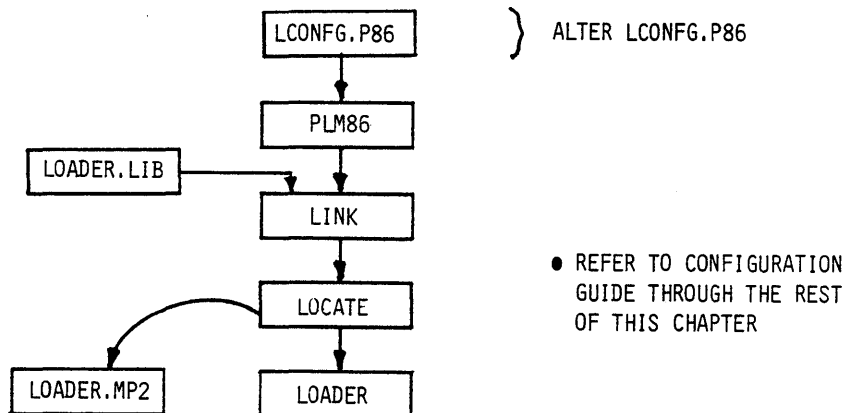
# **APPLICATION LOADER**

Configuration



### STEPS IN BUILDING THE APPLICATION LOADER

- 1) A CONFIGURATION FILE IS NEEDED (LCONFIG.P86)
- 2) COMPILE (PLM86) THE CONFIGURATION FILE
- 3) LINK AND LOCATE THE APP LOADER



11-1

### LCONFIG.P86

```

LOADER$CONFIG: DO;
DECLARE BUF$SIZE          LITERALLY '1024'; /* BYTES */
DECLARE RDBUF$SIZE        LITERALLY '1024'; /* BYTES */

DECLARE LBUF$SIZE         WORD PUBLIC DATA(BUF$SIZE + 11);
DECLARE L$RDBUF$SIZE      WORD PUBLIC DATA(RDBUF$SIZE);

DECLARE $DEFAULT$MEMPOOL  WORD PUBLIC DATA(200H); /*PAGES*/

END LOADER$CONFIG;
  
```

THE SIZE OF THE LOADER INTERNAL BUFFERS FOR OBJECT RECORDS.

THE SIZE OF THE INPUT BUFFERS

L\$DEFAULT\$MEMPOOL SELECTS THE DYNAMIC MEMORY (MEMPOOL) REQUIREMENT FOR THE OBJECT-FILE BEING LOADED.

NOTE: THIS VALUE IS SPECIFIED IN PAGES (1 PAGE = 16 BYTES). THIS PARAMETER HAS NO EFFECT ON 'RQ\$A\$LOAD' SYSTEM CALL.

11-2

## THE SUBMIT FILE (86/330 STYLE)

```

PLM86 /RMX/DLOADER/LCONFIG330.P86 COMPACT OPTIMIZE(3) NOTYPE ROM &
OBJECT(/RMX/DLOADER/LCONFIG.OBJ) PRINT(/RMX/DLOADER/LCONFIG330.LST) WORKFILES(:WORK.:WORK:)

LINK86 /RMX/DLOADER/LOADR(2) LIB(LDRINT), &
/RMX/DLOADER/LCONFIG.OBJ, &
/RMX/DLOADER/LOADR(2) LIB(LDRENT), &
/RMX/DLOADER/LJBCR(3) LIB, &
/RMX/DLOADER/LOADR(2) LIB, &
/RMX/DEIOS/EPIFC.LIB, &
/RMX/DBIOS/IPIFC.LIB, &
/RMX/DNUCLUS/RPIFC.LIB &
TO /RMX/DLOADER/LOADER.LNK &
PRINT(/RMX/DLOADER/LOADER.MP1) &

NOPUBLICS EXCEPT(RQLOADERINITTASK, RQLOADERINITERROR)

LOC86 /RMX/DLOADER/LOADER.LNK TO /RMX/DLOADER/LOADER &
MAP PRINT(/RMX/DLOADER/LOADER.MP2) &
NOLINES NOCOMMENTS NOSYMBOLS &
ORDER(CLASSES(CODE, DATA)) &
SEGSIZE(STACK(0), DATA(2)) &
ADDRESSES(CLASSES(CODE(21))) &

```

CP2 =CONFIGURATION PARAMETER 2  
 A - ABSOLUTE  
 P - ABSOLUTE + PIC  
 L - ABSOLUTE + PIC + LTL  
 O - ABSOLUTE + PIC + LTL WITH OVERLAYS  
  
 CP 3 =CONFIGURATION PARAMETER 3  
 N - NO LOAD-JOB FUNCTION  
 A - ASYNCHRONOUS LOAD-JOB FUNCTION  
 S - ASYNCHRONOUS + SYNCHRONOUS LOAD-JOB FUNCTION

LOCATE ADDRESS

*SUBMIT /RMX/DLOADER/LOADER(DATE, LOC_ADR, CP2, CP3)

11-3

## LOCATE ADDRESSES

- THE LOC86 PROGRAM GENERATES A MAP FILE CALLED /RMX/DLOADER/LOADER.MAP
- EXAMINING THE MAP WE OBTAIN THE ENDING ADDRESS OF THE LOADER (USED FOR LOCATING THE NEXT JOB)

### MEMORY MAP OF MODULE LDRINT

#### SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS
00200H	00216H	0017H	A	(ABSOLUTE)	
1D150H	1F432H	22E3H	W	CODE	CODE
1F440H	1F441H	0002H	G	DATA	DATA
1F450H	1F450H	0000H	G	??SEG	
1F450H	1F450H	0000H	W	STACK	STACK
1F450H	1F450H	0000H	W	MEMORY	MEMORY

11-4

DON'T FORGET ROOT JOB MACRO!



- ADD A %JOB MACRO TO YOUR SYSTEM ROOT JOB
- MUST BE BETWEEN IOS AND EIOS %JOB MACROS

```
;APP LOADER
%JOB(0,          %'OBJECT DIRECTORY SIZE
      20H,20H,   %'POOL SIZE (MIN, MAX)
      50,5,     %'MAX OBJECTS AND TASKS
      0,        %'MAX JOB PRIORITY
      0:0,0,    %'EXCEPTION HANDLER, MODE
      0,        %'JOB FLAGS
      130,      %'INIT TASK PRIORITY
      1d15:0,  %'INIT TASK ENTRY
      0,        %'INIT TASK DATA SEGMENT ADDRESS
      0,160,   %'INIT TASK STACKADDRESS, STACKSIZE
      0)       %'INIT TASK FLAGS
```

11-5

ICU86

- ICU86 WILL
  - CREATE LCONFIG.P86
  - CREATE A SUBMIT FILE THAT INCLUDES COMPILING LINKING AND LOCATING OF THE APP LOADER
  - ADD A JOB MACRO TO THE ROOT JOB
  - COMPUTE THE STARTING ADDRESS OF THE LOADER

11-6





OBJECTIVES:

EXECUTE A STUDENT APP LOADER APPLICATION JOB IN AN RMX86 O.S. ENVIRONMENT

INTRODUCE (EIOS) SYSTEM CALLS:

- RQ\$\$\$LOAD\$IO\$JOB

USE ICU TO BUILD A SYSTEM CONTAINING:

- A NUCLEUS
- A BIOS
- AN EIOS
- AN APPLICATION LOADER

CREATE SOURCE CODE:

- A SOURCE FILE NAMED START.P86
- A SOURCE FILE NAMED LDRLAB.P86

COMPILE (PLM86), LINK, AND LOCATE AN APPLICATION JOB, THAT WILL CALL UPON THE EIOS TO COMMUNICATE WITH A FILE IN A FLOPPY, CONTAINING A LOADABLE JOB, LOAD THE JOB AND EXECUTE

THE LOADABLE JOB WILL CONTAIN A SIMPLE TASK THAT FLASHES THE LIGHTS IN THE LIGHT BOX

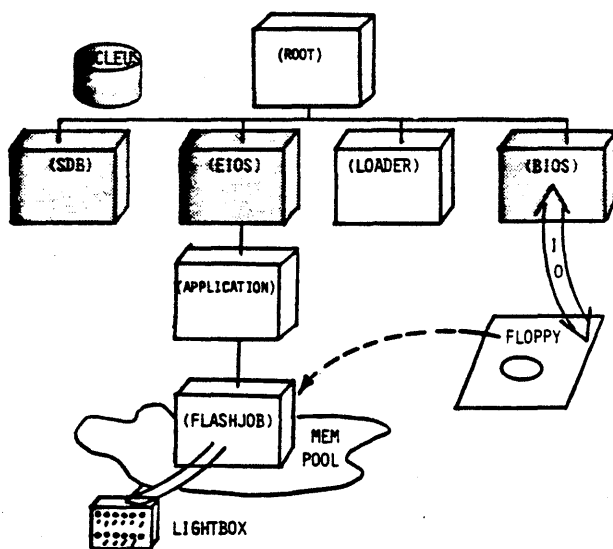
STEP 1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED (/ "TEAM NAME" /LAB6) AS THE LOGICAL NAME (:LAB:)

- A FILE / "TEAM NAME" /LAB6 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *

LAB 6 (PART A)



• LAB OBJECTIVE

- USE THE APPLICATION LOADER TO LOAD A MAIN MODULE (JOB), SYNCHRONOUSLY USING THE EIOS
- THE STUDENT WILL BE GIVEN THE NUCLEUS, EIOS, BIOS, SDB
- THE STUDENT WILL SUPPLY APPLICATION, ROOT, LOADER AND WILL BUILD AND PLACE FLASH JOB

***** LAB SIX (APP LOADER SYSTEM) *****

STEP2:

BUILD A SYSTEM CONTAINING THE APPLICATION LOADER

- * WE WILL USE THE SYNCHRONOUS LOADER CALL TO LOAD AN IO JOB FROM A FLOPPY INTO MEMORY
- * SINCE THIS REQUIRES THAT OUR SYSTEM INCLUDE THE EXTENDED IO, WE WILL USE THE SYSTEM WE ALREADY BUILT IN LAB FIVE.

THE (ICU) IS INVOKED BY TYPING THE FOLLOWING  
-ICU86 /RII5.0/LAB5/ICU.DEF TO :LAB:ICU.DEF

- * WHERE ICU.DEF IS THE NAME OF THE FILE WE HAVE CHOSEN TO CONTAIN THE INFORMATION NEEDED TO CONFIGURE OUR O.S.

WHEN THE ICU SIGN ON ENTER THE COMMAND C , TO MODIFY THE SCREENS

APP LOADER SCREENS:

Application Loader

(IBS) Internal Buffer Size (0 - 0FFFFh)	0400H
(RBS) Read Buffer Size (0 - 0FFFFh)	0400H
(LJT) Load Job Type (None/Async/Sync)	Synchronous and Asynchronous
(DMP) Default Memory Pool Size (0 - 0FFFFh)	0100H
(CT) Code Type (Abs/Pic/Ltl/Ovr)	Overlay, LTL, PIC and Abs
(ALR) Application Loader in ROM (Yes/No)	No

STEP3:

AFTER YOU ENTER ALL OF THE SCREENS ENTER G TO GENERATE

EXIT THE ICU

SUBMIT THE ICU.CSD FILE TO GENERATE YOUR SYSTEM

-SUBMIT :LAB:ICU.CSD

STEP4:

BUILD THE RESIDENT JOB

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:LDRLAB.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:LDRLAB.P86

* THIS SOURCE FILE IS THE APPLICATION TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

-----  
CREATE A RESPONSE MAILBOX

-----  
LOAD THE JOB FROM :FDO:FLASHJOB

-----  
CREATE FILE CONNECTION TO TERMINAL */* @(4,':TO:')

-----  
OPEN TERMINAL FILE */* (R/W) , SHARE ALL , *NOTE

-----  
WRITE MESSAGE TO TERMINAL */* MESSAGE = "FILE HAS BEEN LOADED

-----  
CLOSE TERMINAL FILE */*

-----  
** DELETE SELF ** CALL EXIT\$IO\$JOB  
-----

***** LAB SIX (APP LOADER SYSTEM) *****

STEP5:

- * ROOT JOBS ABSOLUTELY ADDRESS THE STARTING LOCATION OF THE STUDENT'S JOB CODE. THE ENTRY POINT MAY VARY IF INTERNAL PROCEDURES OR CHARACTER CONSTANTS ARE USED.  
FOR THIS REASON IT IS ADVISABLE TO CREATE AND LINK A START TASK TO THE REST OF THE APPLICATION CODE TO FIX THE ENTRY POINT'S OFFSET INTO THE CODE
- * THIS APPLICATION JOB WILL BE A SECOND LEVEL JOB. A TASK WITHIN THIS JOB IS NOT REQUIRED TO MAKE A CALL TO RQ\$END\$INIT\$TASK, THE EIOS CODE SUPPLIES A TASK THAT CALLS RQ\$END\$INIT\$TASK
- * IN ORDER TO DEBUG OUR CODE BEFORE IT "CRASHES" WE MAY WISH TO INVOKE THE 957 MONITOR AT THE START OF OUR JOB'S EXECUTION. THIS CAN EASILY BE ACCOMPLISHED BY PLACING A "CAUSE\$INTERRUPT(3)" INSTRUCTION AT THE BEGINNING OF OUR CODE (IN OUR START TASK).
- * WE WILL USE THE SAME START TASK THAT WE USED IN LAB TWO

-COPY /RII5.0/LAB2/START.P86 TO :LAB:START.P86

***** LAB SIX (APP LOADER SYSTEM) *****

STEP6:

COMPILE THE SOURCE FILES (START.P86 AND LDRLAB.P86)

- PLM86 :LAB:START.P86
- PLM86 :LAB:LDRLAB.P86

- * IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING
- * IF COMPILATION IS SUCCESSFUL THE COMPILER WILL CREATE FOR EACH OF THE SOURCE FILES:

- A LIST FILE NAMED ":LAB:(SOURCE).LST"
- AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 :LAB:START.OBJ,&
       :LAB:LDRLAB.OBJ,&
       /RMX5.0/DUTILS/EPIFL.LIB,&
       /RMX5.0/DUTILS/IPIFL.LIB,&
       /RMX5.0/DUTILS/LPIFL.LIB,&
       /RMX5.0/DUTILS/RPIFL.LIB &
       TO :LAB:JOB.LNK &
       NOMAP
```

LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS

```
LOC86 :LAB:JOB.LNK &
      TO :LAB:LABJOB &
      SC(3) SEGSIZE(STACK(0)) &
      ORDER(CLASSES(CODE,DATA,STACK)) &
      ADDRESSES(CLASSES(CODE(1040H))) &
      NOINITCODE &
      OC(PURGE)
```

YOU MUST NOW ADD THE USER JOB AND THE SDB TO THE SYSTEM,  
USING THE LIB86 UTILITY

```
-LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
DELETE :LAB:RMX86(INT3TASKMOD)
ADD /DINT3/INT3JOB to :LAB:RMX86
EXIT
```

- * THE STUDENT MAY "OPTIONALLY" USE A "GIVEN" SUBMIT FILE THAT WILL COMPILE , LINK , LOCATE AND ADD THE FINAL MODULE TO THE SYSTEM

- SUBMIT :LAB:JOB.CSD

STEP6:

BUILD THE NON RESIDENT JOB

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:FLASHJOB.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:FLASHJOB.P86

* THIS SOURCE FILE IS THE NON RESIDENT TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

```
-----  
DO FOREVER  
-----  
OUTPUT OFFH TO PORT 09CH  
-----  
GO TO SLEEP FOR 1/4 SEC  
-----  
OUTPUT 0 TO PORT 09CH  
-----  
GOT TO SLEEP FOR 1/4 SEC  
-----
```

STEP7:

COMPILE THE SOURCE FILES (FLASHJOB.P86)

- PLM86 :LAB:FLASHJOB.P86

* IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING

* IF COMPILATION IS SUCCESFUL THE COMPILER WILL CREATE

- A LIST FILE NAMED ":LAB:(SOURCE).LST"
- AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 :LAB:FLASHJOB.OBJ,      &  
        /RMX5.0/DUTILS/RPIFL.LIB  &  
        TO :LAB:FLASHJOB &  
        NOMAP SEGSIZE(STACK(512)) BIND
```

COPY THE JOB INTO A FILE ON THE FLOPPY

COPY :LAB:FLASHJOB OVER :FDO:FLASHJOB

***** LAB SIX (APP LOADER SYSTEM) *****

STEP8:

* A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB6/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NESSESARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

* GOOD LUCK...!





## **CHAPTER 12**

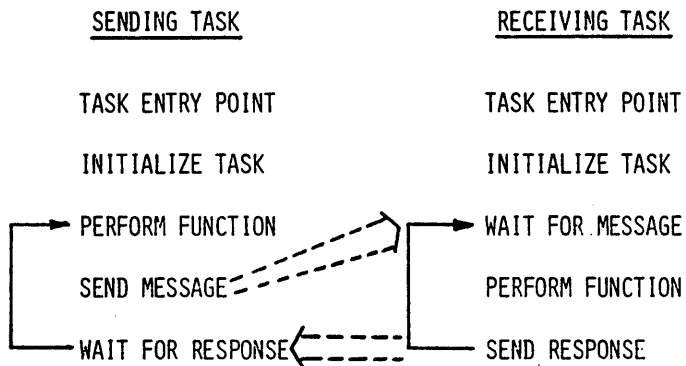
### **iMMX 800**

#### **Multi Message eXchange Software**

- BASIC CONCEPTS
- CHANNELS
- MMX SYSTEM CALLS
- THE MMX JOB



iMMX 800  
MESSAGE TRANSFER  
MESSAGE SENDING AND RECEIVING MODEL

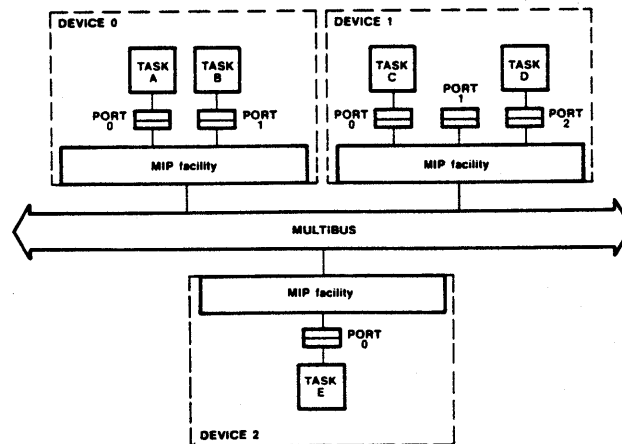


- RMX SOFTWARE IMPLEMENTS THIS MODEL OF INTER TASK COMMUNICATION BETWEEN TASKS RESIDING ON THE SAME DEVICE (BOARD).
- iMMX SOFTWARE GENERALIZES THE MODEL TO ACCOMMODATE INTERDEVICE COMMUNICATION.

12-1

BASIC CONCEPTS

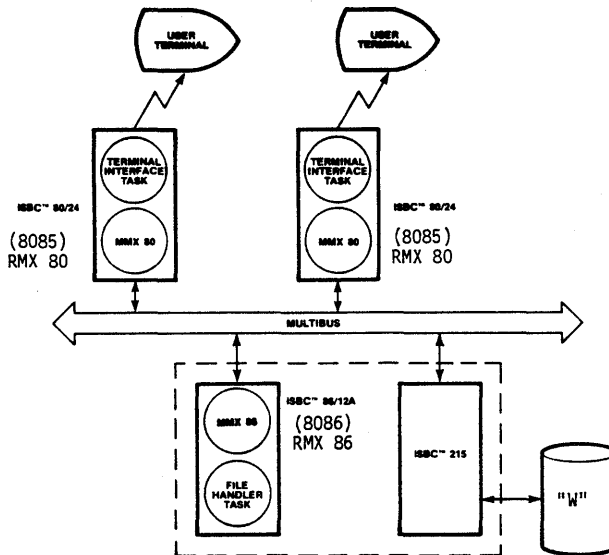
- DEVICE - A PROCESSOR BOARD IN A SYSTEM
- PORT - A LOGICAL DELIVERY MECHANISM WHICH UTILIZED FIFO ORDER (QUEUE)



12-2

### DATA BASE APPLICATION EXAMPLE

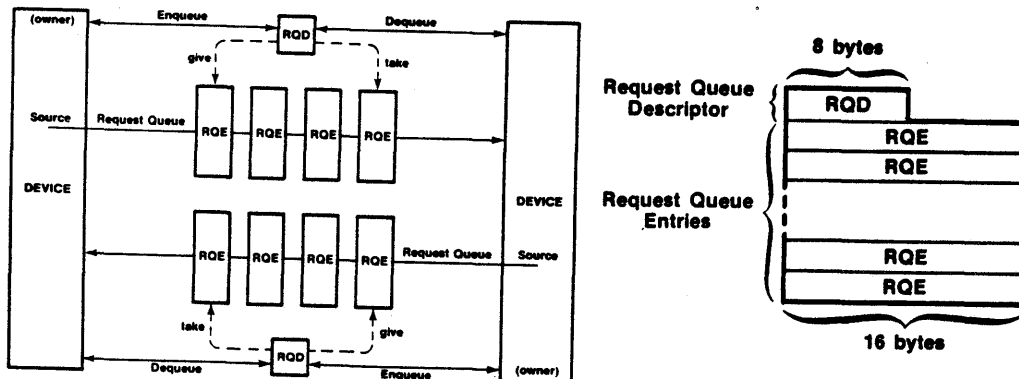
- 2 OPERATORS - 2 TERMINALS ACCESSING DATA FILES
- THE TERMINALS CONTROLLED BY RMX 80
- THE DATA BASE ("WINCHESTER") CONTROLLED BY RMX 86 BASIC I/O SYSTEM



12-3

### CHANNELS

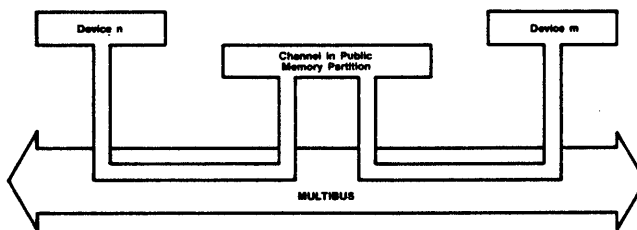
- COMMUNICATION BETWEEN DEVICES IS IMPLEMENTED USING CHANNELS
  - A CHANNEL CONSISTS OF A PAIR OF QUEUES
  - ONE CHANNEL MUST BE DEFINED FOR EACH DEVICE PAIR WHICH WILL COMMUNICATE WITH EACH OTHER



12-4

### CHANNELS

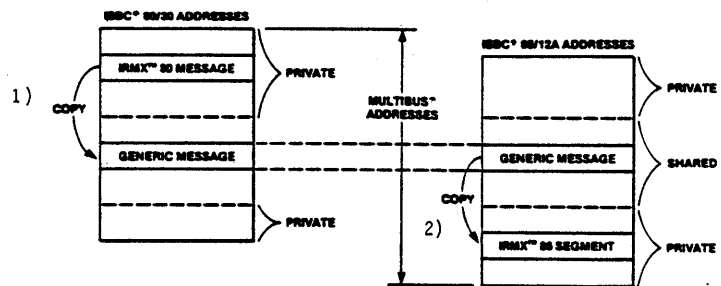
- A CHANNEL MUST RESIDE IN A MEMORY SEGMENT ACCESSIBLE BY BOTH DEVICES WHICH USE THAT CHANNEL
  - GLOBAL MEMORY
  - DUAL PORT MEMORY



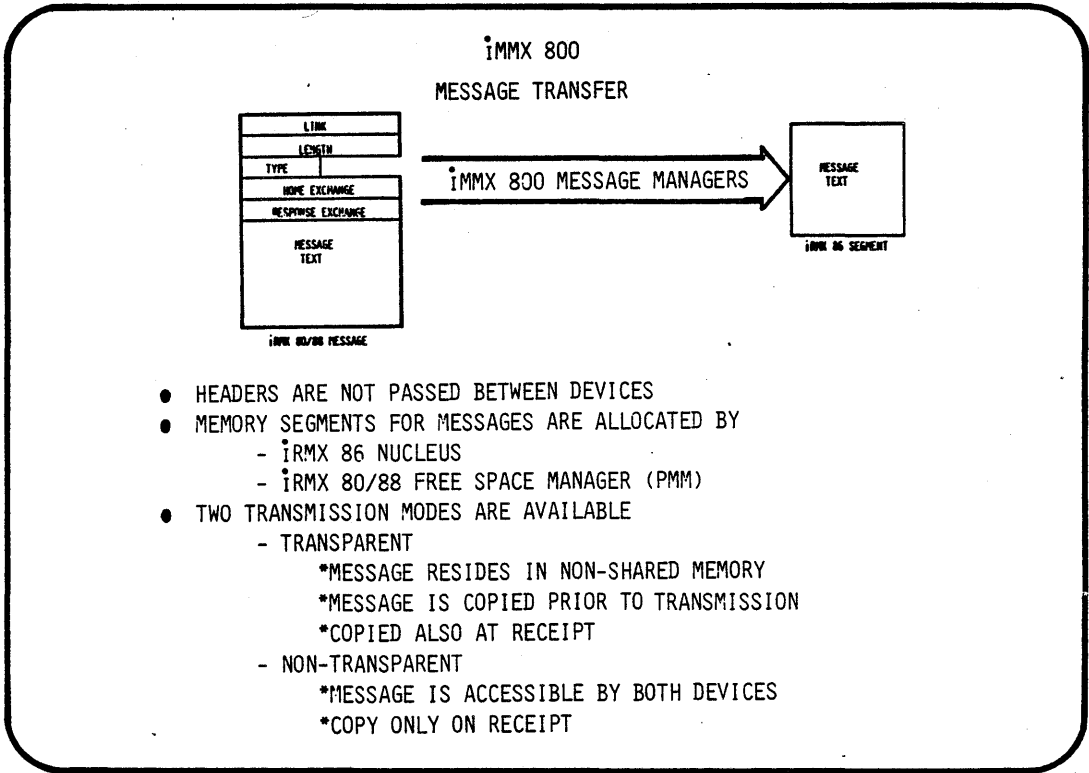
12-5

### MESSAGE TRANSFER (FULL DELIVER)

- 1) MESSAGE IS COPIED ON SEND
- 2) MESSAGE IS COPIED ON RECEIVE



12-6



SUPPORTED SINGLE BOARD COMPUTERS

iRMX 80 OPERATING SYSTEM	iRMX 88 OPERATING SYSTEM	iRMX 86 OPERATING SYSTEM
iSBC 80/24	iSBC 86/05	iSBC 86/05
iSBC 80/30	iSBC 86/12A	iSBC 86/12A
iSBC 544	iSBC 86/14	iSBC 86/14
iSBC 569	iSBC 86/30	iSBC 86/30
	iSBC 88/25	iSBC 88/25
	iSBC 88/40	iSBC 88/40
	iSBC 88/45	iSBC 88/45

## iMMX 800

- COMES IN THREE VERSIONS
  - OPERATION UNDER iRMX 80 NUCLEUS
    - *iMMX 800/80
  - OPERATION UNDER iRMX 88 NUCLEUS
    - *iMMX 800/880 FOR NON-MEGABYTE SUPPORT
    - *iMMX 800/881 FOR MEGABYTE SUPPORT
  - OPERATION UNDER iRMX 86 NUCLEUS
    - *iMMX 800/86
- ALL THREE VERSIONS PRESENT IDENTICAL USER INTERFACES

12-9

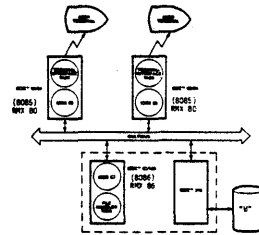
## SOFTWARE MEMORY REQUIREMENTS

EXECUTIVE	K BYTES
iRMX 80 OPERATING SYSTEM	3.7K BYTES
iRMX 88 OPERATING SYSTEM 128K SUPPORT 1MB SUPPORT "COMPACT" "LARGE"	4.8K BYTES 5.5K BYTES 6.3K BYTES
iRMX 86 OPERATING SYSTEM	6.6K BYTES

12-10

### APPLICATION EXAMPLE

- WHEN AN OPERATOR ENTERS A REQUEST AT A TERMINAL, THE FOLLOWING SEQUENCE OCCURS:
  1. A TASK ON THE iSBC 80/24 BOARD IN THE TERMINAL BUILDS A MESSAGE THAT MEETS iRMX 80 MESSAGE-FORMAT REQUIREMENTS AND ISSUES A CQXFER CALL TO MMX 80. (CQXFER IS THE NAME OF THE iMMX 80 TRANSFER PROCEDURE).
  2. MMX 80 TRANSFERS THE MESSAGE TO MMX 86 ON THE iSBC 86/12A BOARD.
  3. MMX 86 REFORMATS THE MESSAGE AND PASSES IT TO AN iRMX 86 TASK.
  4. THE I/O SYSTEM PERFORMS THE NECESSARY I/O OPERATIONS FOR THE TASK.
  5. THE TASK PUTS THE DATA IN A MESSAGE THAT SATISFIES RMX 86 FORMAT CONVENTIONS AND ISSUES A CQXFER CALL TO MMX 86.
  6. MMX 86 TRANSFERS THE MESSAGE TO MMX 80 ON THE iSBC 80/24 BOARD.
  7. MMX 80 REFORMATS THE MESSAGE TO MEET iRMX 80 FORMAT REQUIREMENTS AND PASSES IT TO THE iRMX 80 TASK.
  8. THE TASK EXTRACTS THE DATA FROM THE MESSAGE AND SENDS IT TO THE TERMINAL.



12-11

### iMMX SYSTEM CALLS

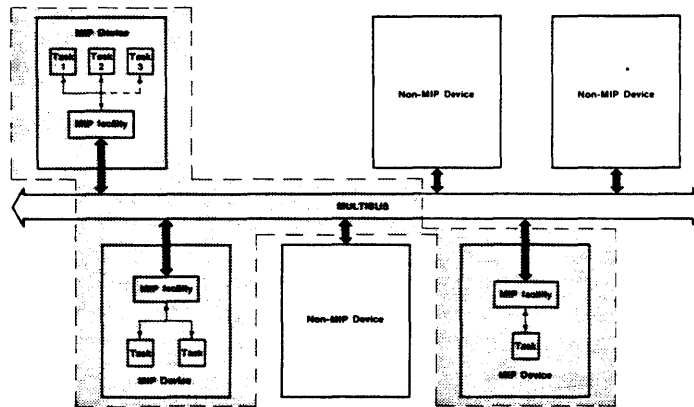
FUNCTION	NAME	DESCRIPTION
FIND PORT	CQFIND	FIND A PORT AND RETURN A CONNECTION-ID.
ACTIVATE PORT	CQACTV	ACTIVATE A PORT FOR RECEIVING MESSAGES FROM OTHER TASKS.
TRANSFER MESSAGE	CQXFER	TRANSFER A MESSAGE TO A PORT IDENTIFIED BY THE CONNECTION-ID.
DEACTIVATE PORT	CQDACT	DEACTIVATE PORT. FURTHER MESSAGES ARE RETURNED TO THE SENDER.
LOSE	CQLOSE	LOSES A CONNECTION TO A PORT.

12-12



### WHAT IS MIP?

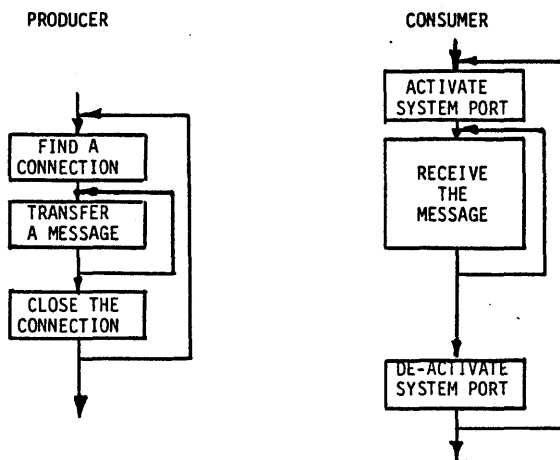
- THE MULTIBUS INTERPROCESSOR PROTOCOL (MIP) IS A SPECIFICATION FOR A SET OF MECHANISMS AND PROTOCOLS.
- PROVIDES AN EXCHANGE OF DATA AMONG TASKS EXECUTING ON VARIOUS SINGLE-BOARD COMPUTERS.



12-13

### MESSAGE MANAGER

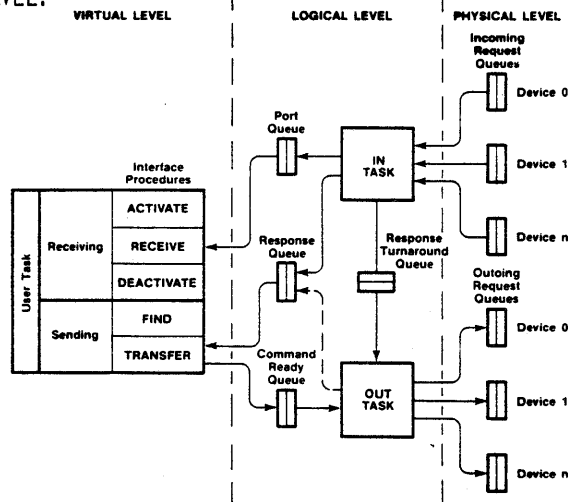
- FUNCTIONALITY FROM USER VIEWPOINT



12-14

### THREE-LEVEL INTERFACE STRUCTURE

- THE VIRTUAL LEVEL, BY WHICH USER TASKS INTERACT WITH THE MIP FACILITY.
- THE PHYSICAL LEVEL, BY WHICH THE MIP FACILITIES ON DIFFERENT DEVICES INTERACT WITH EACH OTHER.
- THE LOGICAL LEVEL, WHICH TRANSLATES BETWEEN THE VIRTUAL LEVEL AND THE PHYSICAL LEVEL.



12-15

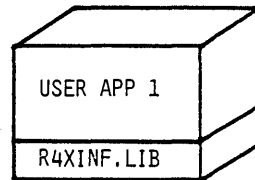
### SIGNALING

- IMM SOFTWARE SUPPORTS FOUR DIFFERENT SIGNALING MECHANISMS:
  - MULTIBUS INTERRUPTS
  - MEMORY MAPPED INTERRUPTS
  - IO-PORT MAPPED INTERRUPTS
  - POLLING
- A SOFTWARE HANDSHAKE THAT USES FLAGS IS ALSO EMPLOYED FOR MORE EFFICIENT THROUGH PUT.
  - E.G. A QUEUE KNOWN TO BE EMPTY IS NOT EXAMINED BY IN\$TASK.

12-16

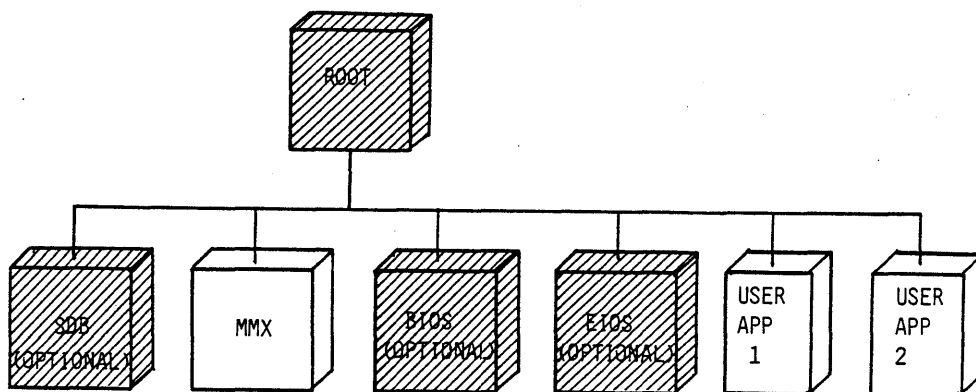
### THE USER APP JOB AND MMX 800/86

- A USER JOB MUST LINK TO A SET OF INTERFACE PROCEDURES
- THESE PROCEDURES ARE IN A LIB NAMED R4LINF.LIB OR R4CINF.LIB



12-17

### MMX 800 IN AN RMX86 ENVIRONMENT



- MMX SOFTWARE IS A JOB IN THE SYSTEM
- THERE IS AN MMX JOB IN EACH SYSTEM OF EACH DEVICE

12-18

#### STEPS IN BUILDING THE MMX JOB

1. BUILD A CONFIGURATION FILE NAMED R4CNFG.P86.
2. COMPILE AND PRODUCE AN OBJECT MODULE.
3. LINK THE MODULE TO A SET OF MMX LIBS.
4. LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS.
5. ENTER A "USER JOB" IN ICU86 FOR THE MMX JOB.

12-19

#### THE CONFIGURATION MODULE (R4CNFG.P86)

- THE CONFIGURATION MODULE IS A SET OF STRUCTURES.
- THESE STRUCTURES CONTAIN INFORMATION ABOUT THE CONFIGURATION AND REQUIREMENTS OF YOUR SYSTEM.
- THESE STRUCTURES FALL INTO THREE CATEGORIES:
  - SYSTEM LEVEL DECISIONS
  - DEVICE LEVEL DECISIONS
  - PORT LEVEL DECISIONS

12-20

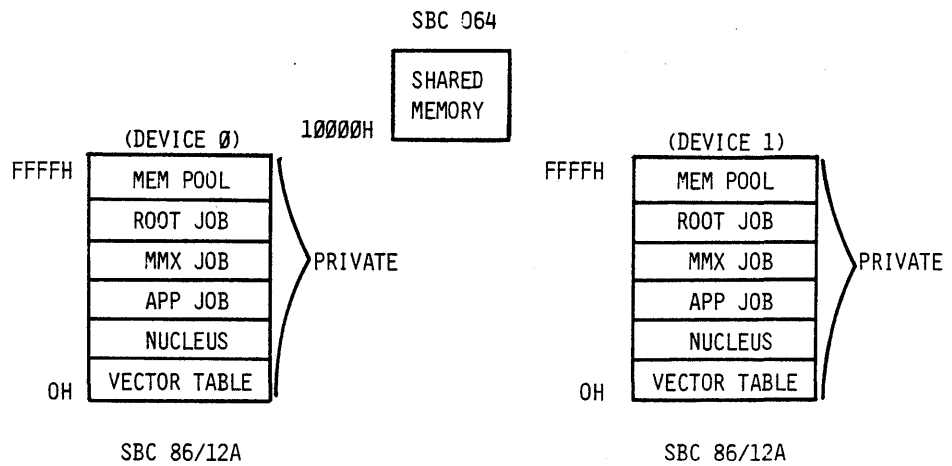
EXERCISE

- THE CONFIGURATION CHAPTER IN THE LMMX800 REFERENCE MANUAL DESCRIBES EACH OF THE STRUCTURES IN DETAIL.
- WITH THE AID OF YOUR INSTRUCTOR FILL IN THE BLANKS TO ACCOMMODATE THE TWO RMX86 DEVICE EXAMPLE.

12-21

AN EXAMPLE CONFIGURATION FOR TWO RMX86 DEVICES

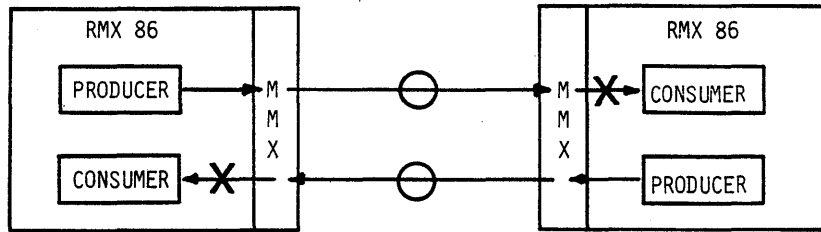
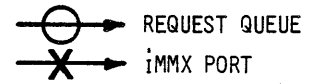
- MEMORY MAP FOR OUR EXAMPLE:



12-22

EXAMPLE CONFIGURATION FOR TWO RMX86 DEVICES

• SEND/RECEIVE ARCHITECTURE



12-23

EXERCISE (CONTINUED)

```
DECLARE CQDVCS BYTE PUBLIC
DATA (____);
```

```
DECLARE CQSKTS BYTE PUBLIC
DATA (____);
```

```
DECLARE CQPRTS BYTE PUBLIC
DATA (____);
```

```
DECLARE CQMDLY WORD PUBLIC
DATA (____);
```

```
DECLARE CQITWT WORD PUBLIC
DATA (____);
```

12-24

EXERCISE (CONTINUED)

```
DECLARE DSDT ( _____ ) DSD$ENTRY$TYPE
PUBLIC
DATA ( _____,
      _____,
      _____,
      _____,
      _____,
      _____ );
```

```
DECLARE LPT$ROM ( _____ ) LPT$ROM$ENTRY$TYPE
PUBLIC
DATA ( _____ );
```

```
DECLARE LPT$RAM ( _____ ) LPT$RAM$ENTRY$TYPE
PUBLIC;
```

EXERCISE (CONTINUED)

```
DECLARE DCM$ROM ( _____ ) DM$ROM$ENTRY$TYPE
PUBLIC
DATA ( _____,
      _____,
      _____,
      _____,
      _____,
      _____ );
```

```
DECLARE DCM$RAM ( _____ ) DM$RAM$ENTRY$TYPE
PUBLIC;
```

```
DECLARE CQSGLY WORD PUBLIC
DATA ( _____ );
```

```
DECLARE CQIDPD WORD PUBLIC
DATA ( _____ );
```

EXERCISE (CONTINUED)

DECLARE SFT (_____) SFT\$ENTRY\$TYPE

PUBLIC

DATA (_____,

_____,

_____,

00CEH,

0003H

00H,

0000H,

0000H,

000H,

0000H,

0000H);

DECLARE CQIDSS BYTE PUBLIC

DATA (____);

DECLARE IDST (____) IDS\$ENTRY\$TYPE

PUBLIC

DATA (____);

12-27

EXERCISE (CONTINUED)

DECLARE CQPLHS BYTE PUBLIC

DATA (____);

DECLARE PLHTBL (____) POOL\$ENTRY\$TYPE PUBLIC;

DECLARE CQBLKS BYTE PUBLIC

DATA (____);

DECLARE BLKTBL BLOCK\$ENTRY\$TYPE

PUBLIC

DATA (____,

____,

____);

12-28



OBJECTIVES:

EXECUTE A STUDENT MMX800/86 APPLICATION JOB IN AN RMX86 O.S. ENVIROMENT

INTRODUCE (MMX800/86) SYSTEM CALLS:

- CQ\$ACTV
- CQ\$FIND
- CQ\$XFER

CREATE SOURCE CODE:

- A SOURCE FILE NAMED START.P86
- A SOURCE FILE NAMED MMXLAB.P86

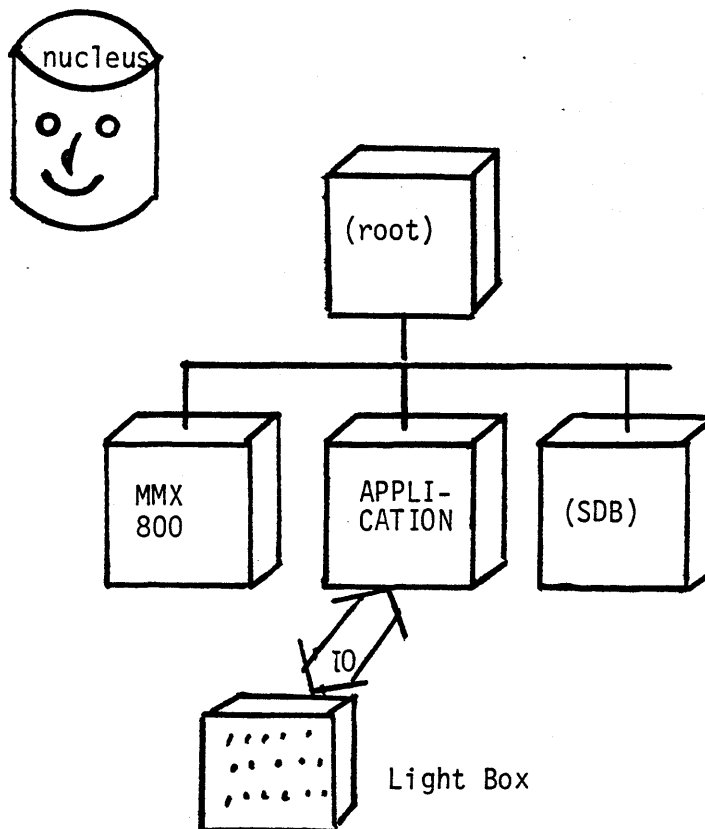
COMPILE (PLM86), LINK, AND LOCATE AN APPLICATION JOB, THAT WILL CALL UPON THE MMX800/86 TO COMMUNICATE WITH AN EXTERNAL DEVICE (THE TRUTH IS THAT WE WILL ONLY USE ONE DEVICE , AND THAT DEVICE WILL COMMUNICATE WITH ITSELF...)

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB7) AS THE LOGICAL NAME (:LAB:)

- AFILE "/TEAM NAME"/LAB7 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



- USE MMX 800/86 SYSTEM CALLS
- COMMUNICATION TO A LIGHT BOX
- SIMULATE INTERDEVICE COMMUNICATION

STEP2:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:MMXLAB.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:MMXLAB.P86

* THIS SOURCE FILE IS THE APPLICATION TASK THAT CONFORMS TO THE FOLLOWING FLOWCHART

-----  
ACTIVATE THE PRODUCER PORT  
-----

FIND THE CONSUMER CONNECTION  
-----

DO FOREVER  
-----

CREATE A SEGMENT  
-----

1ST BYTE IN SEGMENT = READ SWITCHES PORT 9CH  
-----

2ND BYTE IN SEGMENT = READ SWITCHES PORT 9DH  
-----

TRANSFER MESSAGE (SEGMENT) TO OTHER DEVICE  
-----

WAIT AND RECEIVE MESSAGE (SEGMENT) FROM OTHER DEVICE  
-----

LIGHTS PORT 9CH = 1ST BYTE IN MESSAGE RECEIVED  
-----

LIGHTS PORT 9DH = 2ND BYTE IN MESSAGE RECEIVED  
-----

* DELETE SELF **  
-----

STEP3:

- * ROOT JOBS ABSOLUTELY ADDRESS THE STARTING LOCATION OF THE STUDENT'S JOB CODE. THE ENTRY POINT MAY VARY IF INTERNAL PROCEDURES OR CHARACTER CONSTANTS ARE USED.  
FOR THIS REASON IT IS ADVISABLE TO CREATE AND LINK A START TASK TO THE REST OF THE APPLICATION CODE TO FIX THE ENTRY POINT'S OFFSET INTO THE CODE
- * THIS APPLICATION JOB WILL BE A FIRST LEVEL JOB, THIS REQUIRES THAT A TASK WITHIN THIS JOB MAKE A CALL TO RQ\$END\$INIT\$TASK TO RESUME THE ROOT TASK
- * IN ORDER TO DEBUG OUR CODE BEFORE IT "CRASHES" WE MAY WISH TO INVOKE THE 957 MONITOR AT THE START OF OUR JOB'S EXECUTION. THIS CAN EASILY BE ACCOMPLISHED BY PLACING A "CAUSE\$INTERRUPT(3)" INSTRUCTION AT THE BEGINNING OF OUR CODE (IN OUR START TASK).
- * WE WILL USE THE SAME START TASK THAT WE USED IN LAB TWO

-COPY /RII5.0/LAB2/START.P86 TO :LAB:START.P86

STEP4:

COMPILE THE SOURCE FILES (START.P86 AND MMXLAB.P86)

- PLM86 :LAB:START.P86
- PLM86 :LAB:MMXLAB.P86

- * IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING
- * IF COMPILATION IS SUCCESSFUL THE COMPILER WILL CREATE FOR EACH OF THE SOURCE FILES:
  - A LIST FILE NAMED ":LAB:(SOURCE).LST"
  - AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 :LAB:/START.OBJ,&
       :LAB:/MMXLAB.OBJ,&
       /MMX86/R4LINF.LIB,&
       /RMX5.0/DUTILS/EPIFL.LIB,&
       /RMX5.0/DUTILS/IPIFL.LIB,&
       /RMX5.0/DUTILS/RPIFL.LIB &
       TO :LAB:/JOB.LNK &
       NOMAP
```

LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS

```
LOC86 :LAB:/JOB.LNK &
      TO :LAB:/LABJOB &
      SC(3) SEGSIZE(STACK(0)) &
      ORDER(CLASSES(CODE,DATA,STACK)) &
      ADDRESSES(CLASSES(CODE(1040H))) &
      NOINITCODE &
      OC(PURGE)
```

AND FINALLY ADD THE LOCATED MODULE TO THE OTHER PRECONFIGURED PARTS OF OUR SYSTEM

```
LIB86
DELETE :LAB:RMX86(STARTMOD)
ADD :LAB:LABJOB to :LAB:RMX86
EXIT
```

- * :LAB:RMX86 IS A "GIVEN" FILE THAT CONTAINS:
  - A PRECONFIGURED NUCLEUS
  - A PRECONFIGURED MMX800/86
  - A PRECONFIGURED ROOT JOB
  - = A PRECONFIGURED SDB
- * THE STUDENT MAY "OPTIONALLY" USE A "GIVEN" SUBMIT FILE THAT WILL COMPILE , LINK , LOCATE AND ADD THE FINAL MODULE TO THE SYSTEM
  - SUBMIT :LAB:JOB.CSD

***** LAB SEVEN (MMX800/86 SYSTEM) PART A *****

STEP5:

* A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB7/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NESESARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

***** LAB SEVEN (MMX800/86 SYSTEM) PART B *****

OBJECTIVES:

EXECUTE A STUDENT MMX800/86 APPLICATION JOB IN AN RMX86 O.S. ENVIRONMENT

LEARN HOW TO BUILD A CONFIGURATION FILE , AND AN MMX86 JOB TO REPLACE THE ONE GIVEN IN THE PREVIOUS LAB (LAB SEVEN PART A)

CREATE SOURCE CODE:

- A SOURCE FILE NAMED R4CNFG.P86

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB7) AS THE LOGICAL NAME (:LAB:)

- AFILE "/TEAM NAME"/LAB7 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *

STEP2:

MODIFY A SOURCE FILE (PARTIALLY SUPPLIED FOR YOU) NAMED :LAB:R4CNFG.P86 WITH THE "ALTER" TEXT EDITOR

- ALTER :LAB:R4CNFG.P86

* THIS SOURCE FILE IS THE MMX86 CONFIGURATION FILE

***** LAB SEVEN (MMX800/86 SYSTEM) PART B *****

STEP3:

COMPILE THE SOURCE FILE (R4CNFG.P86)

- PLM86 :LAB:R4CNFG.P86

* IF ANY ERRORS OCCURRED DURING COMPILATION , YOU MUST FIX AND RECOMPILE BEFORE CONTINUING

* ELSE THE FOLLOWING FILES WILL BE CREATED

- A LIST FILE NAMED ":LAB:(SOURCE).LST"  
- AN OBJECT FILE NAMED ":LAB:(SOURCE).OBJ"

LINK THE OBJECTS WITH THE INTERFACE LIBRARIES NEEDED (LARGE)

```
LINK86 /MMX86/R4DRVR.LIB(MBEGIN) ,&  
       :LAB:R4CNFG.OBJ,&  
       /MMX86/R4DRVR.LIB      ,&  
       /MMX86/R4XMGR.LIB      ,&  
       /MMX86/R4957P.LIB      ,&  
       /MMX86/R4PMM.LIB       ,&  
       /MMX86/R4UTIL.LIB      ,&  
       /RMX5.0/DUTILS/RPIFC.LIB &  
       TO :LAB:JOB.LNK &  
       NOMAP NOTYPE
```

LOCATE THE LINKED MODULE TO AN ABSOLUTE ADDRESS

```
LOC86  :LAB:JOB.LNK &  
       TO :LAB:MMXJOB &  
       SC(3) SEGSIZE(STACK(0)) &  
       ORDER(CLASSES(CODE,DATA,STACK)) &  
       ADDRESSES(CLASSES(CODE(3000H))) &  
       NOINITCODE &  
       OC(PURGE)
```

AND FINALLY ADD THE LOCATED MODULE TO THE OTHER PRECONFIGURED PARTS OF OUR SYSTEM

```
.IB86  
DELETE :LAB:RMX86(MBEGIN)  
ADD    :LAB:MMXJOB to :LAB:RMX86  
EXIT
```

* THE STUDENT MAY "OPTIONALLY" USE A "GIVEN" SUBMIT FILE THAT WILL COMPILE , LINK , LOCATE AND ADD THE FINAL MODULE TO THE SYSTEM

- SUBMIT :LAB:MMXJOB.CSD

***** LAB SEVEN (MMX800/86 SYSTEM) PART B *****

STEP5:

* A LOCATE MAP AND SOURCE LISTING WILL HELP YOU DEBUG YOUR CODE IF PROBLEMS ARISE . THIS IS THE TIME TO GET THE LISTINGS OUT

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB7/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

* THE 957 DEBUG MONITOR IS PRESENT AND CAN BE USED TO DEBUG YOUR CODE IF NESSESARY. PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL



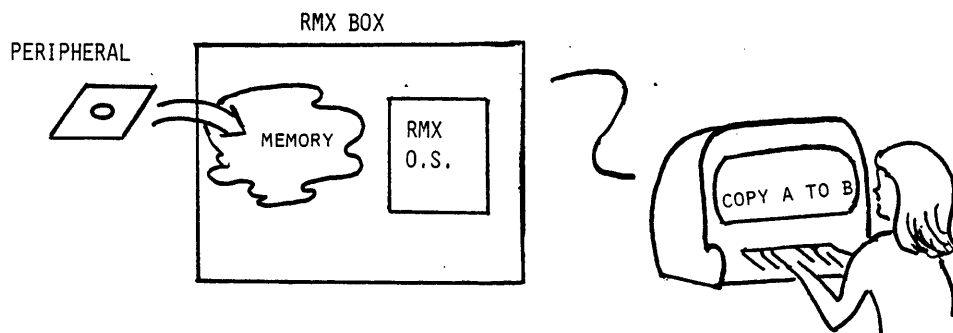
## **CHAPTER 13**

### **HUMAN INTERFACE**

- COMMANDS
- SYSTEM CALLS
- THE RESIDENT USER
- DEFINITION FILES



## HUMAN INTERFACE COMMANDS



- COMMANDS ARE PROGRAMS (COPY, RENAME.....ETC.)
- THESE PROGRAMS ARE JOBS CREATED BY THE H.I.
- LOADED BY THE HUMAN INTERFACE UPON OPERATOR'S REQUEST.

13-1

## INTEL PROVIDES A SET OF COMMANDS

ATTACHDEVICE	DIR	JOB DELETE
ATTACHFILE	DISKVERIFY	LOCK
BACKUP	DOWNCOPY	PERMIT
COPY	FORMAT	RENAME
CREATEDIR	INITSTATUS	RESTORE
DATE		SUBMIT
DEBUG		SUPER
DELETE		TIME
DETACHDEVICE		UPCOPY
DETACHFILE		VERSION

- A COMPLETE DESCRIPTION OF THESE COMMANDS ARE FOUND IN THE iRMX 86 OPERATOR'S MANUAL.

13-2

## SYSTEM CALLS

- A SET OF CALLS ARE AVAILABLE TO AID IN THE CREATION OF CUSTOM COMMANDS.

C\$GET\$INPUT\$CONNECTION	C\$GET\$COMMAND\$NAME
C\$GET\$OUTPUT\$CONNECTION	C\$FORMAT\$EXCEPTION
C\$GET\$CHAR	C\$SEND\$CO\$RESPONSE
C\$GET\$INPUT\$PATHNAME	C\$SEND\$EO\$RESPONSE
C\$GET\$PARAMETER	C\$CREATE\$COMMAND\$CONNECTION
C\$GET\$OUTPUT\$PATHNAME	C\$DELETE\$COMMAND\$CONNECTION
C\$SET\$PARSE\$BUFFER	C\$SEND\$COMMAND

- A COMPLETE DESCRIPTION OF THESE H.I. SYSTEM CALLS ARE FOUND IN THE iRMX86 HUMAN INTERFACE REFERENCE MANUAL.

13-3

## HUMAN INTERFACE INITIAL PROGRAM

- INTEL PROVIDES THE DEFAULT RESIDENT INITIAL PROGRAM.
- THIS PROGRAM IS A STANDARD COMMAND LINE INTERPRETER.
- YOU MAY PROVIDE YOUR OWN INITIAL PROGRAM DURING CONFIGURATION.

13-4

#### THE RESIDENT USER

- THE RESIDENT USER MAY BE:
  - THE ONLY USER IN THE SYSTEM
  - THE FIRST USER IN A MULTI-ACCESS SYSTEM
- RESIDENT USER IS DEFINED DURING CONFIGURATION BY:
  - TERMINAL DEVICE NAME
  - MAX TASK PRIORITY
  - USER ID
  - INITIAL PROGRAM
  - DEFAULT PREFIX
  - POOL SIZE

13-5

#### MULTI-ACCESS USER DEFINITION

- OTHER USERS ARE DEFINED IN FILES THAT DESCRIBE THE OPERATOR AND HIS TERMINAL.
- THE PATHNAMES FOR THESE FILES ARE:
  - CONFIG/TERMINALS (TERMINAL DEFINITION FILE)
  - CONFIG/USER/ID# (USER DEFINITION FILE)
- ID# IS THE ACTUAL ID NUMBER FOR THAT PARTICULAR USER
  - E.G. CONFIG/USER/0082
- IF THESE FILES DO NOT EXIST, THE HUMAN INTERFACE WILL COME UP IN SINGLE-ACCESS MODE.

13-6

### TERMINAL DEFINITION FILE

- # OF TERMINALS
- DEVICE - NAME
- USER - ID
- PARTITION - SIZE
- MAX - PRIORITY
- UNIT - PATHNAME

:SD:CONFIG/TERMINALS

#### EXAMPLE OF FOUR TERMINALS

```
4
T1,42,90 <CR>
T0,65535,80,210,:SD:SPECLI <CR>
T3,85,64,220 <CR>
T2,85,64,225 <CR>
```

13-7

### USER DEFINITION FILE

- USER-ID
- PASSWORD
- DEFAULT-PARTITION
- MAX-PARTITION
- MAX-PRIORITY
- DEFAULT-PREFIX
- INIT-PATHNAME

:SD:CONFIG/USER/65535

#### EXAMPLE OF ONE USER

```
65535,PASS, 64 <CR>
120 <CR>
190 <CR>
:SD:USER/65535 <CR>
```

- A COMPLETE DESCRIPTION OF THESE FILES IS FOUND IN THE IRMX 86 CONFIGURATION GUIDE.

13-8

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

OBJECTIVES:

THE STUDENT WILL USE THE INTERACTIVE CONFIGURATION UTILITY (ICU)  
TO CREATE A SINGLE ACCESS SYSTEM

THIS SYSTEM WILL CONTAIN

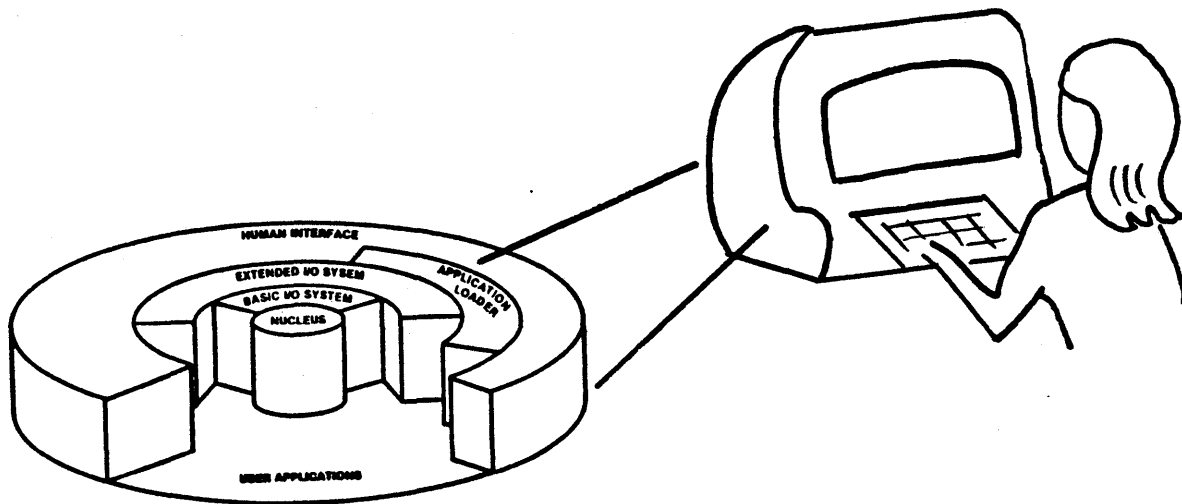
- A NUCLEUS JOB
- A BIOS JOB
- A EIOS JOB
- A LOADER JOB
- A HUMAN INTERFACE JOB

STEP1:

USE THE ATTACH\$FILE COMMAND TO ATTACH THE DIRECTORY NAMED ("/TEAM NAME"/LAB8)  
AS THE LOGICAL NAME (:LAB:)

- AFILE /"TEAM NAME"/LAB8 AS :LAB:

* FOR THE REST OF THIS LAB WE WILL USE THIS LOGICAL NAME *



***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

STEP2:

THE (ICU) IS INVOKED BY TYPING THE FOLLOWING  
-ICU86 :LAB:ICU.DEF

- * WHERE ICU.DEF IS THE NAME OF THE FILE WE HAVE CHOSEN TO CONTAIN THE INFORMATION NEEDED TO CONFIGURE OUR O.S.

WHEN THE ICU SIGN ON ENTER THE COMMAND C , TO MODIFY THE SCREENS

STEP3:

- * TRY FILLING THE SCREENS WITHOUT LOOKING AT THESE FIRST, THEN MATCH YOUR ENTRIES TO THE ONES GIVEN HERE

- * IF YOU DO NOT UNDERSTAND AN ENTRY TYPE ?  
- E.G. OSP? <cr>

- * SOME OF THE SCREENS NEEDED FOR OUR LAB EIGHT CONFIGURATION FOLLOW

Type : RAM = 0104H, 24FFH  
Type : RAM = 26B6H, F7FFH

Human Interface

(ICL) Initial Command Line Size (0-OFFFFH)	0100H
(CNM) Command Name Length (0-255)	0030H
(SYS) System Directory (1-45 characters)	
	:SD:SYSTEM
(DRP) Default Resident Initial Program (Yes/No)	Yes
(RIP) Resident Initial Program (1-45 characters)	
	Default
(CDN) Configuration Device Name (1-14 chars)	:SD:
(PMI) Human Interface Pool Minimum (0-OFFFFH)	0260H
(PMA) Human Interface Pool Maximum (0-OFFFFH)	FFFFH
(HIR) Human Interface in ROM (Yes/No)	No

HI Jobs

(MIN) Jobs Minimum Memory (0-OFFFFH pages)	0200H
(MAX) Jobs Maximum Memory (0-OFFFFH pages)	0000H
(NPX) Numeric Processor Extension Used (Yes/No)	Yes

Resident User

(TDN) Terminal Device Name (1-12 Characters)	TO
(MTP) Maximum Task Priority (0-OFFH)	00A0H
(UID) User ID Number (0-OFFFFH)	0000H
(MIN) Minimum Memory Required (0-OFFFFH)	1000H
(MAX) Maximum Memory Required (0-OFFFFH)	FFFFH
(IPP) Initial-Program Pathname (RESIDENT/1-45 Characters)	
	RESIDENT
(DEF) Default Directory (1-45 characters)	
	:SD:USER



Prefixes

Prefix : 1-45 characters  
 Prefix : :\$:  
 Prefix : :PROG:  
 Prefix : :SYSTEM:  
 Prefix : :LANG:  
 Prefix :

HI Logical Names

Logical Name : logical_name,path_name  
 (1-12 Chars ,1-45 Chars)

Logical Name : WORK, :SD:USER/WORK  
 Logical Name : LANG, :SD:LANG  
 Logical Name : PROG, :SD:USER/PROG

EIOS

(ASC) All Sys Calls in EIOS	Req
(ABR) Automatic Boot Device Recognition (Yes/No)	Yes
(DLN) Default System Device Logical Name (1-12 characters)	SD
(DPN) Default System Device Physical Name (1-12 characters)	IWO
(DFD) Default System Device File Driver (Phys/Str/Named)	Named
(DO) Default System Device Owners ID (0-OFFFFH)	0000H
(EBS) Internal Buffer Size (0-OFFFFh)	0400H
(DDS) Default IO Job Directory Size (5-OFF0h)	0020H
(ITP) Internal EIOS Task's Priorities (0-OFFH)	0083H
(PMI) EIOS Pool Minimum (0-OFFFFH)	0180H
(PMA) EIOS Pool Maximum (0-OFFFFH)	0180H
(EIR) Extended I/O System in ROM (Yes/No)	No

Logical Names

Logical Name : logical_name,device_name,file_driver,owners-id  
 (1-12 Chars ,1-14 Chars ,Physical/Stream/Named, 0-OFFFFH)

Logical Name : BB, BB, Physical, 0000H  
 Logical Name : STREAM, STREAM, Stream, 0000H  
 Logical Name : FDO, WFDDO, Named, FFFFH

Intel Terminal Driver

(IIL) Input Interrupt Level (Encoded)	0068H
(OIL) Output Interrupt Level (Encoded)	0078H
(UDP) USART Data Port (0-OFFFFH)	00D8H
(USP) USART Status Port (0-OFFFFH)	00DAH
(IRP) 8253 Input Rate Port (0-OFFFFH)	00D4H
(ICP) 8253 Input Control Port (0-OFFFFH)	00D6H
(IRC) 8253 Input Counter Number (0-2)	0002H
(IRM) Input Rate Maximum (0-OFFFFFFFFH)	00012C00H
(ORP) 8253 Output Rate Port (0-OFFFFH)	0000H
(OCP) 8253 Output Control Port (0-OFFFFH)	0000H
(ORC) 8253 Output Counter Number (0-2)	0000H
(ORM) Output Rate Maximum (0-OFFFFFFFFH)	00000000H

Intel iSBC 215/218 Driver

(IL) Interrupt Level (Encoded Level)	0058H
(ITP) Interrupt Task Priority (0-OFFH)	0082H
(WIP) Wakeup I/O Port (0-OFFFFH)	0100H

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

```

Intel iSBC 215/218 Unit Information
(NAM) Unit Info Name (1-17 Chars)      uinfo_215gen
(MR) Maximum Retries (0-OFFFFFH)      0009H
(CS) Cylinder Size (0-OFFFFFH)        0000H
(NC) Number of Cylinders (0-OFFFFFH)   0001H
(NFH) Number of Fixed Platters/Disk (0-OFFH) 0001H
(NRH) Number of Remove Platters/Disk (0-OFFH) 0000H
(NS) Number of Sectors/Track (0-OFFFFFH) 000CH
(NAC) Number of Aux. Cylinders (0-OFFH) 0001H
(SSN) Starting Sector Number (0-OFFFFFFFH) 00000000H
(BTI) Bad Track Information (Yes/No)    Yes

```

```

Intel iSBC 215/218 Unit Information
(NAM) Unit Info Name (1-17 Chars)      uinfo_215w
(MR) Maximum Retries (0-OFFFFFH)      0009H
(CS) Cylinder Size (0-OFFFFFH)        0000H
(NC) Number of Cylinders (0-OFFFFFH)   0208H
(NFH) Number of Fixed Platters/Disk (0-OFFH) 0005H
(NRH) Number of Remove Platters/Disk (0-OFFH) 0000H
(NS) Number of Sectors/Track (0-OFFFFFH) 000CH
(NAC) Number of Aux. Cylinders (0-OFFH) 000AH
(SSN) Starting Sector Number (0-OFFFFFFFH) 00000000H
(BTI) Bad Track Information (Yes/No)    Yes

```

```

Intel iSBC 215/218 Unit Information
(NAM) Unit Info Name (1-17 Chars)      uinfo_215pt
(MR) Maximum Retries (0-OFFFFFH)      0009H
(CS) Cylinder Size (0-OFFFFFH)        0000H
(NC) Number of Cylinders (0-OFFFFFH)   01D2H
(NFH) Number of Fixed Platters/Disk (0-OFFH) 0003H
(NRH) Number of Remove Platters/Disk (0-OFFH) 0000H
(NS) Number of Sectors/Track (0-OFFFFFH) 000CH
(NAC) Number of Aux. Cylinders (0-OFFH) 0006H
(SSN) Starting Sector Number (0-OFFFFFFFH) 00000000H
(BTI) Bad Track Information (Yes/No)    Yes

```

```

Intel iSBC 215/218 Unit Information
(NAM) Unit Info Name (1-17 Chars)      uinfo_215f
(MR) Maximum Retries (0-OFFFFFH)      0009H
(CS) Cylinder Size (0-OFFFFFH)        0000H
(NC) Number of Cylinders (0-OFFFFFH)   004DH
(NFH) Number of Fixed Platters/Disk (0-OFFH) 0000H
(NRH) Number of Remove Platters/Disk (0-OFFH) 0001H
(NS) Number of Sectors/Track (0-OFFFFFH) 001AH
(NAC) Number of Aux. Cylinders (0-OFFH) 0000H
(SSN) Starting Sector Number (0-OFFFFFFFH) 00000000H
(BTI) Bad Track Information (Yes/No)    Yes

```

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

Intel iSBC 215/218 Unit Information

(NAM) Unit Info Name (1-17 Chars)	uinfo_215fd
(MR) Maximum Retries (0-OFFFFH)	0009H
(CS) Cylinder Size (0-OFFFFH)	0000H
(NC) Number of Cylinders (0-OFFFFH)	004DH
(NFH) Number of Fixed Platters/Disk (0-OFFH)	0000H
(NRH) Number of Remove Platters/Disk (0-OFFH)	0002H
(NS) Number of Sectors/Track (0-OFFFFH)	001AH
(NAC) Number of Aux. Cylinders (0-OFFH)	0000H
(SSN) Starting Sector Number (0-OFFFFFFFFH)	00000000H
(BTI) Bad Track Information (Yes/No)	Yes

Intel iSBC 215/iSBX 218 Device-Unit Information

(NAM) Device-Unit Name (1-13 chars)	W0
(PFD) Physical File Driver Required (Yes/No)	Yes
(NFD) Named File Driver Required (Yes/No)	Yes
(SDD) Single or Double Density Disks (Single/Double)	Single
(SDS) Single or Double Sided Disks (Single/Double)	Single
(EFI) 8 or 5 Inch Disks (8/5)	8
(GRA) Granularity (0-OFFFFH)	0400H
(DSZ) Device Size (0-OFFFFFFFFH)	00000400H
(UN) Unit Number on this Device (0-OFFH)	0000H
(UIN) Unit Info Name (1-17 Chars)	uinfo_215gen
(UDT) Update Timeout (0-OFFFFH)	0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFH)	0006H
(FUP) Fixed Update (True/False)	True
(MB) Max Buffers (0-OFFH)	00FFH

Intel iSBC 215/iSBX 218 Device-Unit Information

(NAM) Device-Unit Name (1-13 chars)	IW0
(PFD) Physical File Driver Required (Yes/No)	Yes
(NFD) Named File Driver Required (Yes/No)	Yes
(SDD) Single or Double Density Disks (Single/Double)	Single
(SDS) Single or Double Sided Disks (Single/Double)	Single
(EFI) 8 or 5 Inch Disks (8/5)	8
(GRA) Granularity (0-OFFFFH)	0400H
(DSZ) Device Size (0-OFFFFFFFFH)	01DE2000H
(UN) Unit Number on this Device (0-OFFH)	0000H
(UIN) Unit Info Name (1-17 Chars)	uinfo_215w
(UDT) Update Timeout (0-OFFFFH)	0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFH)	0006H
(FUP) Fixed Update (True/False)	True
(MB) Max Buffers (0-OFFH)	00FFH

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

```

Intel iSBC 215/iSBX 218 Device-Unit Information
(NAM) Device-Unit Name (1-13 chars)          PWO
(PFD) Physical File Driver Required (Yes/No)  Yes
(NFD) Named File Driver Required (Yes/No)    Yes
(SDD) Single or Double Density Disks (Single/Double) Single
(SDS) Single or Double Sided Disks (Single/Double) Single
(EFI) 8 or 5 Inch Disks (8/5)                8
(GRA) Granularity (0-OFFFFFH)                0400H
(DSZ) Device Size (0-OFFFFFFFFFH)            0102C000H
(UN) Unit Number on this Device (0-OFFH)     0000H
(UIN) Unit Info Name (1-17 Chars)            uinfo_215pt
(UDT) Update Timeout (0-OFFFFFH)            0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFFH) 0006H
(FUP) Fixed Update (True/False)              True
(MB) Max Buffers (0-OFFH)                    00FFH

```

```

Intel iSBC 215/iSBX 218 Device-Unit Information
(NAM) Device-Unit Name (1-13 chars)          WFO
(PFD) Physical File Driver Required (Yes/No)  Yes
(NFD) Named File Driver Required (Yes/No)    Yes
(SDD) Single or Double Density Disks (Single/Double) Single
(SDS) Single or Double Sided Disks (Single/Double) Single
(EFI) 8 or 5 Inch Disks (8/5)                8
(GRA) Granularity (0-OFFFFFH)                0080H
(DSZ) Device Size (0-OFFFFFFFFFH)            0003E900H
(UN) Unit Number on this Device (0-OFFH)     0008H
(UIN) Unit Info Name (1-17 Chars)            uinfo_215f
(UDT) Update Timeout (0-OFFFFFH)            0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFFH) 0006H
(FUP) Fixed Update (True/False)              True
(MB) Max Buffers (0-OFFH)                    00FFH

```

```

Intel iSBC 215/iSBX 218 Device-Unit Information
(NAM) Device-Unit Name (1-13 chars)          WFD0
(PFD) Physical File Driver Required (Yes/No)  Yes
(NFD) Named File Driver Required (Yes/No)    Yes
(SDD) Single or Double Density Disks (Single/Double) Double
(SDS) Single or Double Sided Disks (Single/Double) Single
(EFI) 8 or 5 Inch Disks (8/5)                8
(GRA) Granularity (0-OFFFFFH)                0100H
(DSZ) Device Size (0-OFFFFFFFFFH)            0007C500H
(UN) Unit Number on this Device (0-OFFH)     0008H
(UIN) Unit Info Name (1-17 Chars)            uinfo_215f
(UDT) Update Timeout (0-OFFFFFH)            0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFFH) 0006H
(FUP) Fixed Update (True/False)              True
(MB) Max Buffers (0-OFFH)                    00FFH

```

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

```

ntel iSBC 215/iSBX 218 Device-Unit Information
(NAM) Device-Unit Name (1-13 chars)           WFDDO
(PFD) Physical File Driver Required (Yes/No)   Yes
(NFD) Named File Driver Required (Yes/No)     Yes
(SDD) Single or Double Density Disks (Single/Double) Double
(SDS) Single or Double Sided Disks (Single/Double) Double
(EFI) 8 or 5 Inch Disks (8/5)                8
(GRA) Granularity (0-OFFFFH)                 0100H
(DSZ) Device Size (0-OFFFFFFFH)              000F9700H
(UN) Unit Number on this Device (0-OFFH)      0008H
(UIN) Unit Info Name (1-17 Chars)            uinfo_215fd
(UDT) Update Timeout (0-OFFFFH)              0064H
(NB) Number of Buffers (nonrandom = 0/rand = 1-OFFFFH) 0006H
(FUP) Fixed Update (True/False)              True
(MB) Max Buffers (0-OFFH)                    00FFH

```

***** THIS JOB SUPPLIES THE INTERRUPT FOR THE MONITOR *****

```

User Jobs
(ODS) Object Directory Size (0-OFFOH)         000AH
(PMI) Pool Minimum (20H - OFFFFH)            0030H
(PMA) Pool Maximum (20H - OFFFFH)            FFFFH
(MOB) Maximum Objects (1 - OFFFFH)           FFFFH
(MTK) Maximum Tasks (1 - OFFFFH)             FFFFH
(MPR) Maximum Priority (0 - OFFH)              0000H
(AEH) Address of Exception Handler (CS:IP)    0000H:0000H
(EM) Exception Mode (Never/Prog/Environ/All) Never
(PV) Parameter Validation (Yes/No)            Yes
(TP) Task Priority (0-OFFH)                   0000H
(TSA) Task Start Address (CS:IP)              0080H:0002H
(DSB) Data Segment Base (0-OFFFFH)           0000H
(SSA) Stack Segment Address (SS:SP)           0000H:0000H
(SS) Stack Size (0-OFFFFH)                   0200H
(NPX) Numeric Processor Extension Used (Yes/No) No

```

Includes and Libraries

```

Path Name (1-45 Characters)
(UDF) UDI Includes and Libs
      /RMX5.0/DUDI/
(HIF) Human Interface Includes and Libs
      /RMX5.0/DINCLSLIBS/
(EIF) Extended I/O System Includes and Libs
      /RMX5.0/DINCLSLIBS/
(ALF) Application Loader Includes and Libs
      /RMX5.0/DLOADER/
(BIF) Basic I/O System Includes and Libs
      /RMX5.0/DINCLSLIBS/
(THF) Terminal Handler and Debugger Includes and Libs
      /RMX5.0/DDEBTH/
(NUF) Nucleus and Root Job Includes and Libs
      /RMX5.0/DNUCLUS/
(ILF) Interface Libraries
      /RMX5.0/DUTILS/
(CAF) Crash Analyzer Includes and Libs
      /RMX5.0/DUDI/
(DTF) Development Tools Path Names
      /LANG/

```

***** LAB EIGHT (H.I. CONFIG THROUGH ICU) *****

STEP4:

AFTER YOU ENTER ALL OF THE SCREENS ENTER G TO GENERATE  
EXIT THE ICU

SUBMIT THE ICU.CSD FILE TO GENERATE YOUR SYSTEM

-SUBMIT :LAB:ICU.CSD

STEP5:

YOU MUST NOW ADD THE SDB TO THE SYSTEM,  
USING THE LIB86 UTILITY

-LIB86  
DELETE :LAB:RMX86(INT3TASKMOD)  
ADD /DINT3/INT3JOB to :LAB:RMX86  
EXIT

STEP6:

YOU ARE NOW READY TO "BOOT" YOUR NEWLY CREATED SYSTEM

IF YOUR EXECUTION VEHICLE IS THE SAME AS THE DEVELOPMENT STATION  
THEN:

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /"TEAM NAME"/LAB8/RMX86

IF YOUR EXECUTION VEHICLE DIFFERS FROM THE DEVELOPMENT STATION  
THEN:

-COPY THE NEWLY CREATED BOOTABLE SYSTEM INTO A FLOPPY.  
( COPY :LAB:RMX86 OVER :FDO:RMX86 )

-PRESS INTERRUPT ON EXECUTION VEHICLE FRONT PANEL

-BOOT THE NEW SYSTEM

.B /RMX86

## **CHAPTER 14**

# **UNIVERSAL DEVELOPMENT INTERFACE**

- SPECIFICATIONS
- LIBRARIES
- DEVELOPMENT PROCESS
- SYSTEM CALLS





## WHAT/WHY UDI?

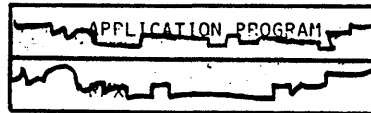
### UNIVERSAL DEVELOPMENT INTERFACE

UDI IS A SPECIFICATION OF A SET OF PROCEDURE CALLS THAT ARE USED TO REQUEST OPERATING SYSTEM FUNCTIONS.

FUNCTIONS ARE IMPLEMENTED BY MODULES THAT TRANSLATE FROM THE UDI STANDARD TO THE ACTUAL OPERATING SYSTEM CALLS.

EACH INTEL OPERATING SYSTEM FOR THE IAPX 86,88 FAMILY PROVIDES A UNIVERSAL DEVELOPMENT INTERFACE OR A SUBSET THEREOF.

UDI



14-1

## UDI FUNCTIONS

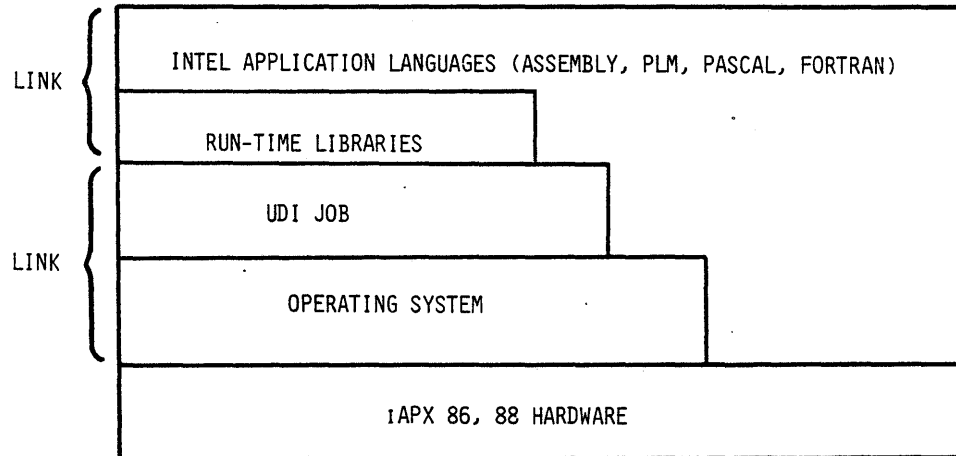
THE KINDS OF FUNCTIONS THAT ARE AVAILABLE THROUGH UDI PROCEDURE CALLS INCLUDE:

- CREATING AND BREAKING CONNECTIONS TO DATA FILES
- OPENING, READING, SEEKING, WRITING, AND CLOSING DATA FILES
- CONTROLLING PROGRAM EXECUTION
- CONTROLLING MEMORY ALLOCATIONS
- HANDLING SYSTEM EXCEPTION CONDITIONS
- CONTROLLING THE PROCESSING OF CONSOLE INPUT & PARSING COMMAND LINES
- FETCHING THE CURRENT DATE AND TIME

14-2

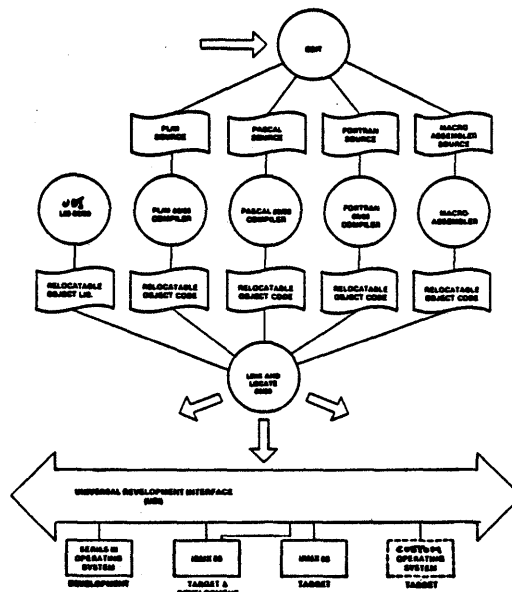
## LIBRARIES

THE iRMX 86 OPERATING SYSTEM SUPPORTS UDI BY PROVIDING UDI INTERFACE LIBRARIES.



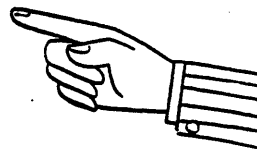
14-3

## SOFTWARE DEVELOPMENT PROCESS



14-4

## CAREFUL!



- YOU CAN MAKE OPERATING SYSTEM CALLS DIRECTLY FROM YOUR APPLICATION (SUBJECT TO SYSTEM AND LANGUAGE RESTRICTIONS).
- IF YOU DO SO, HOWEVER, YOU MAY NOT BE ABLE TO TRANSPORT YOUR APPLICATION TO ANOTHER OPERATING ENVIRONMENT.
- ADHERING TO UDI SPECIFICATIONS ENSURES THAT YOUR APPLICATION REMAINS OPERATING-SYSTEM INDEPENDENT AND TRANSPORTABLE.

14-5

### THE iRMX OPERATING SYSTEM CONSISTS OF A NUMBER OF SUBSYSTEMS

RMX LAYERS	DESCRIPTION
NUCLEUS	THE CORE OF THE iRMX 86 OPERATING SYSTEM AND IS REQUIRED FOR EVERY APPLICATION SYSTEM
TERMINAL HANDLER	PROVIDES A REAL-TIME INTERFACE BETWEEN YOUR TERMINAL AND OTHER SOFTWARE.
BASIC I/O SYSTEM	PROVIDES ASYNCHRONOUS FILE ACCESS CAPABILITIES
EXTENDED I/O SYSTEM	PROVIDES HIGH LEVEL, SYNCHRONOUS FILE ACCESS CAPABILITIES
APPLICATION LOADER	PROVIDES THE CAPABILITY TO LOAD OBJECT FILES INTO MEMORY FROM DISK
HUMAN INTERFACE	PROVIDES AN INTERACTIVE INTERFACE BETWEEN A USER AND SOFTWARE

14-6

UDI CALLS AND iRMX 86 SYSTEM CALLS

UDI CALLS	iRMX 86 SYSTEM CALLS	SUBSYSTEMS
DQ\$ALLOCATE	RQ\$CREATE\$SEGMENT	NUCLEUS
DQ\$ATTACH	RQ\$\$ATTACH\$FILE	EXTENDED I/O SYSTEM
DQ\$CHANGE\$EXTENSION	(NONE)	(NONE)
DQ\$CLOSE	RQ\$\$CLOSE	EXTENDED I/O SYSTEM
DQ\$CREATE	RQ\$\$CREATE\$FILE RQ\$\$GET\$FILE\$STATUS	EXTENDED I/O SYSTEM
DQ\$DECODE\$EXCEPTION	RQ\$C\$FORMAT\$EXCEPTION	HUMAN INTERFACE
DQ\$DELETE	RQ\$DELETE\$FILE	EXTENDED I/O SYSTEM
DQ\$DETACH	RQ\$\$DELETE\$CONNECTION RQ\$\$CLOSE	EXTENDED I/O SYSTEM
DQ\$FREE	RQ\$DELETE\$SEGMENT	NUCLEUS

UDI CALLS AND iRMX 86 SYSTEM CALLS

UDI CALLS	iRMX 86 SYSTEM CALLS	SUBSYSTEMS
DQ\$GET\$ARGUMENT	RQ\$C\$GET\$CHAR	HUMAN INTERFACE
DQ\$GET\$CONNECTION\$STATUS	RQ\$\$GET\$CONNECTION\$STATUS RQ\$A\$GET\$FILE\$STATUS	EXTENDED I/O SYSTEM BASIC I/O SYSTEM
DQ\$GET\$EXCEPTION\$HANDLER	RQ\$GET\$EXCEPTION\$HANDLER	NUCLEUS
DQ\$GET\$SIZE	RQ\$GET\$SIZE	NUCLEUS
DQ\$GET\$SYSTEM\$ID	(NONE)	(NONE)
DQ\$GET\$TIME	RQ\$GET\$TIME	BASIC I/O SYSTEM
DQ\$OPEN	RQ\$\$OPEN	EXTENDED I/O SYSTEM
DQ\$OVERLAY	RQ\$\$OVERLAY	APPLICATION LOADER
DQ\$READ	RQ\$\$READ\$MOVE	EXTENDED I/O SYSTEM
DQ\$RENAME	RQ\$\$RENAME\$FILE	EXTENDED I/O SYSTEM
DQ\$SEEK	RQ\$\$SEEK	EXTENDED I/O SYSTEM
DQ\$SPECIAL	RQ\$\$SPECIAL	EXTENDED I/O SYSTEM
DQ\$SWITCH\$BUFFER	RQ\$SETS\$PARSE\$BUFFER	HUMAN INTERFACE
DQ\$TRAP\$EXCEPTION	RQ\$\$TRUNCATE\$FILE	EXTENDED I/O SYSTEM
DQ\$WRITE	RQ\$\$WRITE\$MOVE	EXTENDED I/O SYSTEM

### EXAMPLE CALL TO REQUEST MEMORY

```
DECLARE STATUS . WORD;  
DECLARE ARRAY_BASE SELECTOR;
```

```
.  
.  
.
```

```
ARRAY_BASE = DQ$ALLOCATE(128, @STATUS);
```

- IF THE REQUEST FAILS THEN  
    ARRAY_BASE EQUALS 0FFFFH  
    AND STATUS = E\$MEM

14-9

### ERROR REPORTING

UDI PROCEDURES RETURN A CONDITION CODE THAT INDICATES THE RESULTS OF EXECUTING A UDI PROCEDURE.

- YOU MUST CHECK THE CONDITION CODE AFTER EACH UDI CALL TO ENSURE PROPER RESULTS

TABLE 6-2. 1RMX 86 EXCEPTION CODES AND MNEMONICS

HEX CODE	MNEMONIC	HEX CODE	MNEMONIC
0000	E\$OK	0065	E\$EOF
0001	E\$TIME	0066	E\$FIXUP
0002	E\$MEM	0067	E\$NO\$LOADER\$MEM

```
.  
.  
.
```

(SEE COMPLETE LISTING IN RUN TIME SUPPORT MANUAL)

14-10

## OTHER UDI FACTS

### INTERRUPT HANDLING

PROGRAMS THAT RUN UNDER THE iRMX 86 OPERATING SYSTEM SHOULD USE iRMX 86 INTERRUPT MANAGEMENT TECHNIQUES TO HANDLE INTERRUPTS.

- THE UDI LIBRARIES DO NOT INCLUDE INTERRUPT MANAGEMENT.

### REENTRANCY

UDI LIBRARIES ARE FULLY REENTRANT WITH THE FOLLOWING RESTRICTIONS:

- EACH JOB MUST HAVE ITS OWN COPY OF THE UDI INTERFACE LIBS.

### MULTITASKING

- THE UDI LIBRARIES ARE FULLY COMPATIBLE WITH A MULTITASKING ENVIRONMENT. HOWEVER, THERE ARE NO UDI CALLS TO CREATE AND DELETE TASKS.

14-11

### LOGICAL NAMES

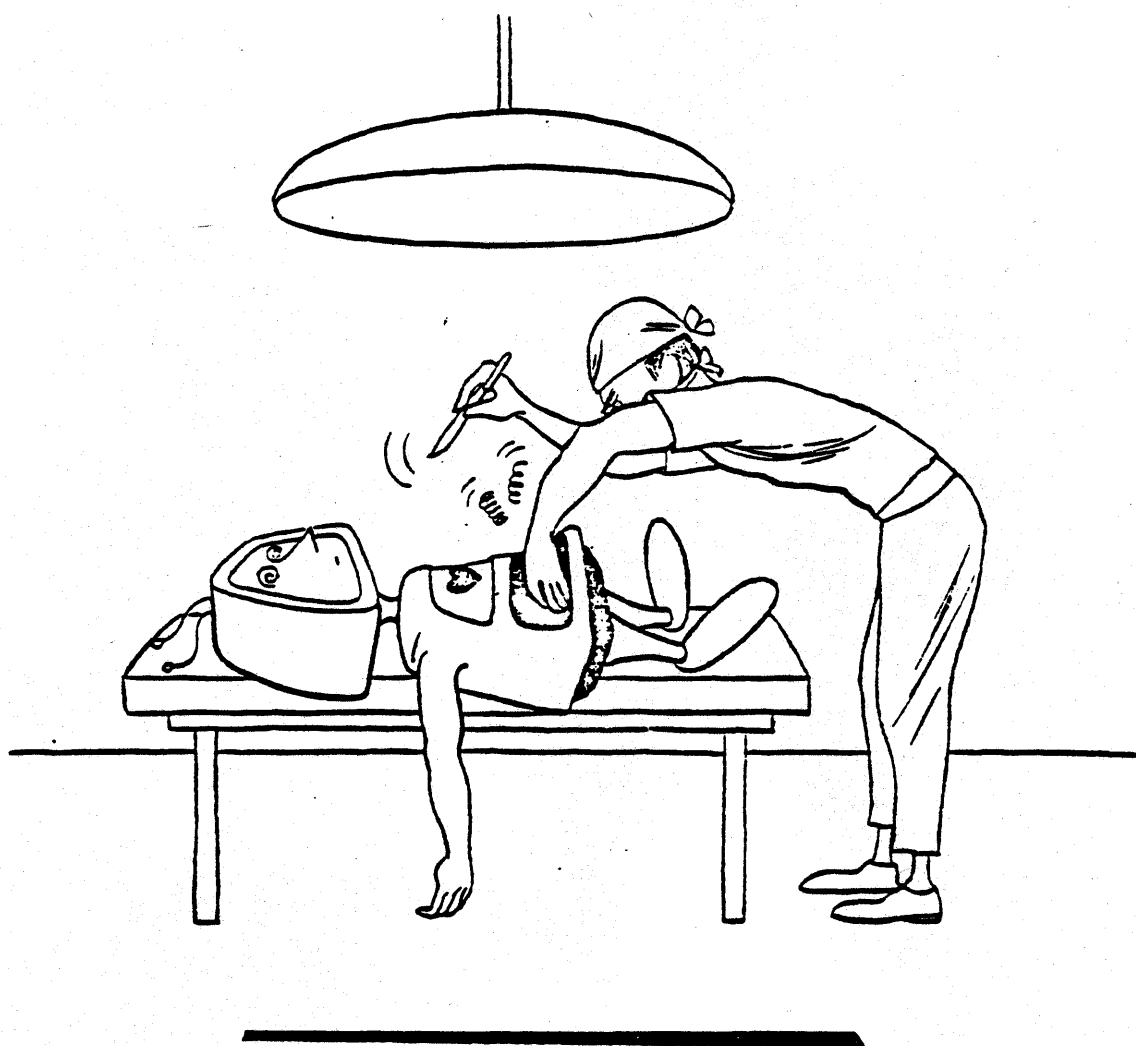
THE UDI USES CERTAIN LOGICAL NAMES TO MEAN SPECIAL THINGS. FOR EXAMPLE, :LP: MEANS "LINE PRINTER", :CO: MEANS "CONSOLE OUTPUT", AND "CI" MEANS "CONSOLE INPUT".

### REQUIREMENTS

A UDI JOB MUST BE CONFIGURED IN YOUR SYSTEM WITH I.C.U.86.

14-12

# APPENDIX







## **APPENDIX A**

### **ALTER EDITOR**

- INSERTION
- CORRECTING MISTAKES
- ENDING THE EDITING SESSION

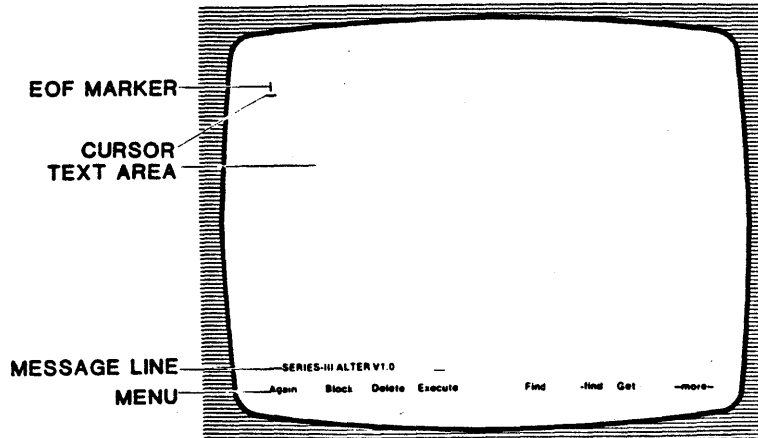


INVOCATION

- RUN ALTER :F1:LAB1.ASM

ALTER IS MENU DRIVEN

INITIAL SCREEN



● TO GET NEXT MENU:

TAB

## THE MENUS

MENU 1

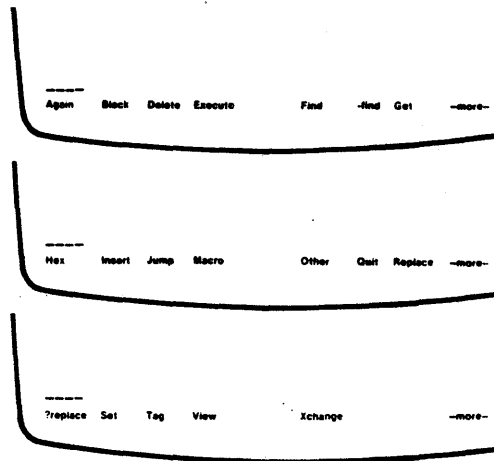
TAB

MENU 2

TAB

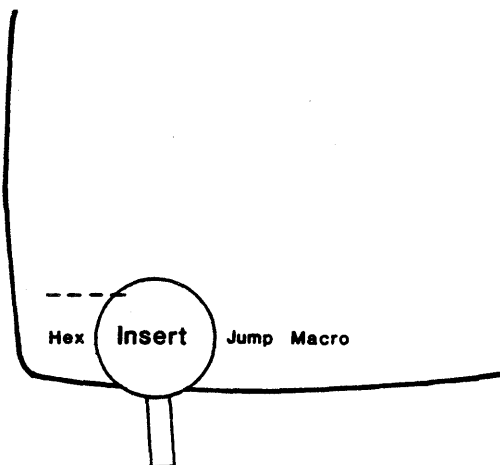
MENU 3

TAB

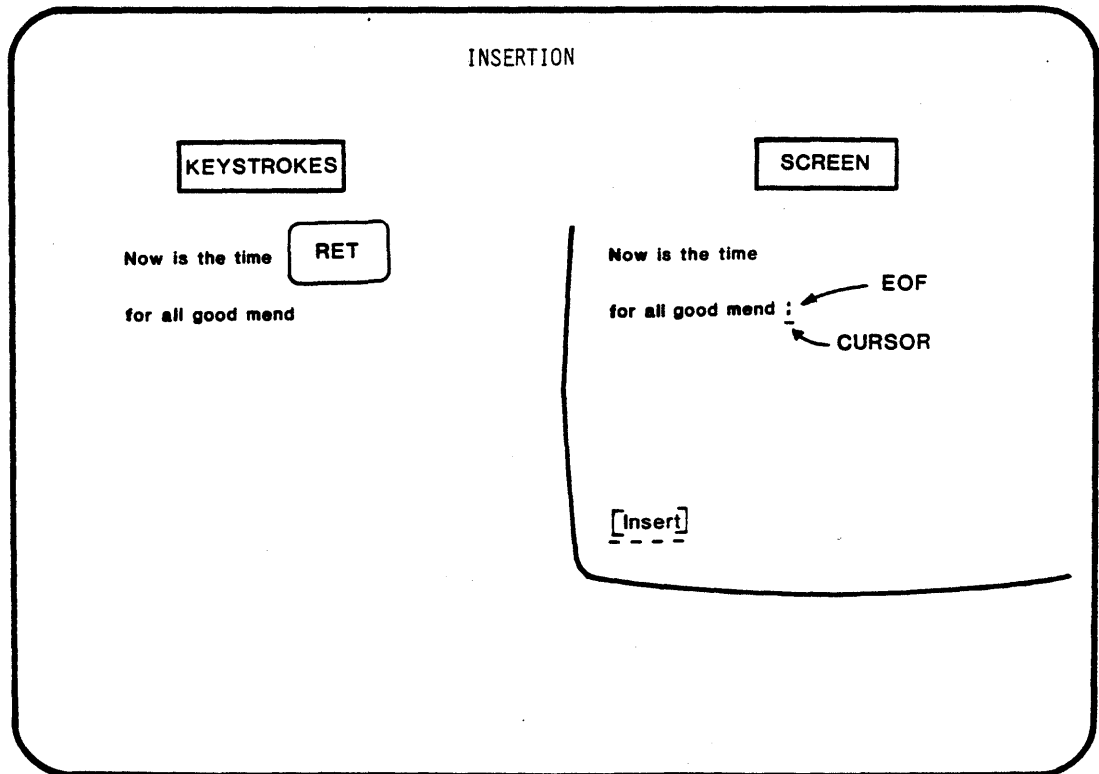
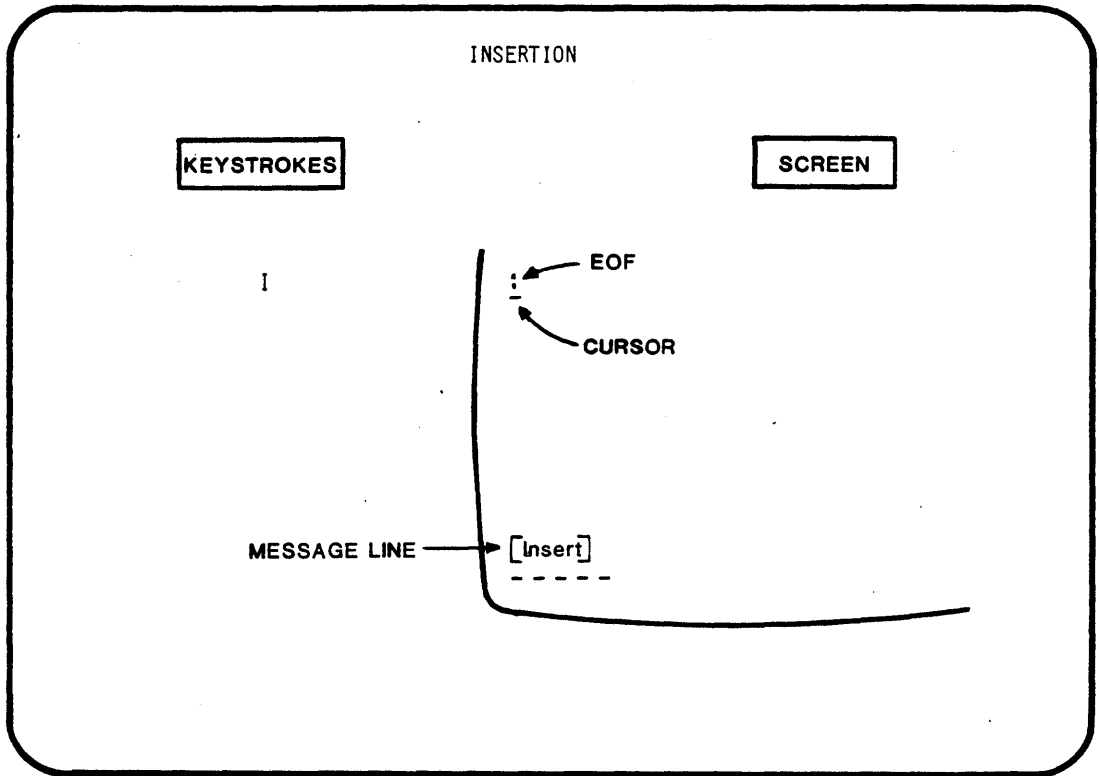


TO INVOKE A COMMAND, KEY THE FIRST LETTER OF THE COMMAND.

## INSERTING NEW TEXT



- TO INSERT TEXT, TYPE I



CORRECTING MISTAKES

KEYSTROKES

SCREEN

^RUBOUT

Now is the time  
for all good men: _

[insert]  
_ _ _

ENDING INSERTION

KEYSTROKES

SCREEN

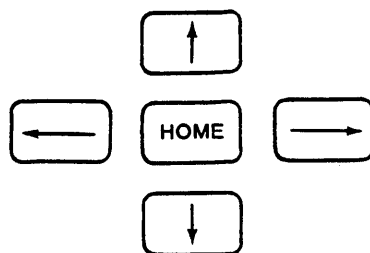
ESC

Now is the time  
for all good men: _

MENU

-----  
Again Block Delete Execute

## CURSOR CONTROL



- ARROW KEYS MOVE CURSOR ONE SPACE OR LINE FOR EDITING

## DELETING TEXT

<b>CONTROL</b>	<b>F</b>	DELETES CHARACTER AT CURSOR
<b>CONTROL</b>	<b>Z</b>	DELETES LINE ON WHICH CURSOR IS POSITIONED
<b>CONTROL</b>	<b>U</b>	UNDO-RESTORES DELETED CHARACTERS

THESE ALSO WORK DURING INSERTION

EXITING ALTER

KEYSTROKES

Q

Insert Jump Macro Other **Quit** Replace

A diagram of a control panel for 'EXITING ALTER'. It features a horizontal line with several buttons labeled 'Insert', 'Jump', 'Macro', 'Other', 'Quit', and 'Replace'. The 'Quit' button is circled and has a vertical stem extending downwards, indicating it is the selected option. A dashed line is positioned above the 'Insert' button.

EXITING ALTER, CONT.

KEYSTROKES

E

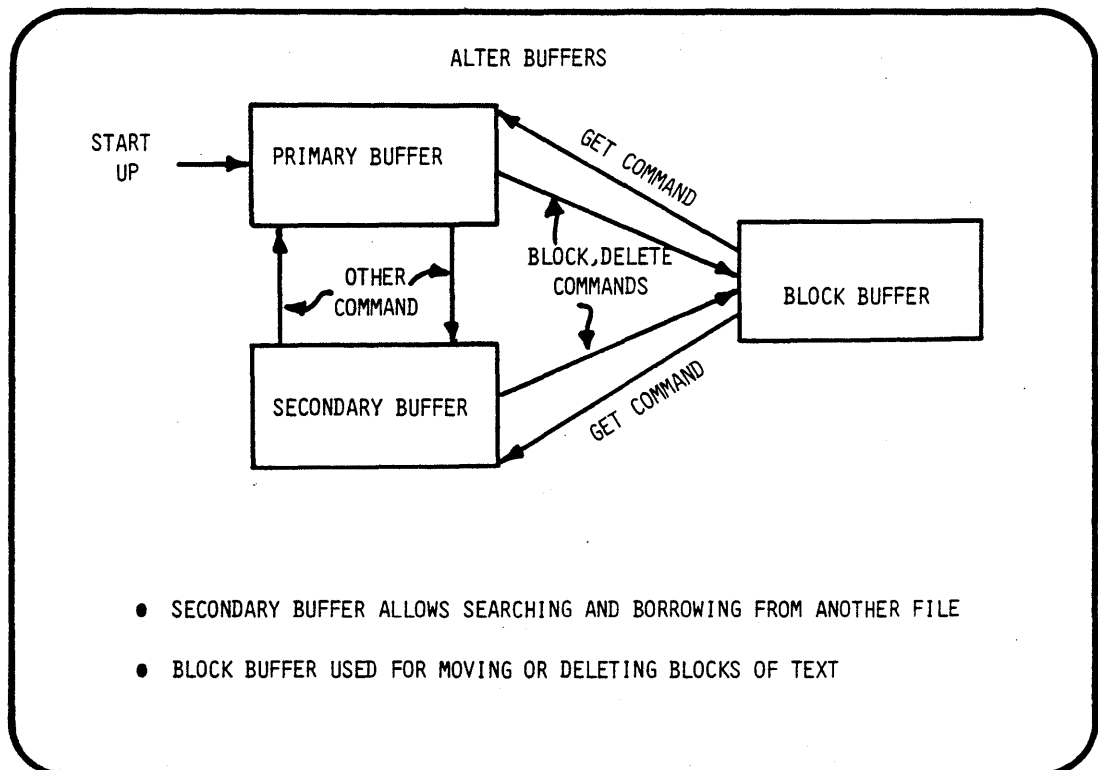
Abort **Exit** Init

A diagram of a control panel for 'EXITING ALTER, CONT.'. It features a horizontal line with three buttons labeled 'Abort', 'Exit', and 'Init'. The 'Exit' button is circled and has a vertical stem extending downwards, indicating it is the selected option. A dashed line is positioned above the 'Abort' button.



## ADVANCED ALTER FEATURES

- EDITING MULTIPLE FILES DURING ONE SESSION
- BLOCK MOVES



### ALTER INVOCATION

- RUN ALTER INPUT FILE [,OTHER INPUT FILE] <CR>

- OTHER INPUT FILE IS FILE TO BE EDITED IN SECONDARY BUFFER

#### EXAMPLES:

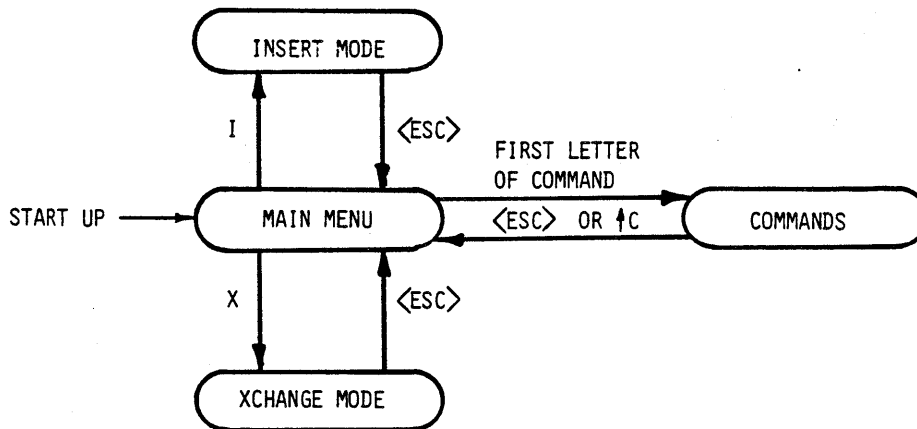
- RUN ALTER :F1:LAB1.ASM <CR>

- RUN ALTER :F1:LAB1.ASM,:F1:LAB1.LST <CR>

OR

- RUN ALTER :F1:LAB1.ASM-LST <CR>

### ALTER MODES







<ESC> EXECUTES COMMAND & RETURNS TO MAIN MENU

↑c ABORTS COMMAND

### XCHANGE MODE

- ALLOWS 'TYPING OVER' OF TEXT
- <ESC> RETURNS ALTER TO MAIN COMMAND LEVEL

### CURSOR MOVEMENT AND PAGING

- |      |                                                                                     |                                     |
|------|-------------------------------------------------------------------------------------|-------------------------------------|
| HOME |  | - MOVES CURSOR TO END OF LINE       |
| HOME |  | - MOVES CURSOR TO BEGINNING OF LINE |
| HOME |  | - PAGES DOWN                        |
| HOME |  | - PAGES UP                          |

## ALTER COMMANDS

### DELETE

KEYSTROKES

D



HOME

D

SCREEN

INTEL CORPORATION

@INTEL CORPORATION

@@INTEL CORPORATION

@INTEL CORPORATION@

- DELETES AND MOVES TEXT TO BLOCK USING BEGIN AND END MARKERS (@)
- RETRIEVE TEXT WITH GET COMMAND

BLOCK -- FOR COPYING TEXT

KEYSTROKES

B



HOME

B

SCREEN

INTEL CORPORATION

@INTEL CORPORATION

@@INTEL CORPORATION

@INTEL CORPORATION@

INTEL CORPORATION

- COPIES TEXT TO BLOCK BUFFER USING BEGIN & END MARKERS (@)
- RETRIEVE TEXT WITH GET COMMAND

GET -- RETRIEVING THE BLOCK BUFFER

KEYSTROKES

G

ESC

- RETRIEVES BLOCK BUFFER TO CURRENT CURSOR POSITION

NOTE: MAY ALSO BE USED TO 'GET' DISK FILES

## FINDING A STRING

### KEYSTROKES

F "STRING" **ESC**

- SEARCHES FORWARD FOR FIRST OCCURENCE OF "STRING" AND MOVES CURSOR IF FOUND
- -F COMMAND SEARCHES BACKWARDS

## REPLACING TEXT

### KEYSTROKES

R "OLD STRING" **ESC** "NEW STRING" **ESC**

- REPLACES FIRST OCCURENCE OF "OLD STRING" WITH "NEW STRING" AND MOVES CURSOR IF FOUND
- ? REPLACE PROMPTS YOU:  
OK TO REPLACE? (Y OR [N])

### REPEAT FUNCTION

- OPTIONAL FACTOR THAT INDICATES THE NUMBER OF TIMES TO EXECUTE A COMMAND
- PRECEDES ENTERING OF COMMAND LETTER
- / - MEANS REPEAT FOREVER

#### EXAMPLE:

10F "SAM" <ESC>

FINDS TENTH OCCURENCE OF SAM

### JUMPING TO BEGINNING OR END OF FILE

#### KEYSTROKES

JS  
OR  
JE

- JS MOVES CURSOR TO BEGINNING OF FILE
- JE MOVES CURSOR TO END OF FILE

### INSERTING CONTROL CHARACTERS

#### KEYSTROKES

H I "HEX VALUE" **ESC**

- INSERTS CONTROL CHARACTER AT CURRENT CURSOR POSITION AND DISPLAYS IT AS ?

EXAMPLE:

H I 0C **ESC** - INSERTS A FORM FEED CHARACTER

### DISPLAYING CONTROL CHARACTERS

#### KEYSTROKES

H O **ESC**

DISPLAYS HEXADECIMAL VALUE OF CHARACTER AT CURRENT CURSOR POSITION



## QUIT

### KEYSTROKES

Q

### MENU PROMPT LINE

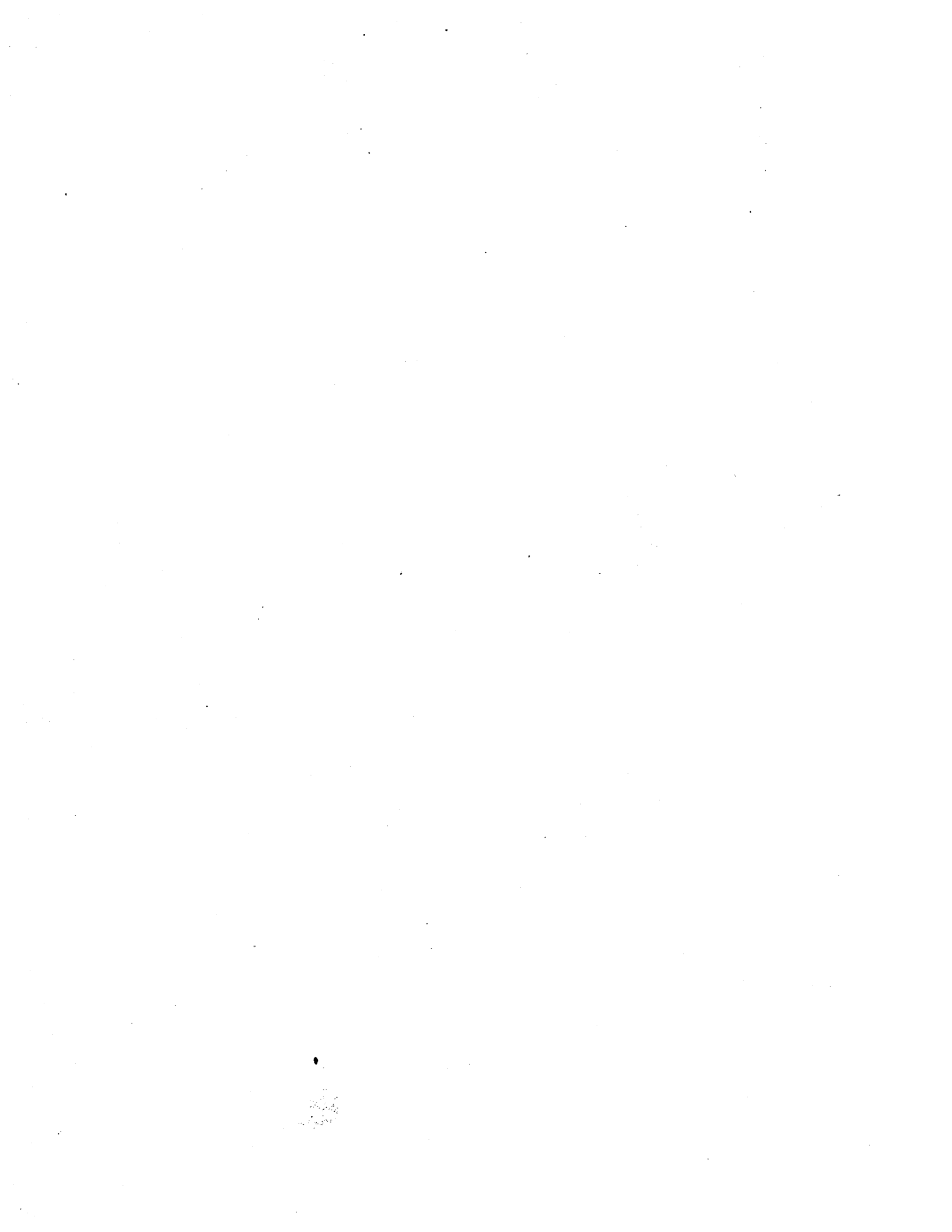
ABORT EXIT INIT UPDATE WRITE

#### SUBCOMMANDS:

- A - ABORT - ALL CHANGES LOST; RETURN TO OPERATING SYSTEM
- E - EXIT - RETURN TO OPERATING SYSTEM; FILE IS UPDATED
- I - INIT - RESTARTS EDITING SESSION; ALL CHANGES LOST
- U - UPDATE - UPDATES FILE; DOES NOT RETURN TO OPERATING SYSTEM
- W - WRITE - PROMPTS YOU FOR NEW FILE TO WRITE TO; DOES NOT RETURN TO OPERATING SYSTEM

## OTHER ALTER FEATURES

- MACROS
- DISK I/O
- TAGS
- ENVIRONMENT SETTINGS



**APPENDIX B**

**PL/M OVERVIEW**



PL/M IS A BLOCK STRUCTURED LANGUAGE

MY\$PROG:

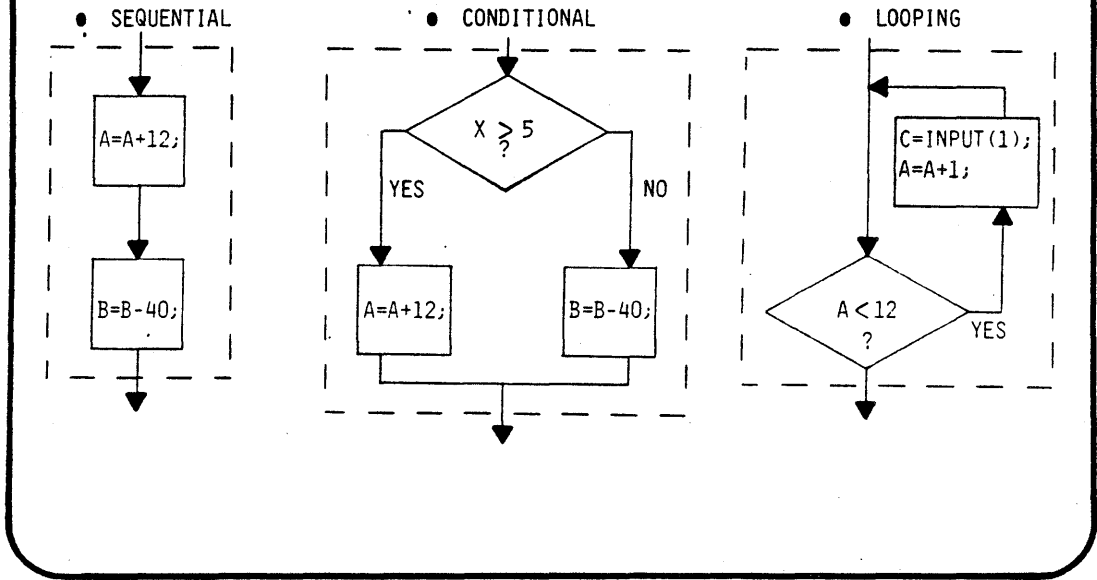
DO;
DECLARATIONS
EXECUTABLE STATEMENTS
END;

- RESERVE SPACE IN MEMORY
- GIVE A NAME TO THAT SPACE
  
- CAUSE WORK TO BE PERFORMED

PL/M STATEMENT FORMAT

- FREE FORMAT
- ENDS WITH A SEMICOLON
- COMMENTS
  - MAY BE USED WHEREVER A SPACE IS LEGAL
  - /* THIS IS A COMMENT */

## BASIC PL/M CONSTRUCTS



## PL/M SAMPLE PROGRAM

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE SAMPLE1  
 OBJECT MODULE PLACED IN :F1:PROG1.OBJ  
 COMPILER INVOKED BY: PLM86.86 :F1:PROG1.PLM

/* THIS PROGRAM ADDS TWO NUMBERS */

```

1      SAMPLE$1:
      DO;
2  1      DECLARE NUM$1 BYTE,
           NUM$2 BYTE,
           SUM BYTE;

3  1      NUM$1 = 3;
4  1      NUM$2 = 2;
5  1      SUM = NUM$1 + NUM$2;

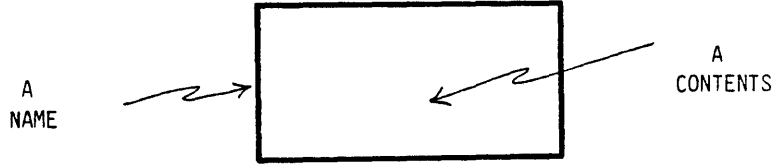
6  1      END SAMPLE$1;
  
```

### MODULE INFORMATION:

```

CODE AREA SIZE      = 0018H      24D
CONSTANT AREA SIZE = 0000H      0D
VARIABLE AREA SIZE = 0003H      3D
MAXIMUM STACK SIZE = 0000H      0D
15 LINES READ
0 PROGRAM WARNINGS
0 PROGRAM ERRORS
  
```

A VARIABLE HAS:



NAME - FIXED THROUGHOUT PROGRAM ....

CONTENTS - TRANSIENT ....

### VARIABLE DECLARATIONS

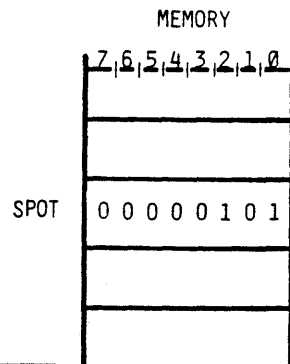
- BEFORE ANY VARIABLE CAN BE USED IT MUST BE DEFINED IN A DECLARATION STATEMENT
- VARIABLE DECLARATIONS
  - RESERVE SPACE IN MEMORY
  - ASSOCIATE AN IDENTIFIER WITH THAT SPACE
  - PRECEDE EXECUTABLE STATEMENTS

PROG: DO;

DECLARE SPOT BYTE;

SPOT =5;

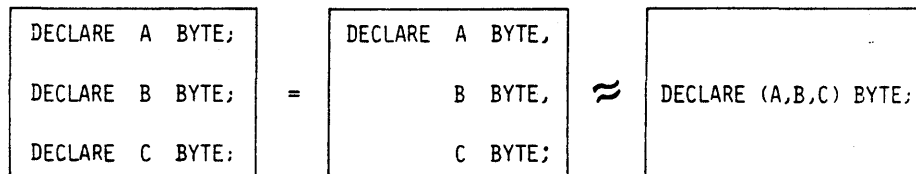
END;



PL/M VARIABLE TYPES

DATA TYPES		BYTES	RANGE
PL/M-80	PL/M-86		
BYTE	BYTE	1	0 TO 255
ADDRESS	WORD	2	0 TO 65,535
	DWORD	4	0 TO $2^{32} - 1$
	INTEGER	2	-32,768 TO +32,767
	REAL	4	$1.17 \times 10^{-38}$ TO $3.37 \times 10^{38}$
	POINTER	2 OR 4	TO BE DISCUSSED LATER
	SELECTOR	2	0 TO 65,535

DECLARATION FORMATS





- SIMPLIFY PROGRAM UPDATES:

```

DECLARE BUFFER$SIZE LITERALLY '256';
DECLARE COUNT WORD;
DECLARE BUFFER(BUFFER$SIZE) BYTE;

```

```

.
.
.
COUNT = BUFFER$SIZE; /*SAME AS: COUNT =256;*/
.
.

```


- IMPROVE DOCUMENTATION:

```

DECLARE SPACE LITERALLY '20H';
DECLARE CR LITERALLY '0DH';
DECLARE LF LITERALLY '0AH';

```

### OPERATOR PRECEDENCE

OPERATOR CLASS	OPERATOR	PRECEDENCE
PRECEDENCE	( )	HIGHEST  LOWEST
UNARY	-, +	
ARITHMETIC	*, /, MOD +, -	
RELATIONAL	<, =, >, <=, >=, <>	
LOGICAL	NOT AND OR, XOR	

- EXPRESSIONS WITH OPERATORS OF EQUAL PRECEDENCE ARE EVALUATED LEFT TO RIGHT

ARITHMETIC EXPRESSION SUMMARY

VARIABLE TYPE		KIND OF ARITHMETIC	OPERAND TYPE	ARITHMETIC OPERATION	RESULT
PL/M-80	PL/M-86				
BYTE AND ADDRESS	BYTE AND WORD	UNSIGNED	1 BYTE, 1 BYTE	+, -, *, /, MOD	1 BYTE 2 BYTE
			1 BYTE, 2 BYTE	+, -, *, /, MOD	2 BYTE
			2 BYTE, 2 BYTE	+, -, *, /, MOD	2 BYTE
	DWORD	UNSIGNED	1 BYTE, 4 BYTES 2 BYTE, 4 BYTES 4 BYTE, 4 BYTES	+, -, *, /, MOD	4 BYTES 4 BYTES 4 BYTES
	INTEGER	SIGNED	INTEGER, INTEGER	+, -, *, /, MOD	INTEGER
	REAL	FLOATING POINT	REAL, REAL	+, -, *, /	REAL

LOGICAL AND RELATIONAL EXPRESSION SUMMARY

VARIABLE TYPE		OPERAND TYPE	RELATIONAL RESULT	LOGICAL RESULT
PL/M-80	PL/M-86			
BYTE AND ADDRESS	BYTE AND WORD	1 BYTE, 1 BYTE	1 BYTE	1 BYTE
		1 BYTE, 2 BYTE	1 BYTE	2 BYTE
		2 BYTE, 2 BYTE	1 BYTE	2 BYTE
	DWORD	1 BYTE, 4 BYTES 2 BYTES, 4 BYTES 4 BYTES, 4 BYTES	1 BYTE 1 BYTE 1 BYTE	4 BYTES 4 BYTES 4 BYTES
	INTEGER	INTEGER, INTEGER	1 BYTE	ILLEGAL
	REAL	REAL, REAL	1 BYTE	ILLEGAL

PORT INPUT AND OUTPUT  
DATA IS "READ" FROM OR "WRITTEN" TO SPECIFIED PORT

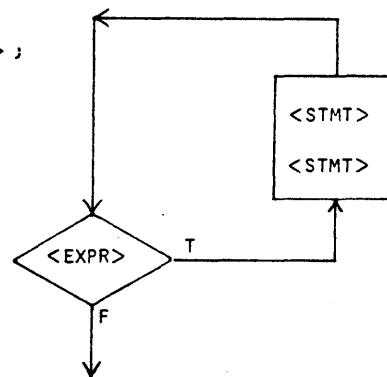
PL/M 80 1 BYTE READ OR WRITTEN	PL/M 86 1 BYTE READ OR WRITTEN	<p>&lt;VARIABLE&gt; = INPUT (&lt;PORT\$EXPR &gt;);</p> <p>OUTPUT(&lt;PORT\$EXPR &gt;) =   &lt;VARIABLE\$EXPR&gt;   ; CONSTANT</p>
	2 BYTES READ OR WRITTEN	<p>&lt;VARIABLE&gt; = INWORD (&lt;PORT\$EXPR&gt;);</p> <p>OUTWORD(&lt;PORT\$EXPR&gt;) =   &lt;VARIABLE\$EXPR&gt;   ; CONSTANT</p>

<PORT\$EXPR>	
PL/M-80	PL/M-86
<ul style="list-style-type: none"> <li>• MUST BE A NUMBER OR A CONSTANT EXPRESSION</li> <li>• $0 \leq \text{PORT\\$EXPR} \leq 255$</li> </ul>	<ul style="list-style-type: none"> <li>• CAN BE A NUMBER, CONSTANT EXPRESSION, OR EXPRESSION</li> <li>• $0 \leq \text{PORT\\$EXPR} \leq 65535$</li> </ul>

DO WHILE BLOCKS

```

<LABELNAME> :] DO WHILE <EXPRESSION> ;
               [<STATEMENT>] ;
               .
               .
               .
               END [<LABELNAME>] ;
    
```

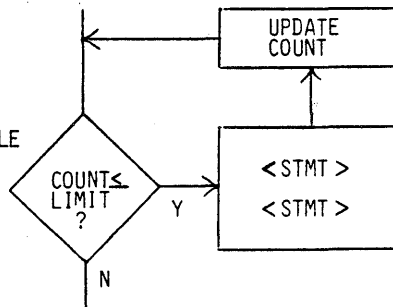


- STATEMENTS EXECUTE AS LONG AS <EXPRESSION> EVALUATES TO A NUMBER WITH BIT 0=1

### ITERATIVE 'DO' BLOCKS

DO <COUNTER\$VARIABLE> = <STARTEXP> TO <LIMITEXP> [BY <STEPEXP> ];  
 [<STATEMENT> ]; . . .  
 END;

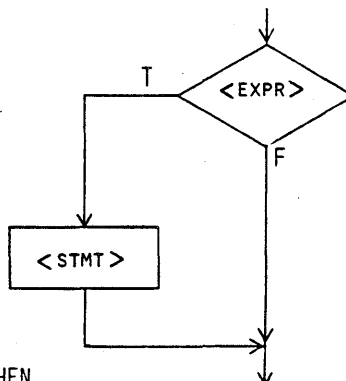
WHERE  
 <COUNTER\$VARIABLE> IS A 1 BYTE OR 2 BYTE VARIABLE  
 <STARTEXP> , <LIMITEXP> AND <STEPEXP>  
 ARE EXPRESSIONS



- STATEMENTS WITHIN AN ITERATIVE DO BLOCK ARE EXECUTED REPEATEDLY

### IF..THEN

THE CONDITIONAL STATEMENT TESTS AN EXPRESSION FOR TRUE OR FALSE  
 AND CAUSES CODE TO BE EXECUTED OR BYPASSED ACCORDINGLY.



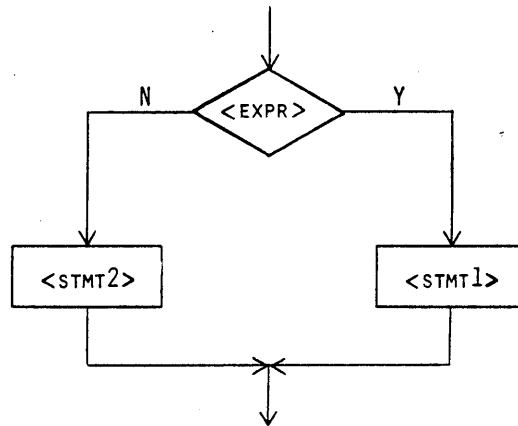
IF <EXPRESSION> THEN  
 <STATEMENT> ;

AN EXPRESSION IS TRUE IF BIT0 = 1.

### IF..THEN..ELSE

"ELSE" CAUSES AN ALTERNATE STATEMENT  
TO BE EXECUTED

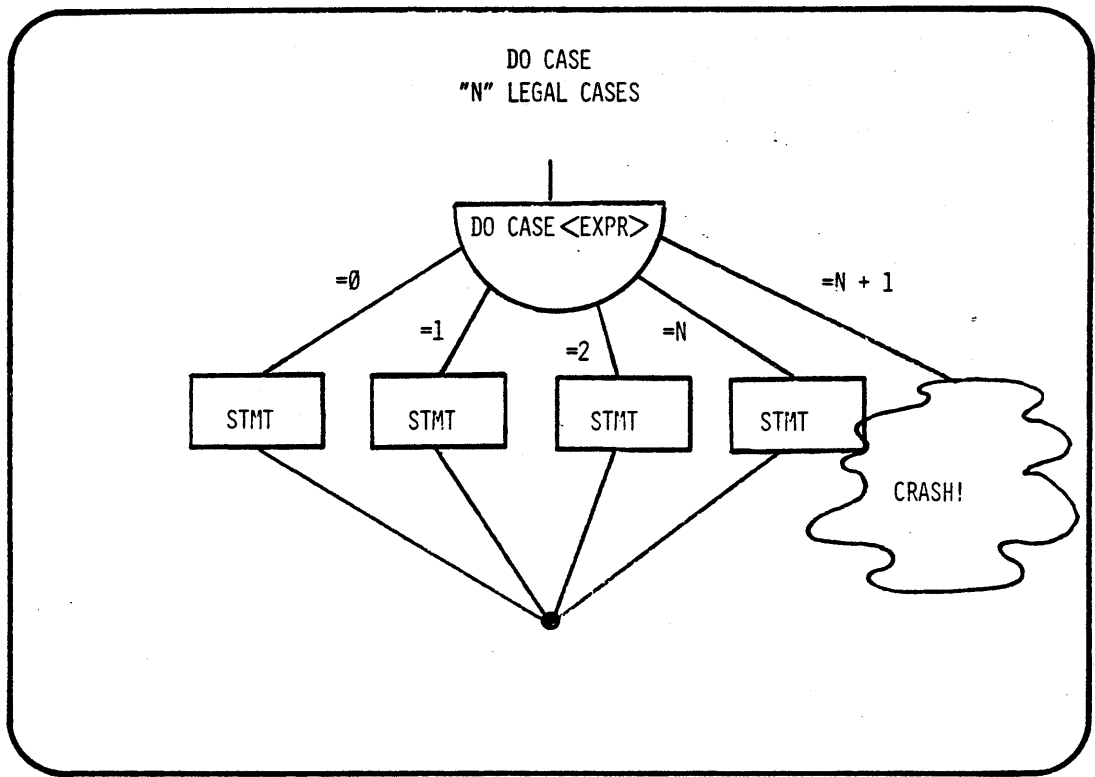
```
IF <EXPRESSION>  
THEN  
  <STATEMENT1> ;  
[ELSE  
  <STATEMENT2>;]
```



### THE 'DO CASE'

```
[<LABELNAME> :] DO CASE <EXPRESSION> ;  
    <STATEMENT>          /* EXECUTED WHEN EXPRESSION = 0 */  
    [<STATEMENT>]        /* EXECUTED WHEN EXPRESSION = 1 */  
    .  
    .  
    [<STATEMENT>]        /* EXECUTED WHEN EXPRESSION = N */  
END [<LABELNAME> ] ;
```

IMPORTANT: NO RANGE CHECK IS PERFORMED ON THE VALUE OF THE EXPRESSION AFTER IT IS COMPUTED. IF THE VALUE COMPUTED IS GREATER THAN THE NUMBER OF 'BASIC\$STATEMENTS', THE PROGRAM CRASHES.



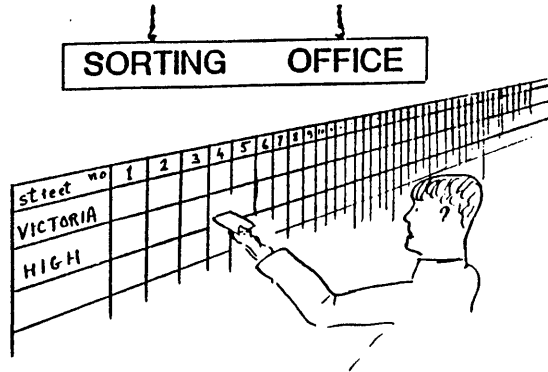
DO CASE EXAMPLE

```

      :
DO CASE (STOP$LIGHT$VALUE);
DO;                                     /* CASE 0 */
  CURRENT$STATE = GREEN$LIGHT;
  TIME = SHORT;
END;
DO;                                     /* CASE 1 */
  CURRENT$STATE = YELLOW$LIGHT;
  TIME = GOTCHA;
END;
DO;                                     /* CASE 2 */
  CURRENT$STATE = RED$LIGHT;
  TIME = ETERNITY;
END;
;                                       /* CASE 3 IS NULL */
  CURRENT$STATE = BLINK$YELLOW;        /* CASE 4 */
END;
      :

```

ARRAYS CAN BE USED TO MANIPULATE  
GROUPS OF RELATED DATA ITEMS



- AN ARRAY OF BOXES FOR EACH STREET
- EACH ARRAY HAS A NAME
- REFER TO BOX FOR NO.4 HIGH STREET:  
HIGH(4)
- ALL MEMBERS OF AN ARRAY MUST BE OF  
THE SAME TYPE

```
DECLARE VICTORIA(120) BYTE;
```

```
DECLARE HIGH (81) BYTE;
```

STORAGE OF DATA INPUT FROM TEMPERATURE SENSOR

```
DATA$IN:
```

```
DO;
```

```
    DECLARE READY LITERALLY '01';
```

```
    DECLARE TEMP$BUFFER(256) BYTE;
```

```
    DECLARE TEMP$BUFFER$PTR BYTE;
```

```
    DO TEMP$BUFFER$PTR = 0 TO 255;
```

```
        DO WHILE INPUT ( 4 ) <> READY;
```

```
        END;
```

```
        TEMP$BUFFER(TEMP$BUFFER$PTR) = INPUT ( 8 );
```

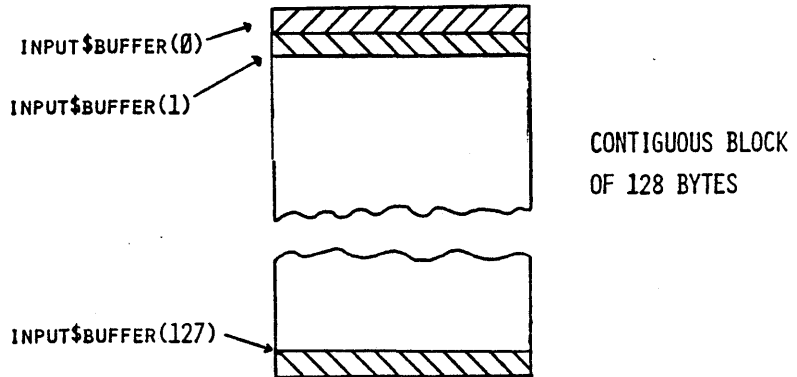
```
    END;
```

```
END;
```

### ARRAY DECLARATIONS

```
DECLARE <ARRAY$NAME> (<ARRAY$CONST>) {  
    BYTE  
    WORD  
    INTEGER  
    REAL  
    POINTER } ;  
    PL/M-86 ONLY
```

DECLARE INPUT\$BUFFER (128) BYTE;

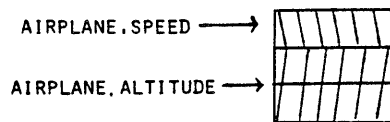


* LATER

### STRUCTURES

- LOGICAL AND PHYSICAL GROUPS OF DISSIMILIAR, RELATED DATA ITEMS
- A STRUCTURE MAY CONTAIN DATA ITEMS OF DIFFERENT TYPES

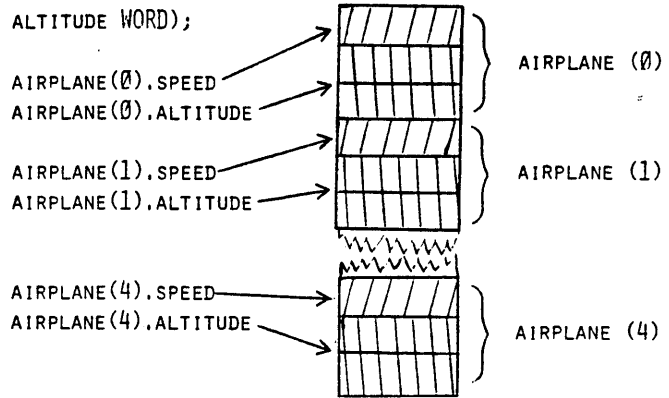
```
DECLARE AIRPLANE STRUCTURE(  
    SPEED BYTE,  
    ALTITUDE WORD);
```





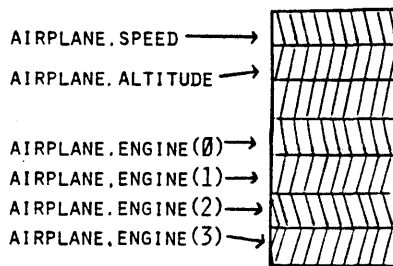
### ARRAY OF STRUCTURES

```
DECLARE AIRPLANE (5) STRUCTURE (  
  SPEED BYTE,  
  ALTITUDE WORD);
```



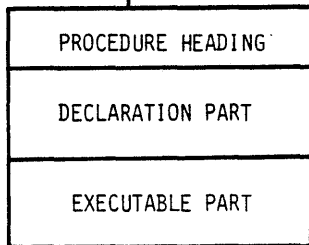
### STRUCTURE WITH AN ARRAY AS AN ELEMENT

```
DECLARE AIRPLANE STRUCTURE(  
  SPEED BYTE,  
  ALTITUDE WORD,  
  ENGINE (4) BYTE);
```



## PROCEDURE DECLARATION

```
MODULE$NAME:  
DO;  
  [ <DECLARATION$STATEMENTS> ]  
  
  PROCEDURE$NAME : PROCEDURE ;  
    [ <DECLARATION$STATEMENT> ] ...  
    <EXECUTABLE$STATEMENT> ...  
    <END [ <PROCEDURE$NAME> ] ;  
  
  [ <DECLARATION$STATEMENTS> ]  
  [ <EXECUTABLE$STATEMENTS> ]  
  
END;
```



## PARAMETERLESS PROCEDURE

```
MAIN: DO;  
  DECLARE (RESULT, OP1, OP2, ANSWER) BYTE;  
  
  SUM: PROCEDURE;          /* PROCEDURE DEFINITION */  
    RESULT = OP1 + OP2;  
  END SUM;  
  
  OP1 = 4;                 /* START OF MAIN */  
  OP2 = 5;  
  CALL SUM;                /* PROCEDURE INVOCATION */  
  
  ANSWER = RESULT;  
END MAIN;
```

- PROCEDURE ACCESSES GLOBAL VARIABLES

## PROCEDURE WITH PARAMETERS

```
MAIN: DO;
      DECLARE (X, Y, ANSWER1) BYTE;
      DECLARE (A, B, ANSWER2) BYTE;
      DECLARE RESULT BYTE;

      SUM: PROCEDURE (OP1, OP2); /* PROCEDURE DEFINITION */
            DECLARE (OP1, OP2) BYTE;
            RESULT = OP1 + OP2;
      END SUM;
      CALL SUM(X, Y);           /* PROCEDURE INVOCATION */
      ANSWER1 = RESULT;
      CALL SUM (A-3,B-2);      /* PROCEDURE INVOCATION */
      ANSWER2 = RESULT;
END MAIN;
```

- TWO INPUT PARAMETERS
- PROCEDURE OUTPUT IS RETURNED IN A GLOBAL VARIABLE

## TYPED PROCEDURES

A SINGLE VALUE IS RETURNED

```
MAIN: DO;
      DECLARE (X,Y,ANSWER) BYTE;

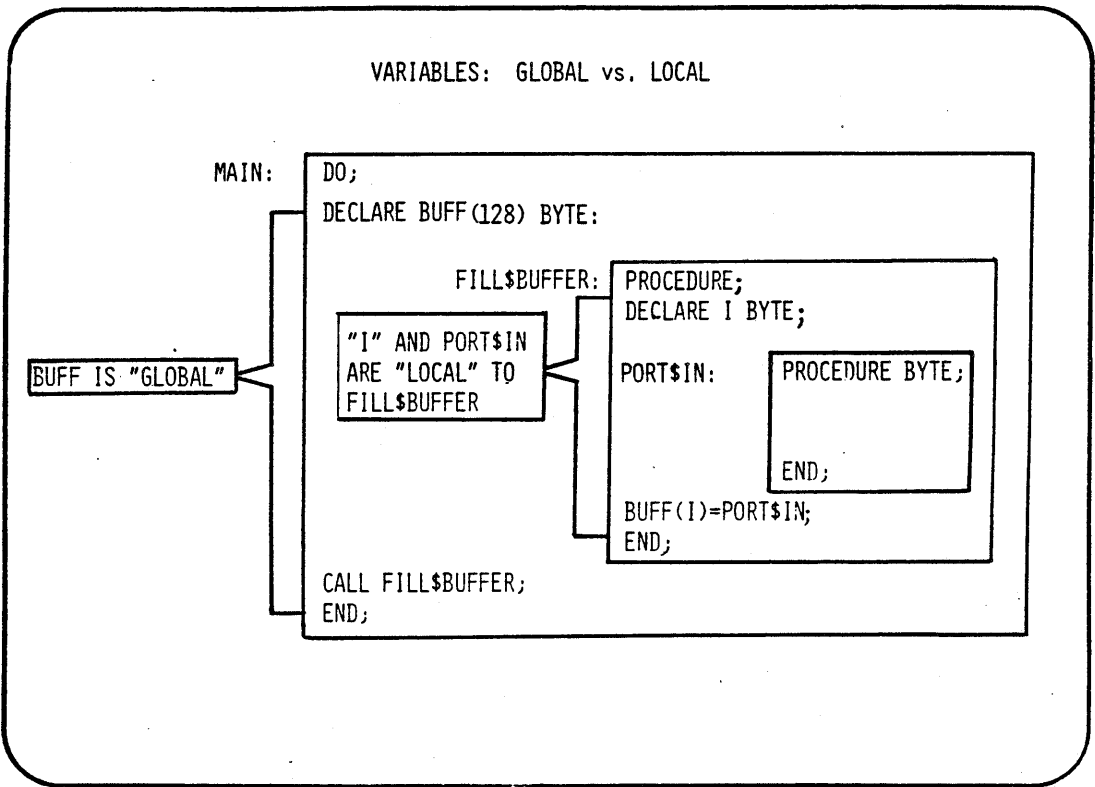
      SUM: PROCEDURE (OP1, OP2) BYTE; /* PROCEDURE DEFINITION */
            DECLARE (OP1, OP2) BYTE;

            RETURN OP1 + OP2;
      END SUM;

      X = 3;
      Y = 2;

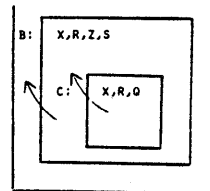
      ANSWER = SUM (X,Y);          /* PROCEDURE INVOCATION */
      .
      .
      .
END MAIN;
```

## VARIABLES: GLOBAL vs. LOCAL



## SCOPE OF VARIABLES

- THE SCOPE OF A VARIABLE IS THE FULL LENGTH OF THE BLOCK, UNLESS IT IS REDECLARED WITHIN A NESTED BLOCK.
- TO DETERMINE IF A VARIABLE/LABEL CAN BE USED IN A BLOCK:
  - 1) IF IT IS NOT DECLARED IN THE BLOCK, GO TO THE NEXT OUTER BLOCK.
  - 2) IF DEFINED, THE SCOPE IS SET, ELSE REPEAT 1) AND 2).
  - 3) IF REACH THE OUTER MOST BLOCK WITHOUT ENCOUNTERING THE DECLARATION, THE VARIABLE/LABEL CANNOT BE USED.



## SYNTAX

- DECLARE <VARIABLE\$NAME> BASED <POINTER\$VARIABLE>

[(<ARRAY\$CONSTANT>)] {
   
     BYTE
   
     WORD
   
     INTEGER
   
     REAL
   
     STRUCTURE
 } ;

- EXAMPLE

```

SUM:  PROCEDURE (OP1_PTR, OP2_PTR, RSLT_PTR);
      DECLARE   OP1_PTR   POINTER,
                OP2_PTR   POINTER,    /* ADDRESS FOR PL/M-80 */
                RSLT_PTR  POINTER;
  
```

```

      DECLARE OP1  BASED OP1_PTR (6) BYTE,
  
```

BASED VARIABLE

BASE

DIMENSION SPECIFIER OF BASED VARIABLE

```

      OP2 BASED OP1_PTR (6) BYTE,
  
```

```

      RESULT BASED RSLT_PTR (6) BYTE;
  
```

## PROGRAM TO SUM TWO ARRAYS USING BASED VARIABLES

```

ARRAY$SUM: DO;
  DECLARE ANSWER (6) BYTE, TOTAL (8) BYTE,
           X (6)   BYTE, A (8)   BYTE,
           Y (6)   BYTE, B (8)   BYTE;
  
```

```

SUM:  PROCEDURE (OP1_PTR, OP2_PTR, RSLT_PTR, ARRAYSIZE);
      DECLARE OP1_PTR   POINTER,
                OP2_PTR   POINTER,
                RSLT_PTR  POINTER,    /* ADDRESS FOR PL/M-80 */
                ARRAYSIZE BYTE;
  
```

```

      DECLARE OP1  BASED OP1_PTR (1) BYTE,
                OP2  BASED OP2_PTR (1) BYTE,
                RESULT BASED RSLT_PTR (1) BYTE;
  
```

```

      DECLARE I  BYTE;
  
```

```

      DO I = 0 TO ARRAYSIZE;
  
```

```

      RESULT (I) = OP1(I) + OP2(I)
  
```

```

      END;
  
```

```

      END SUM;
  
```

```

      CALL SUM(@X, @Y, @ANSWER, LAST(ANSWER));
  
```

```

      CALL SUM (@A, @B, @TOTAL, LAST(TOTAL));
  
```

```

      END ARRAY$SUM;
  
```

NOTE: IN PL/M-80, USE  
 "." INSTEAD OF "a".

A "BASED VARIABLE" IS A PROCEDURE'S MOVABLE TEMPLATE FOR A DATA STRUCTURE DECLARED IN A CALLING PROGRAM.

```

MAIN:
DO;
DECLARE ARRAY$1 (6) BYTE;
        ARRAY$2 (4) BYTE;

ARRAY$HANDLER: PROCEDURE (ARRAY$PTR);
        DECLARE ARRAY$PTR POINTER;
        DECLARE BLOCK BASED ARRAY$PTR (1) BYTE;

        /* EXECUTABLE STATEMENTS */

END ARRAY$HANDLER:
.
.
CALL ARRAY$HANDLER (@ARRAY$1);
.
.
CALL ARRAY$HANDLER (@ARRAY$2);

END MAIN;

```

#### SOME PRIMARY CONTROL NAMES

NOPRINT / PRINT (SOURCE\$FILE.LST)*	DESTINATION OF LISTING
SYMBOLS / NOSYMBOLS*	GENERATE SYMBOL TABLE LISTING
XREF / NOXREF*	GENERATE CROSS REFERENCE LIST
DEBUG / NODEBUG*	RETAINS SYMBOL TABLE

#### PL/M-86 ONLY

OPTIMIZE (0 / 1* / 2 / 3)

ROM/RAM*

TYPE*/NO TYPE

SMALL* /COMPACT / MEDIUM / LARGE

* DEFAULT CONDITION

}	0	MINIMAL CODE OPTIMIZATION
	1:	CONSTANT & COMMON EXPRESSIONS
	2:	#1 PLUS LOCAL CODE OPTIMIZATION
	3:	#2 PLUS FURTHER OPTIMIZATION
		PLACEMENT OF CONSTANTS IN CODE SEGMENT
		(SEE P. 8-6)

## PL/M COMPILER OPERATION

COMMAND SYNTAX:

```
[;<DEVICE>:] | PLM86 |  
| PLM80 | <SOURCE$FILE> [<CONTROLS>]
```

<CONTROLS> IS A SEQUENCE OF EITHER

<PRIMARY\$CONTROLS> WHICH MUST OCCUR BEFORE SOURCE CODE  
<GENERAL\$CONTROLS> WHICH MAY OCCUR ANYWHERE (INVOCATION  
OR IMBEDDED WITHIN THE SOURCE CODE.)

## SOME GENERAL CONTROL NAMES

LIST* / NOLIST

CODE / NOCODE*

EJECT / -*

INCLUDE / -*

OVERFLOW / NOOVERFLOW* (PL/M-86 ONLY)

SUSPEND / RESUME LISTING

GENERATE OBJECT CODE INTERLIST

GENERATE PAGE EJECT

INCLUDE CONTENTS OF ANOTHER FILE

INTEGER OVERFLOW DETECT CODE  
(REQUIRES USER SUPPLIED TYPE 4

INTERRUPT SERVICE PROCEDURE.

DISCUSSED IN CH. 16)

-* NO DEFAULT

## COMPILER CONTROL USAGE

> PLM86 :F1:ALICE.SRC DEBUG SYMBOLS OPTIMIZE(2)

IN THE SOURCE FILE WE HAVE THE FOLLOWING:

'\$' MUST  
APPEAR IN  
COLUMN 1

```
$CODE
BIG$TIME: DO;
$INCLUDE (:F1:PREAMB.LIT)
$EJECT
DECLARE CAROL BYTE;
    ...
$EJECT
$NOLIST
    ...
[<EXECUTABLE$STATEMENTS>]
END;
```

COMPILATION FAILS  
IF :F1: NOT READY  
OR IF :F1: PREAMB.LIT  
DOES NOT EXIST

## PL/M-86: SIZE CONTROL SWITCH

ALLOCATION OF MEMORY AND THE WAY IN WHICH LOCATIONS ARE REFERENCED BY A PROGRAM IS DETERMINED BY THE SIZE CONTROL SWITCH.

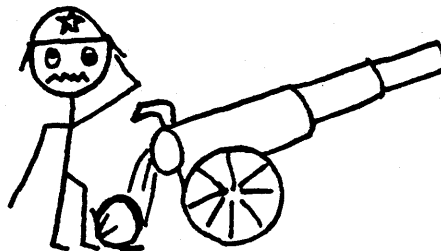
1. 'SMALL' - FOR PROGRAMS WITH LESS THAN 64K BYTES OF CODE AND LESS THAN 64K BYTES OF DATA. (MAXIMUM OF 128K BYTES.)
2. 'COMPACT' - FOR PROGRAMS WITH A MAXIMUM OF 64K BYTES EACH OF CODE, DATA, AND STACK.
3. 'MEDIUM' - FOR PROGRAMS WITH MORE THAN 64K BYTES OF CODE AND LESS THAN 64K BYTES OF DATA.
4. 'LARGE' - FOR PROGRAMS WITH MORE THAN 64K BYTES OF CODE AND MORE THAN 64K BYTES OF DATA.

FOR GREATEST EFFICIENCY, USE THE SMALL CASE WHEN POSSIBLE. UPGRADED PL/M-80 PROGRAMS MUST USE THE 'SMALL' CASE.



## APPENDIX C

### THE iRMX 86 BOOTSTRAP LOADER





### WHY THE NEED FOR A BOOTSTRAP LOADER?

- MAINTENANCE COSTS GREATLY REDUCED
  - MINIMIZE THE NEED TO MANUFACTURE ROM CHIPS
  - SOFTWARE UPGRADES AND BUG FIXES ARE EASILY INSTALLED AND DELIVERED

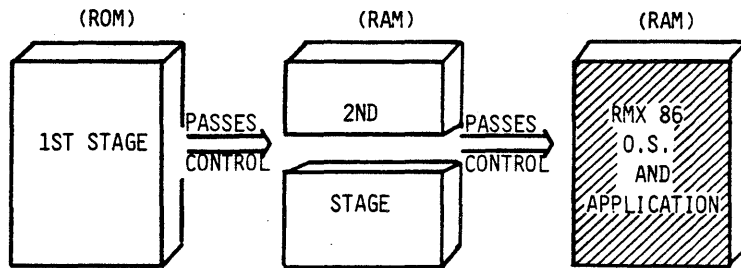


### FUNCTION AND CONTROLLERS

- THE BOOTSTRAP LOADER FUNCTION
  - LOAD THE RMX86 O.S AND APPLICATION SOFTWARE FROM SECONDARY STORAGE INTO RAM
- SECONDARY STORAGE SUPPORT
  - THE RMX86 PRODUCT INCLUDE BOOTSTRAP LOADER DEVICE DRIVERS FOR THE FOLLOWING CONTROLLERS:
    - 1) ISBC 204
    - 2) ISBC 208
    - 3) ISBC 206
    - 4) ISBC 215
    - 5) ISBC 218
    - 6) ISBC 254

## STAGES

- THERE ARE TWO PARTS TO THE APPLICATION LOADER
  - THE FIRST STAGE AND THE SECOND STAGE



- RUNS UPON SYSTEM RESET
- FINDS DEVICE TO LOAD FROM
- LOADS PART OF 2ND STAGE AND TRANSFERS CONTROL
- FINISHES LOADING ITSELF
- FINDS FILE TO LOAD FROM
- LOADS FILE AND TRANSFERS CONTROL

## THE SECOND STAGE

- A VOLUME MAY BE FORMATTED WITH THE HUMAN INTERFACE OR THE FILES UTILITIES.
- THE FORMATTING PROCESS WILL PLACE THE SECOND STAGE ON THE VOLUME WITH NO EFFORT ON YOUR PART.
- THE SECOND STAGE IS  $\approx$  6K OF LTL CODE.

### THE FIRST STAGE

- THE FIRST STAGE CONSISTS OF TWO PARTS
- THE FIRST STAGE RESIDES IN ROM
- DEVICE DRIVER SOFTWARE (PART 1)
  - SIZE DEPENDS ON HOW MANY DEVICE DRIVERS YOU CHOOSE TO INCLUDE. (EACH DRIVER 300 TO 500 BYTES)
- BOOT LOADER CORE (PART 2)
  - THIS PART LOADS THE 2ND STAGE
  - SIZE DEPENDS ON HOW MANY OPTIONS YOU CHOOSE. (SIZE 100 TO 500 BYTES)

### FIRST STAGE OPTIONS

- THE LOCATION OF THE FIRST STAGE
- THE LOCATION WHERE THE FIRST STAGE LOADS THE SECOND STAGE
  - (USUALLY IN THE FREE SPACE OF THE FINAL SYSTEM TO BE LOADED)
- METHOD TO BE USED FOR DEVICE SELECTION
  - NO SELECTION
  - AUTO SELECTION
  - MANUAL SELECTION
- METHOD TO BE USED FOR FILE SELECTION
  - LOADING A DEFAULT FILE NAME
  - ALLOWING THE END USER TO SPECIFY A FILE NAME

#### DEVICE SELECTION

- NO SELECTION
  - BOOTSTRAP LOADER ALWAYS USES SAME DEVICE
  - IF DEVICE IS NOT READY, LOADER TERMINATES
- AUTO DEVICE SELECTION
  - YOU PROVIDE A LIST OF DEVICES
  - THE LOADER CYCLES THROUGH THE LIST UNTIL IT FINDS A READY DEVICE
- MANUAL DEVICE SELECTION
  - THE LOADER PROMPTS THE USER AT THE TERMINAL (*)
  - THE USER ENTERS A DEVICE NAME (E.G. :F0:)
  - IF NAME IS NOT FOUND THEN LOADER SWITCHES TO AUTO DEVICE SELECTION

#### FILE SELECTION

- THE LOADABLE FILE MUST BE A NAMED FILE
- LOADING A DEFAULT FILE
  - THE DEFAULT FILE IS (/SYSTEM/RMX86)
- SPECIFYING A FILE NAME
  - DEVICE SELECTION MUST BE MANUAL
  - (E.G. :YES:LIFE/IS/HARD/IN/THE/FAST/LANE)

## PROBLEMS

### WHAT IF:

- I AM NOT USING ONE OF THE BOOTSTRAP DEVICE DRIVERS SUPPLIED WITH THE RMX86 PRODUCT?
- I DO NOT HAVE THE STANDARD 957B "TERMINAL" SUPPORT?



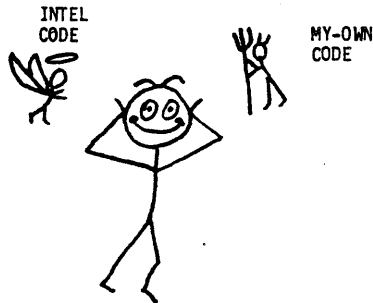
WAIT--DON'T DO IT ! ! !

## WRITING YOUR OWN DEVICE DRIVER

- A DEVICE DRIVER FOR THE BOOTSTRAP LOADER CONSISTS OF TWO PROCEDURES:
  - DEVICE\$INIT AND DEVICE\$READ
- THE PROCEDURES MUST BE WRITTEN IN THE PLM86 LARGE MODEL.
- THE RMX86 LOADER REFERENCE MANUAL SUPPLIES MORE SPECIFIC INFORMATION ABOUT THESE PROCEDURES.

### CUSTOM TERMINAL SUPPORT

- THERE IS INTEL PROVIDED SOURCE CODE (MODIFIABLE TO YOUR NEEDS) FOR TERMINAL COMMUNICATION SUPPORT.
- YOU CAN ALWAYS WRITE YOUR OWN CODE.



### QUIZ!

- I CAN CHOOSE ANY FILE NAME TO BE LOADED? T OR F
- WHAT IS THE MAX NUMBER OF DEVICES I CAN SPECIFY IN AUTO SELECTION? _____
- WHAT ARE POSSIBLE ERROR CAUSES IF BOOTSTRAP LOADER LOOPS IN 2ND STAGE?



HINT -- LOOK IN THE LOADER MANUAL ! ! !





10/10/10

1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100

Software  
PUM R  
Passo  
Adal P  
TMM  
TMM

# intel[®] WORKSHOPS

## Self-Study Introduction to Microprocessors

Introduction to Microprocessors

MCS[®]-48/49 Microcontrollers

MCS[®]-51 Microcontrollers

MCS[®]-80/85 Microprocessors

iAPX 86,88,186 Microprocessors, Part I

iAPX 86,88,186 Microprocessors, Part II

iAPX 286 Architecture

## Software for Non-Programmers

PL/M Programming

Pascal Programming for Microcomputers

Ada[®] Programming

iRMX[™]86 Operating System Part I

iRMX[™]86 Operating System Part II (I/O)

iRMX[™]88,80 Operating System

## System 86/300 Users

NDS-II Network Development System Superuser

Transaction Processing System (iTPS)

Terminal Application Processing System (iTAPS)

iDBP[™] Database Processor

8086-Based Personal Computers

## Peripheral Chips/Data Communications

Speech Communication with Computers

2920 Signal Processor

### **Boston Area**

27 Industrial Avenue, Chelmsford, MA 01824 (617) 256-1374

### **Chicago Area**

Gould Center, East Tower

2550 Golf Road, Suite 815, Rolling Meadows, IL 60008 (312) 981-7250

### **Dallas Area**

12300 Ford Rd., Suite 380, Dallas, TX 75234 (214) 241-8087

### **San Francisco Bay Area**

1350 Shorebird Way, Bldg. B., Mt. View, CA 94043 (415) 940-7800



INTEL Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051 • (408) 987-8080