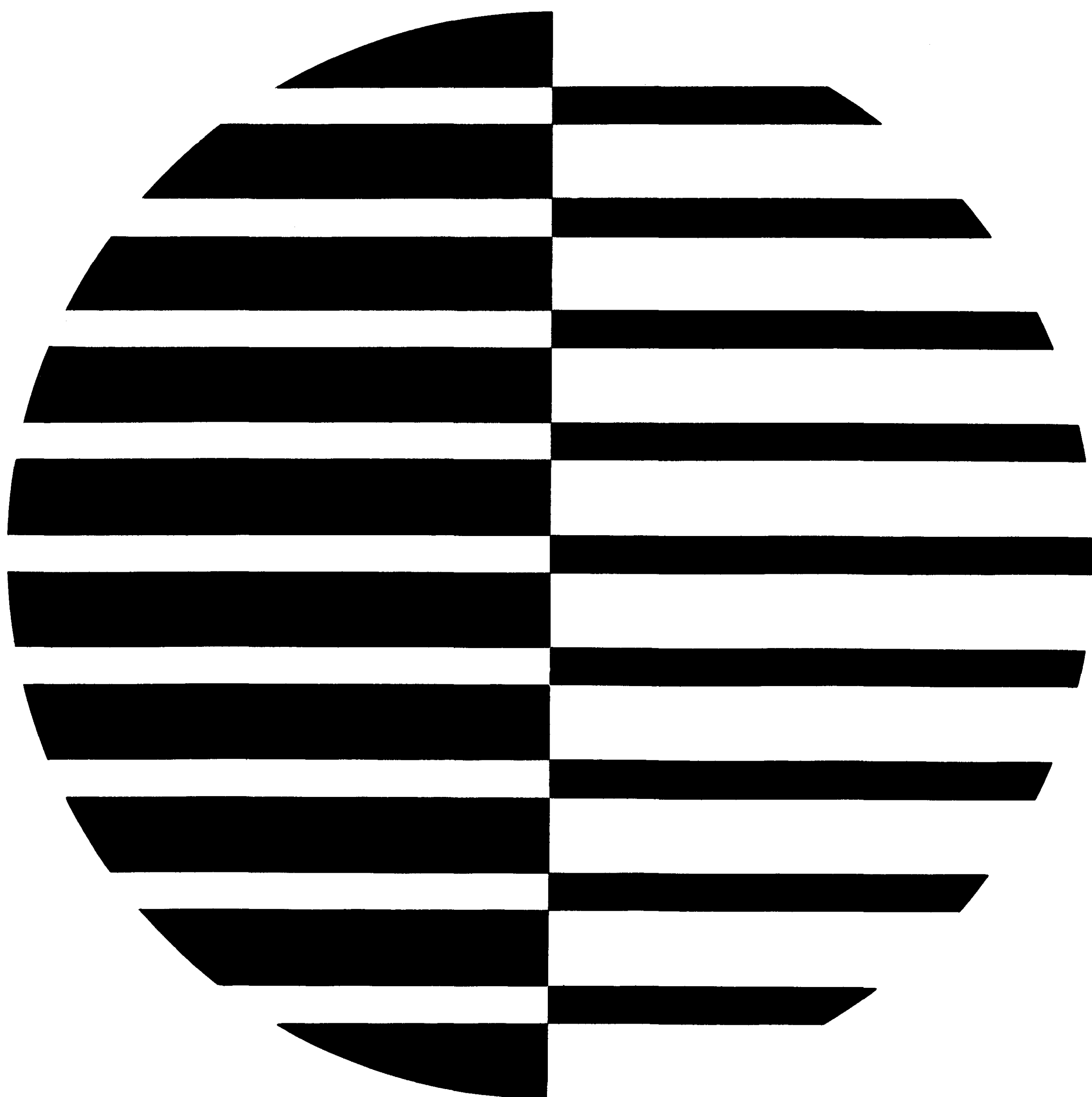
CONTROL DATA® 6000 SERIES COMPUTER SYSTEMS

INTERACTIVE GRAPHICS SYSTEM

Preliminary Reference Manual



RECORD OF REVISIONS	
Revision	Notes
01	Preliminary 6-27-69
02	Minor corrections on pages 7-39, 7-40, 7-42, J-1, J-2, J-3, and M-2;
	8-11-69
03	Changes have been made on pages iii, v, vi, vii, 1-3, 2-4, 2-5, 2-7, 2-15,
	2-16, 2-24, 2-25, 2-26, 3-18, 4-5, 4-7, 4-10, 4-11, 4-12, 4-13, 4-14,
	6-3, 6-6, 6-7, 7-8, 7-9, 7-12, 7-13, 7-14, 7-15, 7-16, 7-17, 7-18,
	7-19, 7-20, 7-21, 7-22, 7-23, 7-24, 7-25, 7-26, 7-30, 7-31, 7-32, 7-36,
	7-38, 7-39, 7-40, 7-41, 7-42, 7-43, 7-44, 7-45, 7-46, 7-50, 7-52, 7-56,
	7-57, 7-58, 7-59, 9-1, 9-2, 9-8, 9-9, 9-17, Gloss-2, Gloss-3, Gloss-4,
	B-1, B-2, B-3, B-4, B-5, B-6, B-7, B-8, C-1, C-2, F-1, H-3, H-6,
	I-1, I-2, J-1, J-2, L-3, M-1, M-2, N-1, N-2 to reflect the system
	design as of January 1, 1970.

Address comments concerning this manual to:

Control Data Corporation
Special Systems Division
Technical Publications
215 Moffett Park Drive
Sunnyvale, California 94086

6000 Series Interactive Graphics System
Publication Number 44616800
Copyright © Control Data Corp., 1969, 1970
Printed in the United States of America

PREFACE

This manual is a general programming and reference guide for the Control Data® 6000 Series Interactive Graphics System. It contains a summary of software operation, as well as its external characteristics. A background knowledge of Control Data 6000 Series software and hardware is needed to properly use this manual.

This book is organized so that an applications programmer can quick-reference programming information without sorting through the detailed background material provided for each subdivision of the system software; programming information does not occupy a separate section, but is isolated within the section that describes the software involved in that particular system function. The only information provided about the 6000 operating system concerns Interactive Graphics modifications to the system.

At the end of Section 7 is a subsection containing summaries and calling formats for the individual graphics routines of the system which are accessible to an applications programmer. In addition, the first section contains a general outline of software operation and appropriate hardware information.

The last section is a System operators' guide.

For more information related to the System's software, see the following publications:

Title	Publication Number
Control Data 6400/6500/6600 SCOPE 3 Reference Manual	60189400
Control Data 6400/6500/6600 SCOPE Operating Guide	60179600
Control Data 6400/6500/6600 Systems Reference Manual	60100000
Control Data 6400/6500/6600 Computer Systems FORTRAN Reference Manual	60174900
Control Data 6400/6500/6600 Computer Systems COMPASS Reference Manual	60190900
Control Data 6000 Series Interactive Graphics System General Information Manual	60237200

CONTENTS

1	INTRODUCTION	1-1
	Major Features	1-1
	Hardware Elements	1-2
	General Software Operation	1-4
	6000 Software	1-4
	1700 Software	1-8
	General Process Chart	1-9
	System Process Chart	1-10
2	6000 INPUT/OUTPUT AND GENERAL PROCESSING	2-1
	Control Points	2-1
	BATCHIO	2-1
	Routine Functions	2-1
	Combined EXPORT and BATCHIO Control Point	2-3
	K Display	2-5
	Dayfile Entries	2-5
	Job Output	2-6
	Job Input	2-7
	Graphics Program Card Deck	2-8
	Control Cards	2-8
	Program Cards	2-12
	Data Cards	2-13
	Sample Program Decks	2-13
	System Utility Functions	2-20
	Task File Creation	2-21
	Task Directory	2-22
	Task File Maintenance	2-24
	Graphics Program Aborting	2-25
	Scheduler	2-26
	Graphics Reformatter	2-26
	Scheduling of Graphics Control Points	2-27
	Routine Communication and Housekeeping	2-29
	Graphics Control Points	2-29
	Initialization	2-29
	Structure	2-30
	Number	2-30
	Size	2-31
	Files	2-31
	Graphics Common File	2-31
	Local Files	2-31
	Input Files	2-31
	Output Files	2-31
3	EXPORT/IMPORT AND DATA COMMUNICATION	3-1
	Introduction	3-1
	EXPORT	3-1
	Initialization	3-1
	Processing Control	3-2
	Communication Control	3-2
	EXPORT Servicing Cycle	3-3
	EXPORT Counters	3-4
	Data Transfer	3-4
	Character Set	3-9
	File Processing	3-10
	Termination	3-10
	Job Flow	3-10
	Initialization	3-10
	Input from Cards	3-10
	Input from Graphics Console	3-11
	Output to Graphics Console	3-11
	Output to Printer and Punch	3-11
	Error Detection Scheme	3-12
	IMPORT	3-13
	Routines	3-14
	Graphics Data Transfers	3-16

4 GRAPHICS HARDWARE INFORMATION 4-1
General Description 4-1
Graphics Console 4-1
Controls 4-2
Display Presentation 4-4
Potential Phosphor Damage 4-7
1744 Digigraphics Controller 4-7
Registers 4-7
Command Bytes 4-8
Control Bytes 4-10
Display Macros 4-13
Display Buffer Memory Layout 4-13

5 1700 GRAPHICS FUNCTIONS 5-1
Buffer Translator 5-1
Program Aborting 5-1
1700 Basic Graphics Package 5-1
System Expansion 5-2

6 DISPLAY ITEMS AND PICK PROCESSING 6-1
Display Item ID Block 6-1
Queue Handler 6-2
Pick Types 6-3
Queue Handler Functions 6-3
Fetch and Wait Queues 6-4
Queue Mechanism Operation 6-4
6000 Computer Pick Processing 6-7

7 6000 BASIC GRAPHICS PACKAGE 7-1
Routine Types 7-1
Graphics Hardware Interface 7-1
Application Executive 7-2
Graphics Utilities 7-3
Data Handler 7-3
Associative Addresses 7-9
Programming Conventions 7-10
Summary of User FORTRAN Callable Routines 7-10
Program Initiation 7-11
Program Console Control 7-11
Program Task Control 7-12
Special ID Block Assignment 7-13
Control of Queue Handler and Pick Processing 7-18
Fetching ID Blocks from Console Entries 7-21
Control of Console Alphanumeric Input 7-25
Frame-Scissoring Displays 7-26
Display Item Generation 7-29
Storing and Displaying Items 7-38
Control and Use of the Tracking Cross 7-44
Use of the Data Handler 7-47
Example of Bead Use 7-54
Voluntary Abortion of a Job 7-55
Hardcopy File Creation 7-56
Additional Routines for Display Font Creation 7-57

8 PROGRAMMING CONSIDERATIONS 8-1
Time Accounting 8-1
Memory Allotment and List Processing Efficiency 8-1
Data Handler Component Codes 8-1
Display Item Addresses 8-2
Macro Handling 8-2
Optimum Task Length 8-3
Non-Graphics Data Handler Use 8-3
Data Handler Common Files 8-5

9	SYSTEM OPERATOR'S GUIDE	9-1
	6612 Console	9-1
	Control Point Assignment and Release	9-1
	BATCHIO, B and K Displays	9-2
	EXPORT	9-5
	Dayfile/B Display Messages	9-7
	1700 Computer Console	9-8
	Initialization and Restart Procedure	9-8
	Communications Failure	9-9
	Control Type-Ins	9-10
	Output Messages	9-12
	Error Codes	9-16
	GLOSSARY	Glossary-1

APPENDICES

A	6000 Basic Graphics Package Routine Index	A-1
B	Graphics System Error Messages	B-1
C	Character Code Equivalents	C-1
D	Sample Data Handler File Dump	D-1
E	6000 Series Central Memory Word Organization	E-1
F	Card Formats	F-1
G	Cyclic Error Detection	G-1
H	Sample Graphics Programs	H-1
I	Differences Between 6000 Basic Graphics Package and 3000 Digigraphics Control Package Mark 4.0	I-1
J	Creating Alphanumeric Display Fonts	J-1
K	Hexadecimal/Octal Conversion Table	K-1
L	Re-entering a Graphics Task Overlay	L-1
M	System Packing of IBUF Description Buffers	M-1
N	Omission of Main from Program Coding	N-1
O	Coding Examples	O-1

FIGURES

1-1	Remote Class, Hardware Configuration	1-2
1-2	Intermediate Class, Hardware Configuration	1-3
1-3	Local Class, Hardware Configuration	1-3
1-4	Software Interactions	1-5
1-5	General Process Chart	1-9
1-6	System Process Chart	1-10
2-1	BATCHIO Control Point Field	2-4
2-2	File Creation Run Deck	2-15
2-3	UPDATE File Correction and Creation Deck	2-16
2.3.1	Run, Creation and Execution Deck	2-16
2-4	Task Addition Maintenance Run Deck	2-17
2-5	Task Replacement Maintenance Run Deck	2-18
2-6	Sample Deck to Purge and Store File	2-19
2-7	Sample Deck to Purge File within System	2-19
2-8	Execution Run Card Deck	2-20
2-9	Task Directory	2-23
2-10	Typical Time-Slice	2-29

2-11	Graphics Control Point Field	2-30
3-1	Conditions Present During One EXPORT Service Cycle	3-3
3-2	EXPORT Counters	3-4
3-3	EXPORT Graphics Transfer Buffers	3-6
3-4	Sample IMPORT Graphics Transfer Buffer	3-16
4-1	Function Keyboard	4-2
4-2	Alphanumeric Keyboard	4-3
4-3	Display Grid System	4-5
4-4	Sample Display Surface Organization	4-6
4-5	Display Buffer Block Diagram	4-14
6-1	Display Item ID Block in 1700	6-1
7-1	Typical Bead Arrangement	7-4
7-2	Four Cylinder Engine	7-5
7-3	List Structure Example	7-6
7-4	Data Handler File Block Structure	7-8
7-5	Example of a Frame-Scissored Arc	7-28
7-6	Example of Components in a Bead	7-54
7-7	Alphanumeric Display Font	7-58
7-8	Numeric Display Font	7-58
8-1	Sample Data Handler Batch Deck Using RFL	8-4
8-2	Sample Data Handler Batch Deck Using REDUCE	8-5
9-1	DSD 6612 K Display	9-3
G-1	Typical Encoder/Decoder	G-3
N-1	MAIN Communications Area	N-2

TABLES

2-1	Printer Format Control Characters	2-6
3-1	Status Word Codes	3-7
3-2	Directive Word Codes	3-17
4-1	Function Keyboard Status in IH, IV	4-3
4-2	Sample Frames	4-7
9-1	Equipment Mnemonics	9-4
9-2	Buffer Messages	9-4
9-3	EXPORT Messages	9-6
9-4	IMPORT Control Type-Ins	9-10
9-5	Job Location	9-12
9-6	Output Messages	9-13
9-7	Error Codes	9-16
M-1	IBUF/1744 Byte Comparison, Item Description Byte Generators	M-1
N-1	6000 Package External Linkages	N-3

INTRODUCTION 1

The Control Data 6000 Series Interactive Graphics System is designed to permit real-time use of a large computer by a graphics console operator – without degrading the capabilities of the machine.

Interactive Graphics accomplishes this by using a small machine, a Control Data 1700 Computer, to control the basic functions of the graphics hardware; the system uses the 6000 series computer only to handle more difficult manipulations and to do the mathematical work required by the applications programmer or the console user.

The Digigraphics 274 Display Consoles connected to the smaller computer permit the user to create, display, store, retrieve, and modify any graphic forms necessary for the active analysis of a problem – as well as giving him a means of entering data directly. These graphic forms can then be expanded or changed by the user in a real-time environment through his application program and the Interactive Graphics System.

The system can process the types of programs usually run in Batch-processing mode, but it eliminates the user waiting time of that mode, and provides a user with much greater flexibility in his use of the computer than batch-processing permits.

The system handles problems that:

- can best be represented in symbolic, graphic, or geometric form (such as schematics, diagrams, layouts, lattice structures, geologic cross-sections, and paths of motion)
- can best be described using mathematical functions (dynamic analyses)
- require human intervention (such as transcribing data for digital processing, empirical problem-solving, and geographic studies)

MAJOR FEATURES

Interactive Graphics includes these unique features:

- Graphics programming is done only on the 6000 Series computer – the 1700 Computer software operates without programmer intervention.
- Graphics programs can be written in standard FORTRAN 2.0, independent of display hardware characteristics.

- Data files can be tailored to fit the specific needs of an application programmer's job.
- Batch and graphic processing is performed concurrently; both types of jobs can be entered through the 1700 Computer, as well as at the 6000 Series site.
- Interactive Graphics can simultaneously service 24 independent graphics consoles through four 1700 Computers.
- The 1700 Computers are not dedicated to graphics work, but can perform other functions – even when graphics jobs are in the system.

HARDWARE ELEMENTS

The hardware configuration of the 6000 Series Interactive Graphics System is very versatile; the system can be configured for either remote, local, or intermediate operation. Figures 1-1, 1-2, and 1-3 show typical systems for each type of configuration.

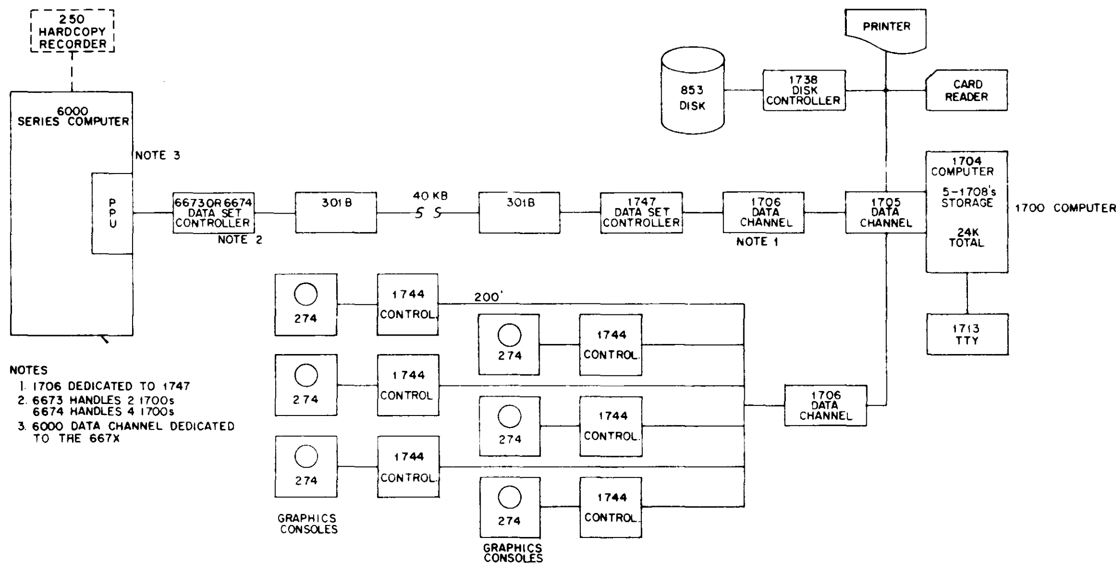


Figure 1-1. Remote Class, Hardware Configuration

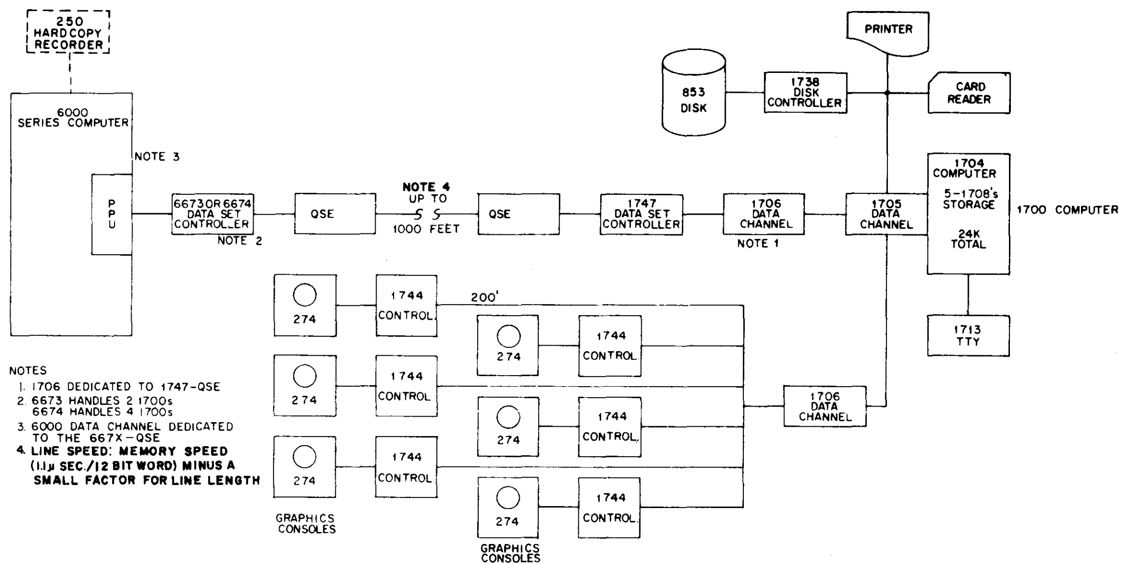


Figure 1-2. Intermediate Class, Hardware Configuration

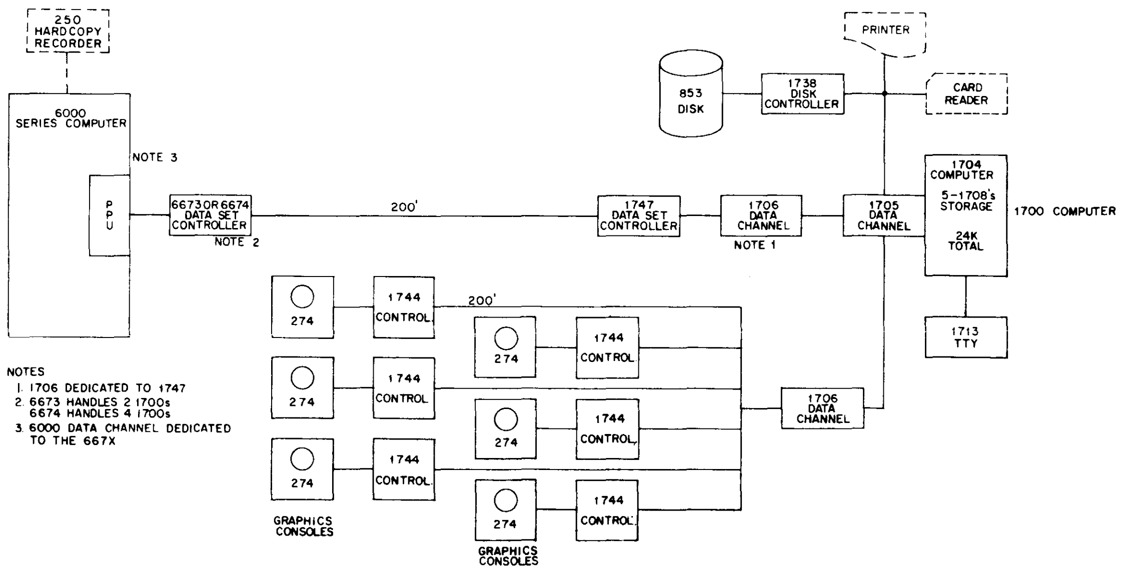


Figure 1-3. Local Class, Hardware Configuration

The full hardware capability of the Interactive Graphics System includes:

- Any standard 6000 Series hardware system, including a 6673 or 6674 Data Set Controller
- Four 1700 Computers, each with a 1747 Data Set Controller and at least 24K memory
- One hardcopy recorder
- One 1713 Teletypewriter per 1700
- One 853 or 854 Disk Pack per 1700
- One card reader per 1700
- One card punch per 1700
- One printer per 1700
- Six 274 Digigraphics Consoles per 1700

GENERAL SOFTWARE OPERATION

Interactive Graphics software operates as two separate but communicating groups of routines – one in the 6000 Series computer, the other in each of the 1700 Computers. Figure 1-4 shows the relationship between the groups.

6000 SOFTWARE

The 6000 Series portion of the System software includes:

- The SCOPE 3.1.2 operating system, with several added graphics features
- The standard FORTRAN 2.0 compiler
- The 6000 Basic Graphics Package, for actual graphics programming
- The Scheduler, to provide time-sharing for graphics programs
- An EXPORT program, for communication between the 6000 Series and 1700 Computers

SCOPE FEATURES

Because graphics programs require a real-time environment, they cannot be allowed to compete with batch jobs for the use of central memory control points; instead, one or two of

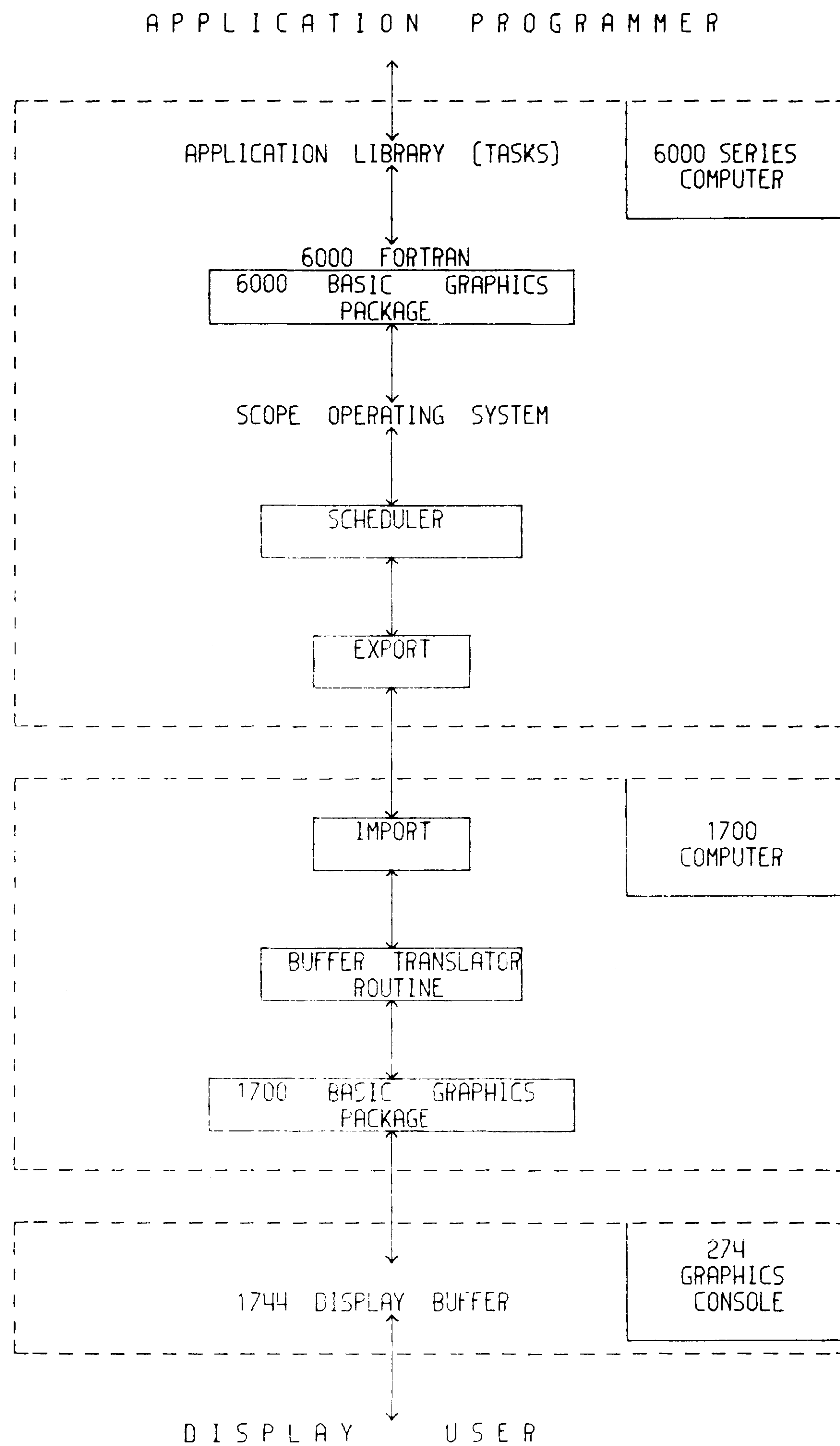


Figure 1-4. Software Interactions

SCOPE's control points are dedicated to graphics use. The number can be varied as needed by the 6000 operator, depending on the ratio of the graphics job load to the batch-processing load. If graphics programs are not being run, all control points can be made available for batch use.

The real-time requirements of graphics jobs also prohibit them from competing with batch jobs for central memory storage space. Therefore, each graphics control point has a certain amount of memory reserved for the use of graphics programs. This amount is chosen by the 6000 operator, and can be changed as needed for the most efficient distribution of memory between graphics jobs and batch jobs.

To further economize on the use of central memory, the SCOPE input/output package is replaced by a multi-device input/output package called BATCHIO. BATCHIO performs the same functions as the non-graphics package, and shares its control point area with EXPORT (the non-graphics package requires the use of separate control points). In addition, BATCHIO will read cards punched in three different coding systems.

The SCOPE library for Interactive Graphics includes three utility routines for the use of graphics programs:

- The task file creater, AEFILE
- The task file dump routine, AEDUMP
- The random-access file creator, AELOAD

Graphics programs are written as a series of overlays, each performing a task. The AEFILE routine places these overlays in mass storage as a random-access file with an index that can keep track of hundreds of overlays. The applications programmer can make additions to and deletions from this file; a task is located within the file and placed in central memory when it is needed (location and loading is performed by 6000 Basic Graphics Package routines).

The AEDUMP routine is used to remove unneeded information from the task file and to rewrite the file in a form that can be stored outside the system.

The AELOAD utility program is used to restructure the file produced by AEDUMP into a form that can be used as a task file.

All three utility routines are used at batch-processing control points so that file maintenance does not tie up the System's graphics-processing resources.

PROGRAMMING FEATURES

The 6000 Basic Graphics Package allows the user to write programs in FORTRAN without worrying about the maintenance of a display-oriented graphics data base or the mechanics of

communicating with the display. The Package contains an expandable library of subroutines that provide efficient and complete access to the graphics hardware (and two-way communication with it) without limiting application types or data structures. The Package is designed so that the programmer's only concern is communication with the graphics console operator and the computational requirements of the application; he is not aware of the internal functions of the Package, since there are no system-specified data areas that must be manipulated.

REAL-TIME MULTIPROGRAMMING

If several graphics programs are in the system at the same time, a form of time-sharing must be used so that each graphics console user believes that his program has sole use of the 6000 Series computer.

Graphics programs share their use of the 6000 Series central memory through a mechanism controlled by the Scheduler. The Scheduler looks at the programs waiting for execution in its graphics input queue, the graphics input request of currently executing programs, and at the programs themselves. The Scheduler then decides whether to roll out a program and roll in a new one from the input queue, or to roll in an old program that was rolled out while waiting for an input request to be serviced.

The Scheduler determines how long each program should be allowed to remain at a graphics control point on the basis of the central and peripheral processor time the program used when it last resided at a control point; this gives short graphics programs priority over longer ones. A lower limit, chosen by each installation, is imposed on the Scheduler's determination of a program's permitted resident time.

EXPORT FEATURES

EXPORT performs all data communication between the 1700 Computers and the 6000 Series computer. The Interactive Graphics version of EXPORT provides the same services for remote batch programs as the non-graphics version, and has several additional features:

- EXPORT is automatically loaded by BATCHIO whenever it is needed, rather than manually loaded as in the non-graphics version.
- EXPORT monitors the resident time of each graphics program, and calls the Scheduler into a peripheral processor when a program's permitted resident time has elapsed.
- EXPORT periodically scans each graphics control point for an input or output request and automatically transfers graphics output data to its own output buffers for transmission to the proper 1700.
- Graphics data from a 1700 is queued by EXPORT when it is received for later use by an application program (batch data is turned over to SCOPE for processing, as in the non-graphics version).

- EXPORT overlays are stored in Central Memory Resident, rather than in mass storage, to reduce the overhead time of data communication processing.
- EXPORT processes remote batch data and graphics data concurrently.

1700 SOFTWARE

The 1700 portion of the Interactive Graphics software consists of three groups of routines:

- An IMPORT program, to handle all communications between the 6000 Series computer and the 1700 Computer.
- The Buffer Translator
- The 1700 Basic Graphics Package

IMPORT FEATURES

The Interactive Graphics version of IMPORT 1700 has all of the data communication features of the non-graphics version, and interfaces with drivers to run a line printer, card reader, and card punch.

DATA TRANSLATION

The Buffer Translator reformats the graphics data buffers received by IMPORT from the 6000 Series computer into calls to the 1700 Basic Graphics Package. In this manner, data from the 6000 Basic Graphics Package is translated into a display-oriented data base. The Buffer Translator also formats data from the graphics consoles for transmission to the 6000 Series computer.

1700 GRAPHICS ROUTINES

The 1700 software includes a group of graphics routines called the 1700 Basic Graphics Package. These routines act like drivers for the graphics consoles, sending display information to the 1744 Controllers according to instructions received from the 6000 Basic Graphics Package calls. The 1700 routines also process interrupts and data from the graphics consoles, queueing the information until the program in the 6000 requests it. The application programmer does not use the 1700 Package routines when coding a job.

ADDITION OF SOFTWARE FUNCTIONS

Additional 1700 functions can be incorporated in the Interactive Graphics System without altering the existing software; the 1700 can be used to drive remote devices for specific applications, without hardware modification, other than the addition of memory.

GENERAL PROCESS CHART

The General Process Chart in Figure 1-5 follows a user's program through the Interactive Graphics System and shows the relationships between the hardware and software at various stages in the program's processing.

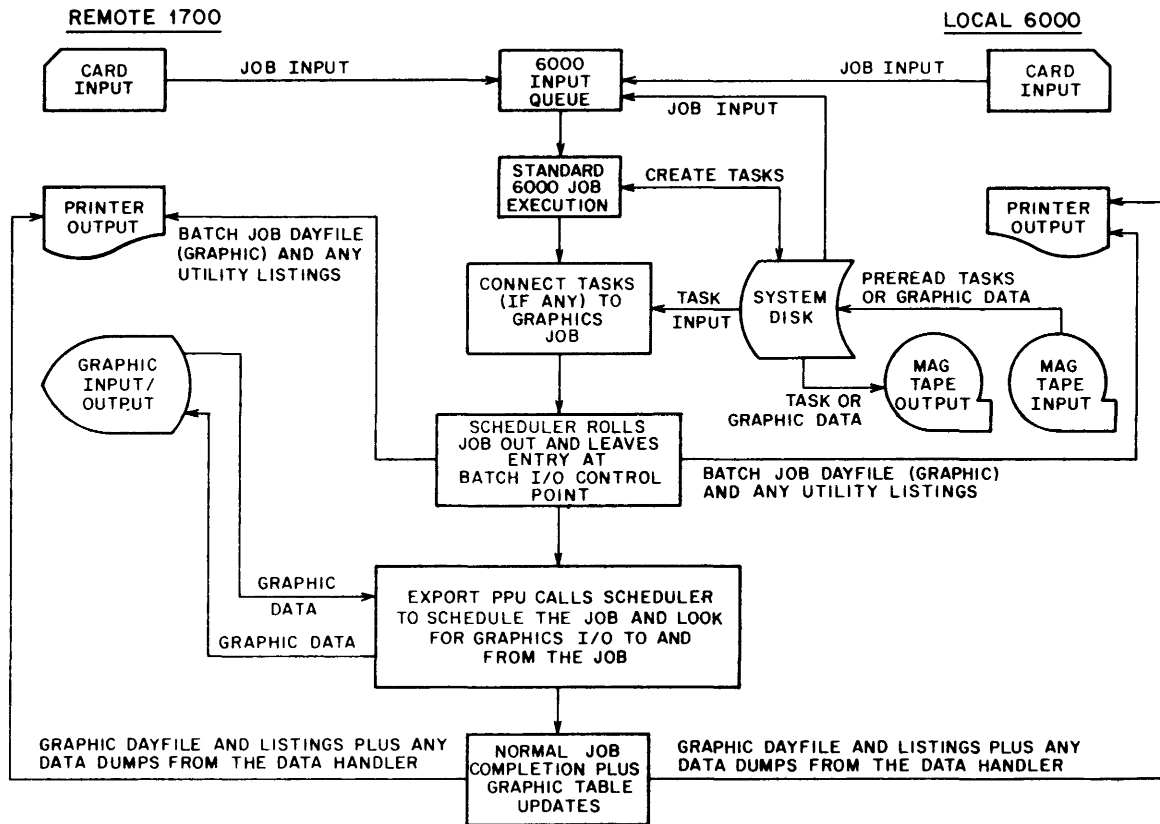


Figure 1-5. General Process Chart

SYSTEM PROCESS CHART

The System Process Chart in Figure 1-6 also follows a program through the System, and shows in more detail the interaction of the parts of the software with the hardware. This chart is a schematic of the flow of data through the System during graphics program processing.

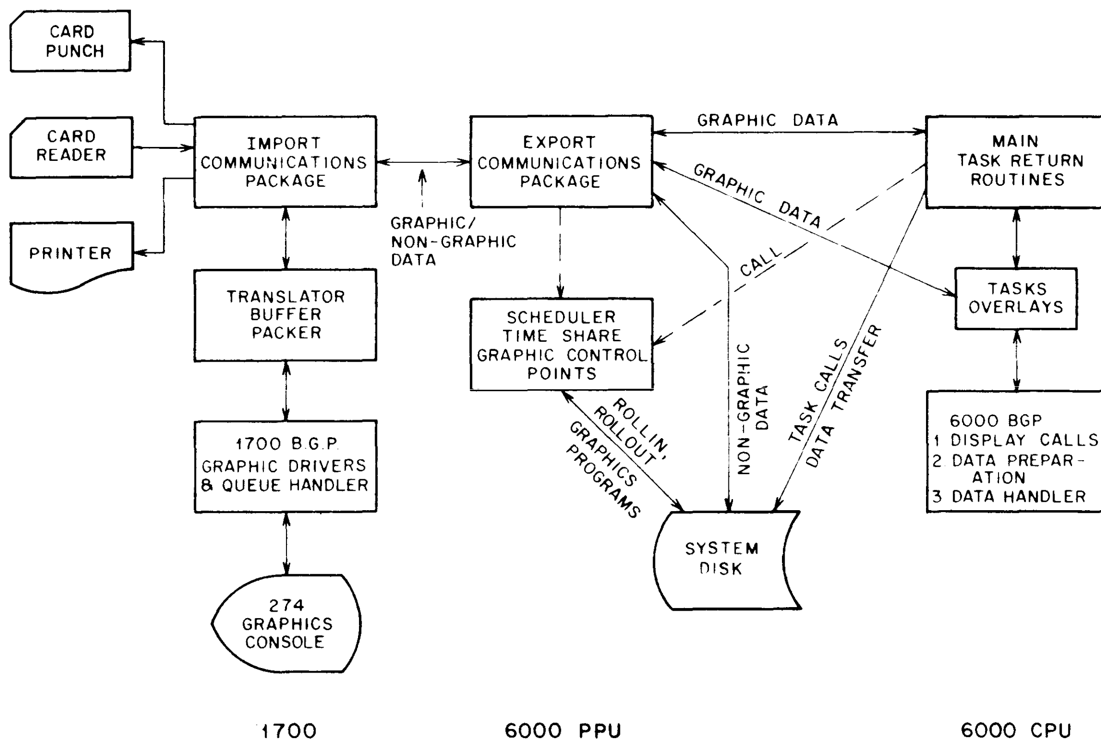


Figure 1-6. System Process Chart

6000 INPUT/OUTPUT AND GENERAL PROCESSING 2

CONTROL POINTS

Two to three of the seven control points provided by the standard SCOPE operating system are reserved in the modified form of the system used by Interactive Graphics. One to two can be designated as graphics control points by the installation and are then used exclusively for graphics programs, although the system makes them available on command for batch use. The third control point is reserved for the combined use of the BATCHIO and EXPORT programs.

BATCHIO

BATCHIO can simultaneously drive up to seven of the following Control Data devices in any combination, through each peripheral processor it uses.

- 501 or 505 Line Printers
- 415 Card Punches
- 405 Card Readers

The BATCHIO package consists of three peripheral processor primary overlay programs, plus ten secondary overlays; these programs are assigned to pool PPU's as needed. The primary overlays include:

- 1IO (BATCHIO Manager Program)
- 1CD (Input/Output Driver)
- 1PS (Service Program)

ROUTINE FUNCTIONS

1IO

The 1IO routine monitors the input devices and the SCOPE output queue. When it detects a need for input or output action, it assigns an appropriate available device to the 1CD driver and associates the device with one of the 16 buffers in the BATCHIO control point field. 1IO then assigns the proper output file to 1CD.

If 1CD is not currently running when 1IO detects a need for input or output action, it loads 1CD into another pool PPU. 1IO also transfers 6612 operator END, REPEAT, or SUPPRESS type-in commands (see Section 9) to the appropriate buffer area for execution by 1CD; 1IO goes into recall every three-quarters of a second.

1CD

The PPU used by 1CD remains dedicated to it while any input or output activity is occurring. In addition, the driver calls transient PPU's to perform 1PS functions and manage buffers.

The 1CD routine:

- Reads job input files from the card reader, performs code conversion and checksumming, and places the file in the SCOPE input queue. 1CD will perform a reread operation when a card read compare error is encountered, and displays error messages when a checksum or validity error occurs.
- Reads print files from the disk, performs code conversion, executes END, REPEAT, and SUPPRESS type-in commands, and prints files. 1CD provides printer status messages on the K display.
- Reads PUNCH files from the disk, code converts them if necessary, executes END and REPEAT commands, and punches the files. 1CD provides card punch status messages on the K display, and will repunch any cards that have compare errors.
- Uses standard SCOPE circular buffering, generates request stack entries for disk access, and calls 1PS to perform housekeeping functions.

1CD goes into recall when all input and output activity is finished.

1PS

1PS is called into another PPU by 1CD to perform the following actions:

- Call 2TJ to translate job cards and create File Name Table entries for input files.
- Call 2DF to drop output files after they have been printed or punched.
- Access the dayfile to execute an END type-in command (see Section 9).
- Rewind print files and call 2LD to print a banner page (see Job Output).
- Rewind punch files and call 2CD to generate LACE card data.

COMBINED EXPORT AND BATCHIO CONTROL POINT

ASSIGNMENT

The 6612 operator has the option of assigning BATCHIO and EXPORT to a control point either automatically or manually (see Section 9). If he does it automatically, the two programs will be assigned to control point one and the contents of the control point field will be preserved in the event of a partial recovery dead start.

If the programs are assigned manually to any other control point, the contents of the control point field will be lost when a partial recovery dead start is performed.

BATCHIO cannot be assigned to run at more than one control point.

INITIALIZATION

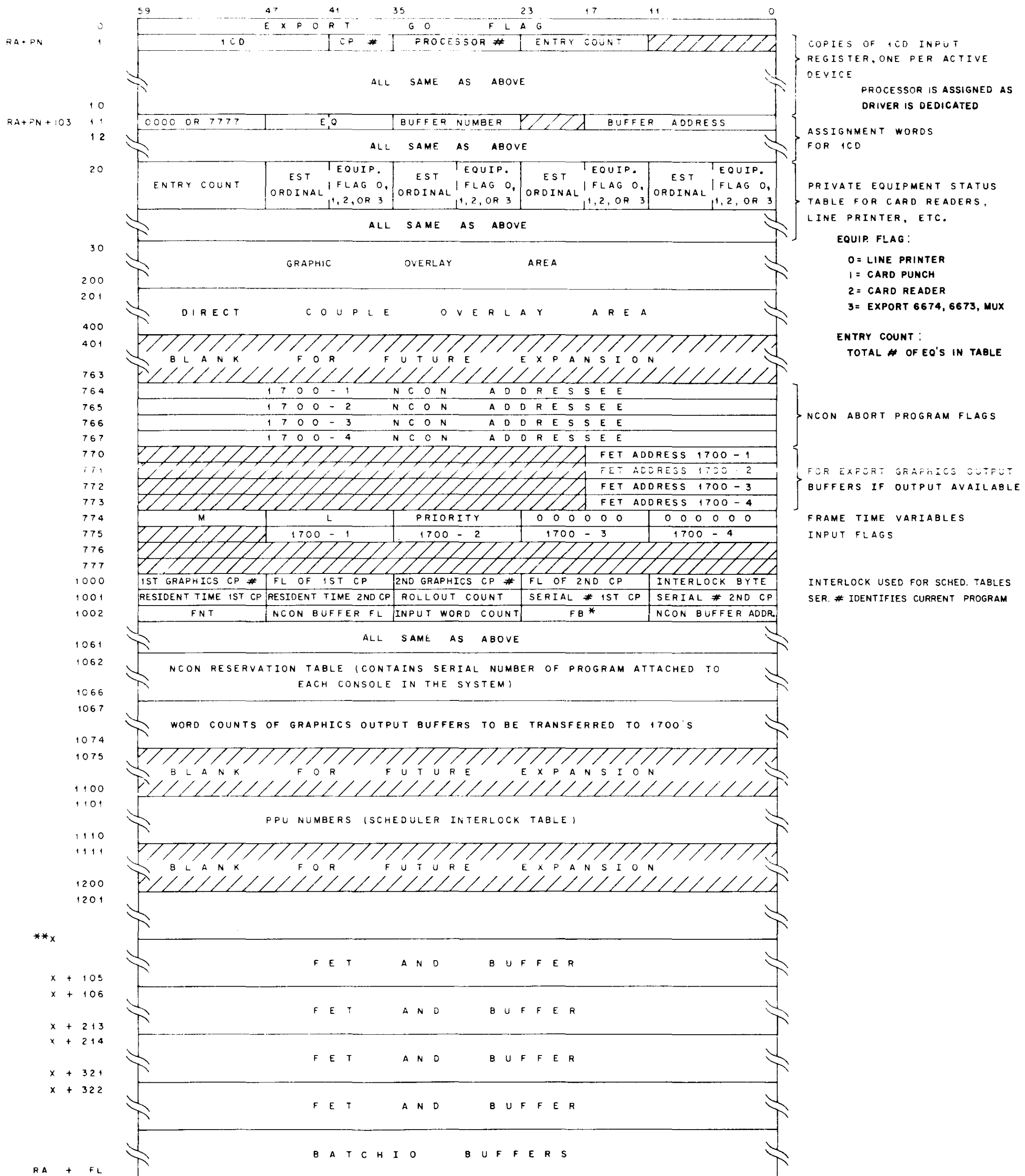
After the control point is assigned, the BATCHIO initialization routine is loaded into a PPU and structures the three parts of the control point field. These parts are:

- Message buffers, flags, and pointers; used by BATCHIO and EXPORT for inter-communication.
- Area used for graphics; contains status and Scheduler tables for graphics control points at which programs are being executed, and contains communication buffers for the graphics consoles of each 1700 Computer. Multiple buffers of data fit in each of these communication buffers.
- Sixteen BATCHIO file environment tables and buffer areas; these are used by the BATCHIO driver and by EXPORT for actual input and output.

After BATCHIO is loaded, it waits for a code signal from the 1700 Computer. This signal tells the 6000 Series machine that the 1700 is operational, and causes EXPORT to be loaded into another PPU by BATCHIO.

STRUCTURE AND USE

Figure 2-1 shows the structure of the BATCHIO/EXPORT control point field. The sixteen buffer areas are shared by BATCHIO and EXPORT. EXPORT assigns as many areas to each 1700 as are needed. When BATCHIO or EXPORT finishes using a buffer, it is returned to the pool of idle buffers for reassignment.



*FB = FLAG BITS FOR STATUS OF THIS NCON. IF FB = 1XX, NCON AVAILABLE; IF FB = 2XX OR 3XX, NCON ABORTED; IF FB = 4XX, AVAILABLE VACATED FILE; FB = 1XXX THRU 7XXX IS A DYNAMIC PRIORITY WEIGHT.

**X = 1201 + 100_8 * NUMBER OF CONSOLES

Figure 2-1. BATCHIO Control Point Field

K DISPLAY

A SCOPE DSD display presents the status of the sixteen BATCHIO buffer areas and any messages associated with them. This K display is shown in Figure 9-1.

EQUIPMENT MNEMONICS

A mnemonic is provided for each piece of equipment which can be accessed through the BATCHIO control point. These mnemonics appear on the K display when the devices are using the buffers. The mnemonics valid for this system are listed in Table 9-1.

BATCHIO BUFFER MESSAGES

BATCHIO produces the messages in Table 9-2 on the K display when it encounters abnormal conditions in the hardware it services. Each message appears in the message area of the buffer used by the device; the last message to appear on the K display also appears on the third line of the BATCHIO control point entry for the B display (see Section 9).

EXPORT messages do not appear on the K display.

DAYFILE ENTRIES

STANDARD

A dayfile entry is made for every read, print, or punch operation. This entry gives the job name and card or line count for accounting purposes.

DIAGNOSTICS

BATCHIO makes dayfile entries for certain equipment conditions at dead start time. These entries may not require operator action, and do not appear on the K display; they appear only on the third line of the B display and in the dayfile.

RESERVED MESSAGE

The dayfile entry:

    EQxx, CHyy, RESERVED. TURNED OFF.

indicates that the piece of equipment with the Equipment Status Table ordinal xx on data channel yy has a hardware reserved status condition. This status is used only with dual access controllers and indicates that the alternate controller is using the device. Device xx is automatically turned off in the Equipment Status Table.

REJECT MESSAGE

The dayfile entry:

EQxx, CHyy, REJECT. TURNED OFF.

indicates that the device with the Equipment Status Table ordinal xx on channel yy has returned a reject status. The device is automatically turned off in the table.

JOB OUTPUT

PRINTER FORMAT AND CONTROL CHARACTERS

A banner page, consisting of the file name in large characters, is printed at the beginning of each file output. Files are printed until the End-of-Information on the disk is reached; no printed indication of an End-of-File or End-of-Record is given.

A line of print may be 136 characters long. The first character is not printed if it is one of the format control characters in Table 2-1, which follows:

TABLE 2-1. PRINTER FORMAT CONTROL CHARACTERS

Character	Operation
/	Print on next line
0	Skip 1 line before printing
-	Skip 2 lines before printing
1	Print on top of next page
2	Advance to last line on page before printing
↓	Skip to printer format channel 1 after printing
<	Skip to printer format channel 2 after printing
>	Skip to printer format channel 3 after printing
≤	Skip to printer format channel 4 after printing
≥	Skip to printer format channel 5 after printing
¬	Skip to printer format channel 6 after printing
8	Skip to printer format channel 1 before printing
7	Skip to printer format channel 2 before printing
6	Skip to printer format channel 3 before printing
5	Skip to printer format channel 4 before printing
4	Skip to printer format channel 5 before printing
3	Skip to printer format channel 6 before printing
R	Set Auto Eject mode. (The perforation at the top or the bottom of the page is skipped automatically.)
Q	Clear Auto Eject mode. (Print continuously from the top to the bottom of each page.)
+	Supress paper advance before print. (Overprint the last line with this line.)

The printer remains in Auto Eject mode until a control character is encountered, and returns to it after that line has been printed. Printing occurs on consecutive lines in Auto Eject mode.

CARD PUNCHING FORMATS

A LACE card is punched at the beginning of each file. It contains the job name associated with that file, and is similar in appearance to the banner page produced by the printer.

End-of-Record cards are punched without level numbers.

LACE, End-of-Record, End-of-File, and cards with compare errors are offset for easy recognition.

Data cards are punched in one of five formats (see Appendix F):

- End-of-Record cards (level zero only)
- End-of-File cards
- Normal mode binary cards
- Free-form mode binary cards
- Standard 6000 Hollerith coded cards

JOB INPUT

HOLLERITH TYPES

BATCHIO can read coded cards that have been punched in one of two versions of Hollerith code:

- Standard 6000 Hollerith
- ICT 1900 Hexadecimal Hollerith

These codes are given in Appendix C. Coded cards are assumed to be in standard 6000 Hollerith code until a Hollerith Switch card is encountered. The format and use of Hollerith Switch cards is described in Appendix F.

CARD READING FORMATS

BATCHIO reads cards that have been punched in either of two modes (see Appendix F). BATCHIO begins reading each card file in Normal mode, and while in that mode can read:

- Standard 6000 binary cards
- End-of-Record cards
- End-of-File cards
- Enter Free-form cards

- Hollerith Switch cards
- Coded cards

Coded cards are assumed to be in standard 6000 Hollerith code until a Hollerith Switch card is encountered.

In Free-form mode, BATCHIO can read:

- Free-form mode binary cards
- Exit Free-form mode cards
- Absolute End-of-File cards (also serve as Exit Free-form mode cards)

CARD TRANSLATION

If BATCHIO detects a character for which no internal display code equivalent exits, it will translate the character as a $ (53 in internal display code); there is no maximum permissible number of such validity errors.

If BATCHIO encounters a Mode changing card or Hollerith Switch card that it does not recognize, it produces the message:

NOT RECOGNIZED

in the program's output file, followed by the image of the card. The job is then aborted.

Error messages produced during card translation appear nowhere in the system except in the program's output file.

GRAPHICS PROGRAM CARD DECK

User programs are compilea and executed on the 6000 Series computer system. The user submits his application program as a normal FORTRAN batch job card deck; the following discussion assumes that the card deck is punched in standard 6000 Hollerith code.

The card deck consists of control cards, program cards, and data cards. The control cards specify how the job is to be processed, and are followed by the FORTRAN program cards and the data cards. The deck ends with an End-of-File card (6-7-8-9 quadruple-punched in column one).

CONTROL CARDS

JOB CARD

The first control card, the Job card, must indicate the job name, priority, central processor time limit, and memory requirements of the program. Fields are separated by commas, and the last field is terminated by a period. Fields other than the job name may appear in any order. All capitalized letters must appear on the card; they are required by SCOPE.

n, Pp, Tt, CMfl, ECfl.

n	Alphanumeric job name, which begins with a letter and is 1 to 7 characters long.
Pp	Equals priority level in octal, with a 1 as the lowest priority; the upper limit on p is an installation option.
Tt	t equals central processor time limit for the whole job, including compilation and execution, in seconds and is 1 to 5 octal digits.
CMfl	fl equals total central memory field length of the job, with a maximum of 6 octal digits.
ECfl	fl equals total extended core storage field length required in terms of 1000_8 word blocks, with a maximum of 7777_8; this parameter maybe omitted.

RUN CARD

The RUN card is usually the second control card in the deck. It calls the FORTRAN compiler, and provides Compiler mode, field length, and file names as follows:

RUN (cm,fl, bl, if, of, rf, lc, as, cs)

cm =	G	Indicates compile and execute without a list, unless explicit LIST cards appear in the deck.
	S	Indicates compile with source list but do not execute; if execution is desired, LGO card must follow RUN card.
	P	Indicates compile with source list and punch deck on file PUNCHB, but no execution.
	L	Indicates compile with source and object list, but no execution.
	M	Indicates compile with source and object list; produce a punch deck on file PUNCHB, but do not execute.

fl	Object program field length in octal; if omitted, it is set equal to the field length at compile time.
bl	Object program I/O buffer length in octal; if omitted, it is assumed to be 2022.
if	File name for compiler input; if omitted, it is assumed to be INPUT
of	File name for compiler output; if omitted, it is assumed to be OUTPUT.
rf	File name on which the binary information is always written; if omitted, it is assumed to be LGO.
lc	Octal line-limit of an object program on the OUTPUT file. If the line count exceeds the specified line limit, the job is terminated; if omitted, it is assumed to be 10,000.

as — ASA switch. If nonzero or nonblank, it causes the ASA I/O list/format interaction at execution time.

cs — Cross-reference switch. If nonzero, a cross-reference listing is produced.

LGO CARD

This card calls the SCOPE General Purpose System Loader and begins program execution – regardless of the parameters on the RUN card. The format for this card is:

```
LGO.
```

AEFILE CARD

The AEFILE card calls the graphics task file creation utility routine, AEFILE, which restructures the program file identified by the second data card into an indexed random-access COMMON file of primary overlays, identified by the name on the first data card. AEFILE can also be used to add or replace overlays in the file, and make changes within overlays.

This card has the format:

```
AEFILE.
```

AEDUMP CARD

This card calls the AEDUMP utility routine, which reads a random-access file (the graphics task file), removes all rewritten records and indexes, and writes it as a serial-access file with an index as its first record.

The format for this card is:

```
AEDUMP (s, o)
```

s — Name of the random-access file to be used as a source; this is the file created by AEFILE or AELOAD.

o — Name of the serial-access file to be produced.

AELOAD CARD

The AELOAD card calls the AELOAD utility routine, which reads a serial-access file (the file produced by AEDUMP) with an index as its first record. AELOAD then writes the file as a random-access file with an index of disk addresses as its last logical record. The file produced by AELOAD can be used as a graphics COMMON file.

The AELOAD card has the format:

```
AELOAD (s, o)
```

s — Name of the serial-access file to be read as a source; this is the output file of AEDUMP

o — Name of the random-access file to be produced

COMMON CARD

This card attaches any existing COMMON file named in its parameter field to the program, and changes its status in the File Environment Table so that no other program will have access to it while the current program is running. When the program the file is attached to terminates, the file is returned to the system and may be reassigned by another program's COMMON card. The COMMON card's format is:

```
COMMON, fn.
```

fn — Name of the COMMON file (usually the graphics task file created by AEFILE or AELOAD) to be assigned to the program.

RELEASE CARD

The RELEASE card eliminates the COMMON file named in its parameter field from the system. When SCOPE encounters a RELEASE card, it changes the file's File Environment Table/File Name Table entry so that the file is reclassified as a local program file. When the program ends, all of its local files are automatically destroyed. This card has the format:

```
RELEASE, fn.
```

fn — Name of the COMMON file (usually the graphics task file) to be destroyed.

EXIT CARD

When SCOPE detects a program error, it searches the program's control card record for an EXIT card. If it finds one, it performs any actions specified by the control cards following the EXIT card, then terminates the program.

If an error occurs and no EXIT card exists, SCOPE simply terminates the job with a dayfile message.

If no error occurs, an EXIT card (and any control cards following it) is ignored. The EXIT card format is:

```
EXIT.
```

or

```
EXIT (S)
```

If the S parameter is used, EXIT processing is also done when assembly or compilation errors cause termination.

PROGRAM CARDS

Several program cards are required by Interactive Graphics. Program cards are separated from control cards and data cards by End-of-Record cards (7-8-9 triple-punched in column one), and are punched as standard FORTRAN cards.

MAIN (ZERO-LEVEL) OVERLAY CARD

This card causes the FORTRAN compiler to translate the program overlay following it as a zero-level overlay. Zero-level overlays always reside in core when the program is at a control point, and serve to link blank COMMON areas between higher level overlays.

The main overlay card has the format:

```
OVERLAY(lfn, 0, 0)
```

lfn   Name to be assigned to the source file of overlays (produced by the SCOPE General Purpose System Loader).

CALL MAIN CARD

This card calls the Application Executive MAIN program; when encountered at compile and loading time, it causes the Executive's MAIN program to be loaded into the zero-level overlay as a subprogram from the SCOPE system library.

If this card is not used, the programmer must supply his own executive zero-level overlay to set up a call to AEFILE, load tasks, fetch buttons, and so forth. This card has the format:

```
CALL MAIN
```

TASK LEVEL OVERLAY CARD

This card is used to begin each task overlay and serves as an End-of-Record card for the overlay preceding it. It has the format:

OVERLAY (p, s)

p — Primary overlay level number in octal; must be greater than zero and less than 100_8.

s — Secondary overlay level number in octal; must be positive and less than 100_8.

Overlays need not be numbered sequentially in an input file; however, no overlay may have a primary number smaller than that of the overlay preceding it, and no overlay may precede another with the same primary number but a smaller secondary number.

DATA CARDS

If a graphics program uses the Application Executive MAIN program, there must be at least one data record in its deck.

The first data record contains the file name parameter cards used by the Executive's MAIN program. The file names on these cards are standard seven character alphanumeric names, starting in column one of the card. The first card must contain the name assigned to the graphics COMMON file; the second card (used only during a file creation run) must contain the name of the file produced by the General Purpose System Loader. This source file name must agree with the name given on the program's main overlay card (see above).

SAMPLE PROGRAM DECKS

Figures 2-2 through 2-8 depict program decks for various task file creation, maintenance, and execution functions. The operation of the system utility routines called by the control cards is explained in more detail later in this section.

ZERO-LEVEL OVERLAY CONTENT

The zero-level overlays in all runs of a job must be identical. The overlays in the task file are linked to FORTRAN and Application Executive entry points within the zero-level overlay and are relocated with respect to the first word address of the zero-level overlay's blank COMMON. Unless the same zero-level overlay is used for all runs, task loading and COMMON linkage will not occur properly.

If the zero-level overlay, the size of blank COMMON, or the number of files used is changed, a new file creation run should be made to alter the linkages and loading addresses for each of the tasks in the task file.

If the name of a file or a blank COMMON location is changed without changing the zero-level overlay's core requirements, it is necessary only to change the task overlays affected by the name changes; this can be done with a file maintenance run.

All file requirements (such as INPUT, OUTPUT, or TAPE6) must be listed on the zero-level overlay's PROGRAM card; they may not appear on a PROGRAM card in any other overlay.

The FORTRAN compiler allocates File Environment Table entries and buffers for these files and sets pointers to the allocations for use during the execution run. Each subsequent allocation of a file with a given name is written over the previous one, so that a file listed in the zero-level overlay and in another overlay will have pointers only in the latter. Therefore, when the zero-level overlay is entered at execution time, the FORTRAN linkage routine will try to find a File Environment Table entry for file but will fail; the pointers that it searches for will be unavailable because they are in an overlay that has not yet been loaded.

When the linkage routine's search fails, the job is aborted with the diagnostic message:

```
NO OUTPUT FILE FOUND
```

This portion of a program would cause such a diagnostic:

```
OVERLAY (SOURCE, 0, 0)
PROGRAM ONE (TAPE6)
    •
    •
    •
OVERLAY (1, 0)
PROGRAM TWO (TAPE6)
    •
    •
    •
```

FILE CREATION RUNS

Figure 2-2 shows a typical card deck for an initial file creation run, using the Application Executive MAIN program and the system AEFILE routine. This deck can create a file with a maximum of 63_{10} primary or secondary overlays.

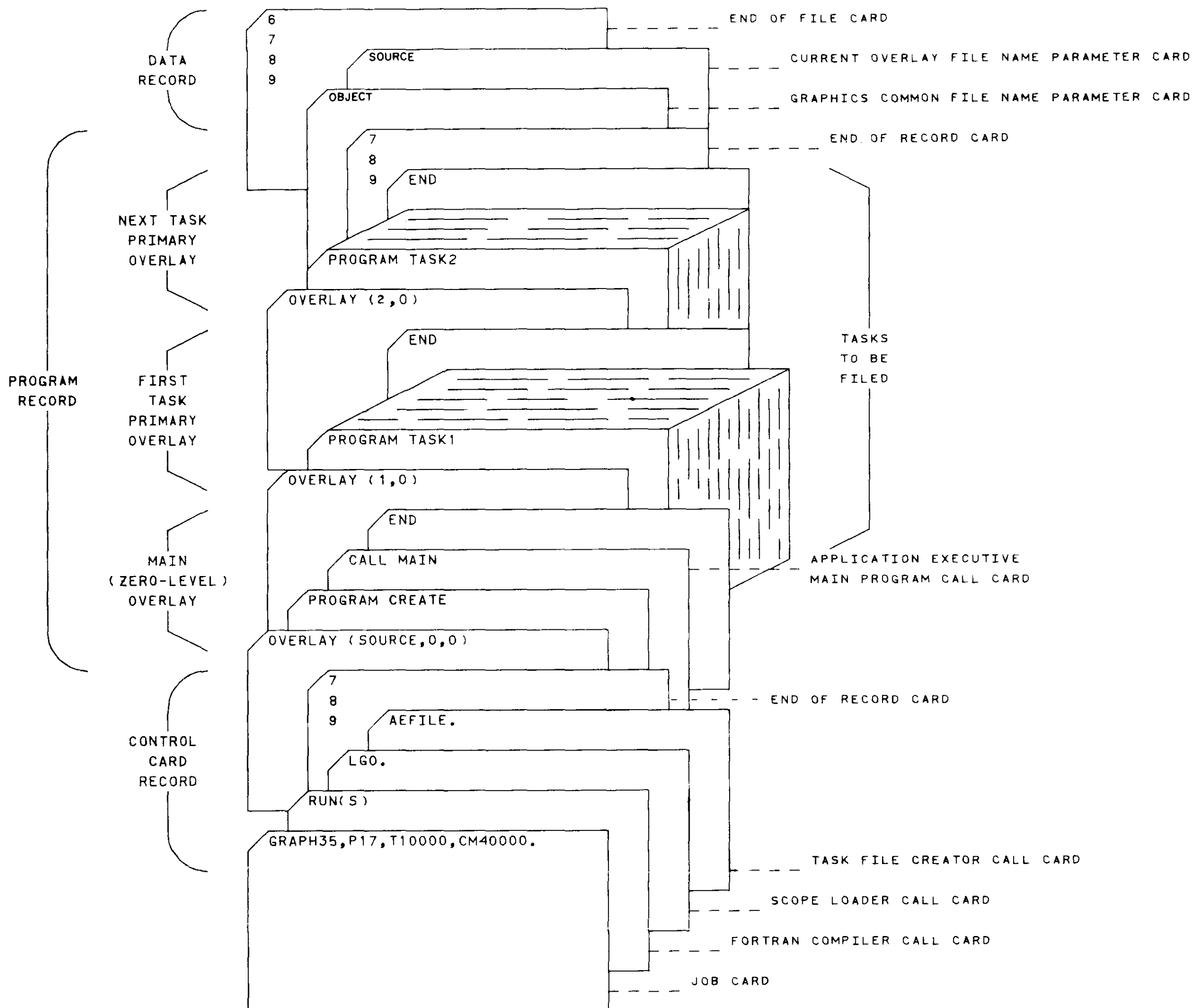


Figure 2-2. File Creation Run Deck

A graphics COMMON file containing more than 63_{10} overlays can be created. A deck, such as the one shown in Figure 2-4, can be used to build a task file that contains as many overlays as the installation-specified limit MNOVL will permit (see AEFILE routine).

FILE MAINTENANCE RUNS

If a program library has been created for graphics jobs, and it has the same format as the sample deck shown in Figure 2-2, then a task file can be created from it. By using the system UPDATE program, the programmer can make corrections during the same run. Figure 2-3 shows a deck that will form a corrected task file from an UPDATE library tape; the routines in card deck LBTASK will be placed in the file SOURCE from tape OLDPL, and task file OBJECT will be produced.

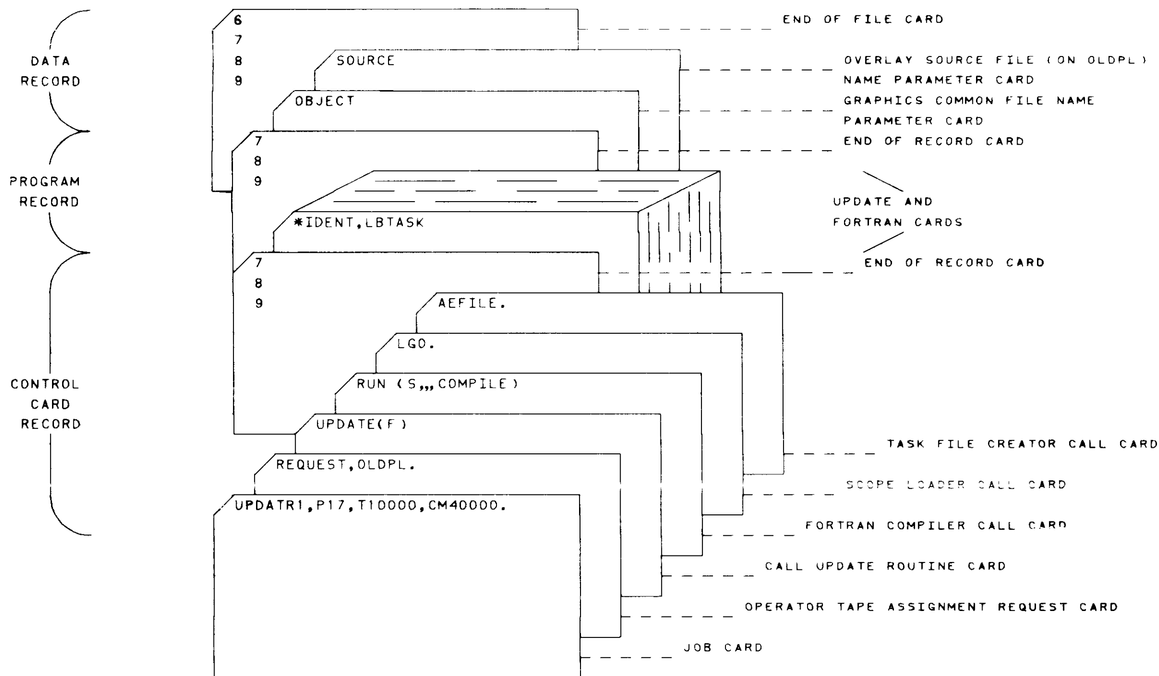


Figure 2-3. UPDATE File Correction and Creation Deck

This deck is used to create, run, and execute the program in one pass through the computer.

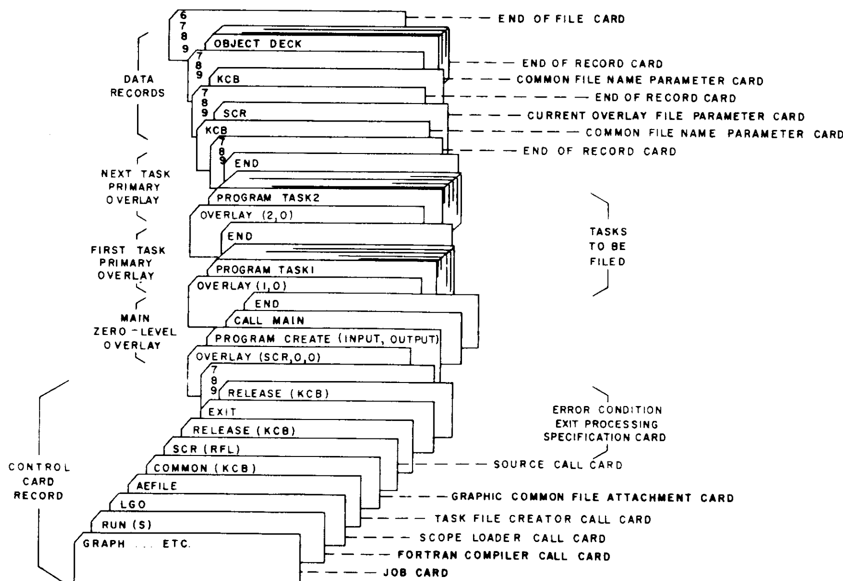


Figure 2-3.1. Run, Creation, and Execution Deck

Task overlays can be added to an existing graphics COMMON file by using AEFILE. Figure 2-4 shows a sample deck which adds a primary level overlay ADDTASK to the end of the file created by the deck in Figure 2-2.

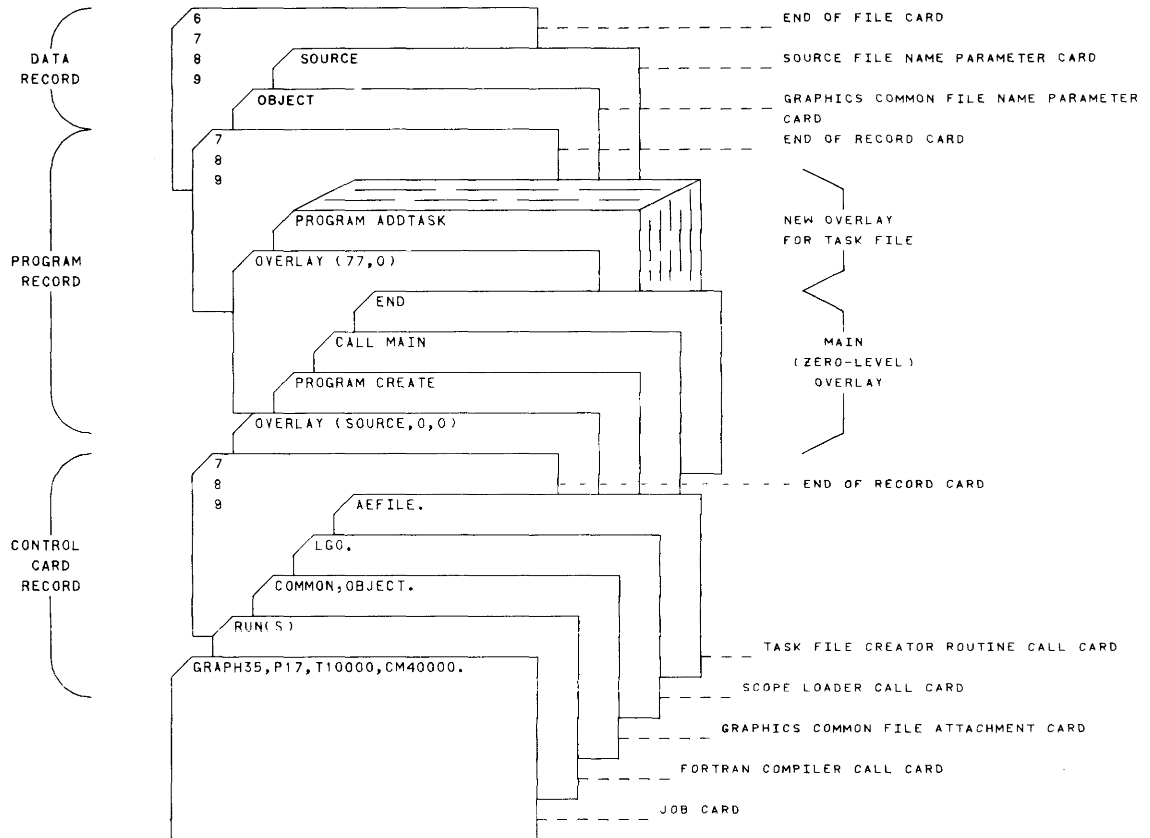


Figure 2-4. Task Addition Maintenance Run Deck

Task overlays may also be replaced within a graphics COMMON file by using AEFILE. Figure 2-5 shows a sample deck that will substitute the revised primary overlay TASK1 for the original primary overlay TASK1 in the file created by the deck shown in Figure 2-2. The substitution is made according to the name given on the new task's PROGRAM card - the new task will replace the old task with the same name within the file.

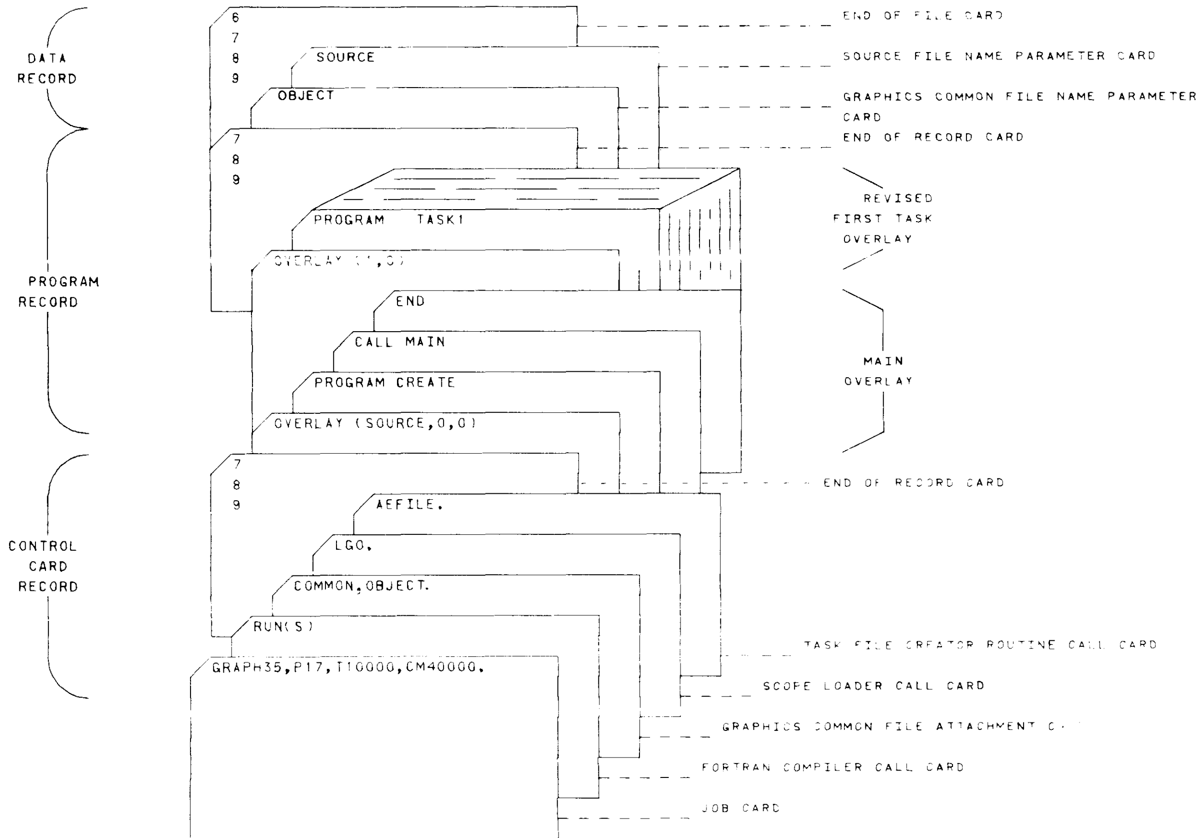


Figure 2-5. Task Replacement Maintenance Run Deck

Figure 2-6 shows a deck that will take the file OBJECT created by any of the preceding decks and store it in a purged form on magnetic tape as a file called SOURCE.

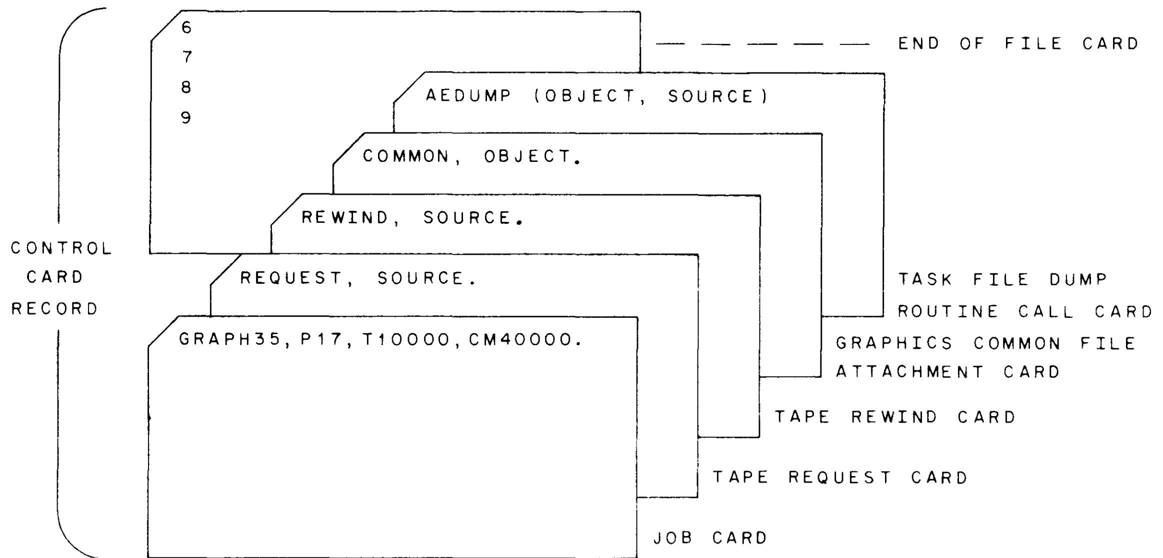


Figure 2-6. Sample Deck to Purge and Store File

Figure 2-7 shows a sample deck that purges the file OBJECT1 (similar to OBJECT of Figures 2-2 through 2-5) and recreates it as file OBJECT for use in a subsequent execution run.

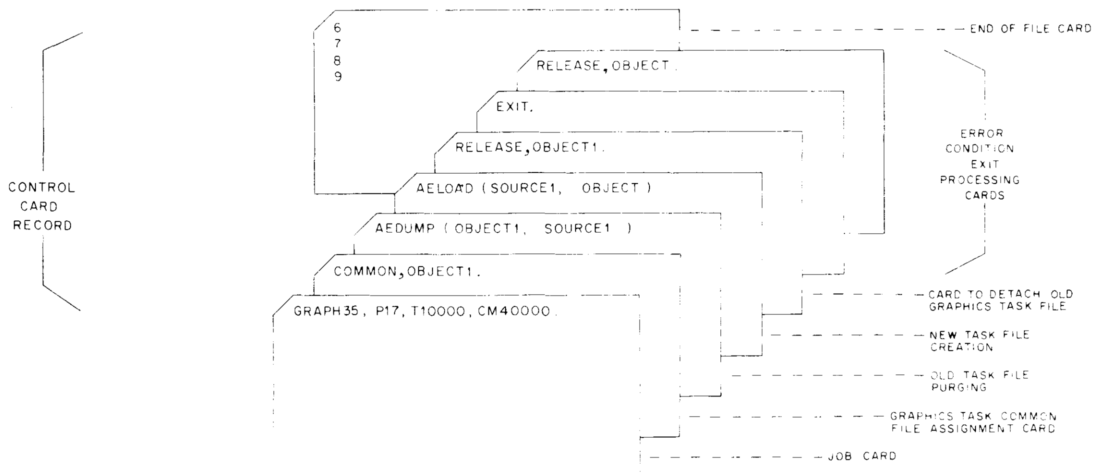


Figure 2-7. Sample Deck to Purge File Within System

PROGRAM EXECUTION RUN

Figure 2-8 shows a typical program execution run card deck; the program uses the graphics COMMON file called OBJECT, which was created by the decks in the preceding figures.

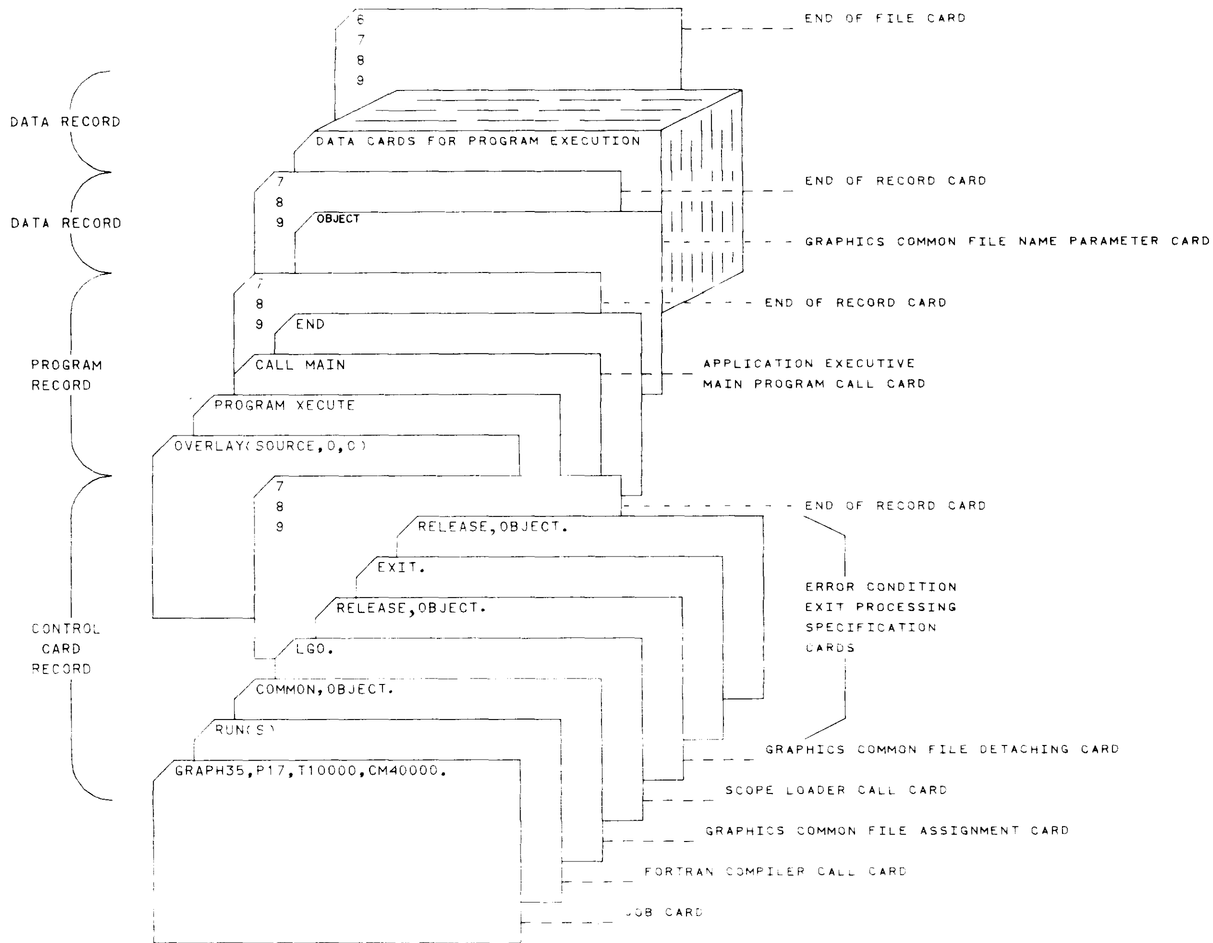


Figure 2-8. Execution Run Card Deck

SYSTEM UTILITY FUNCTIONS

In addition to the Scheduler and its subroutine, a graphics program uses SCOPE routines to create and maintain the graphics task file, and to process abort conditions.

TASK FILE CREATION

Initially, application programs can enter the system either through a remote card reader at the 1700, or at the 6000's card reader. Remote entry gives the programmer a convenient tool for program debugging.

After the program is submitted to the system, the control cards in its first logical record determine further processing.

First, SCOPE queues the job in the batch input queue, according to the priority on its job card, and creates the proper entries in the system File Environment Table/File Name Table (FET/FNT).

When SCOPE assigns the program to a batch job control point, the next control card is processed. This is the RUN card, which calls the FORTRAN compiler.

After compilation, the next control card is processed. For a graphics task file creation run, this would be the LGO card.

The LGO card calls SCOPE's General Purpose System Loader (GPSL), which takes the compiler's output, satisfies all 6000 Basic Graphics Package references from the system library, and organizes this data into a serial-access scratch file of overlays. This file is given the name specified on the main (or zero-level) OVERLAY card; it is written one overlay to a record and positioned after the program's first record (the main or zero-level overlay record).

Each record of this file contains two tables. The first is the 77 or prefix table; the second is the 50 or overlay table. The 50 table contains two header words with the format:

59	47	41	35	17 0
5000	Primary Overlay Level Number	Secondary Overlay Level Number	FWA of Overlay with Respect to Control Point RA	Address of Overlay Entry Point with Respect to Control Point RA
Overlay Entry Point Name				Program Address

followed by the binary text of the overlay.

SCOPE continues processing the LGO card by starting program execution; the program is initially treated as a batch job and executed at a batch job control point.

The first instructions executed are in the zero-level main overlay. These are supplied by either the programmer or the Application Executive MAIN program (see Section 7), and place file names in RA+2 and RA+3 of the program's current control point area.

The program then passes control back to SCOPE for normal termination of LGO processing. This consists of executing the next card in the control card record – which should be an AEFILE card in a file creation run deck. This card calls the system AEFILE routine.

AEFILE ROUTINE

AEFILE is the graphics task file creator; it reads the name of the loader-created overlay file from RA+2 and then writes that file (without the zero-level overlay record) on the system disk as absolute-addressed FORTRAN overlays.

This new file is the program's graphics COMMON file. It consists of named random records, each containing a 50 table and a primary overlay (the 77 table is not written into the graphics COMMON file). The record name is taken from the overlay entry point name in the second word of the 77 table.

AEFILE catalogs the disk address of each overlay record, and writes a task directory containing this information as the last logical record of the graphics COMMON file.

TASK DIRECTORY

The task directory (Figure 2-9) can contain pointers for MNOVL overlays (MNOVL is an installation parameter). The applications programmer can make additions to and deletions from an existing task file, since each task is accessible through its name in the task directory.

The task directory contains two blocks of information, the first of which is a standard index for a named random file.

The first block consists of one header word and two central memory words for each overlay record in the graphics COMMON file. The header word is negative, to indicate that the information block following it is a named random index.

The second block of information contains one entry (a single central memory word for each overlay record). This block is treated as a suffix to the index in the first block, and is used by the Application Executive routines (see Section 7) to load the task overlay during program execution.

Only the first block of the task directory is used to read or write the graphics COMMON file, but both blocks are included in the index pointers when the file is closed or opened, so that they will be retained on the disk as a catalog.

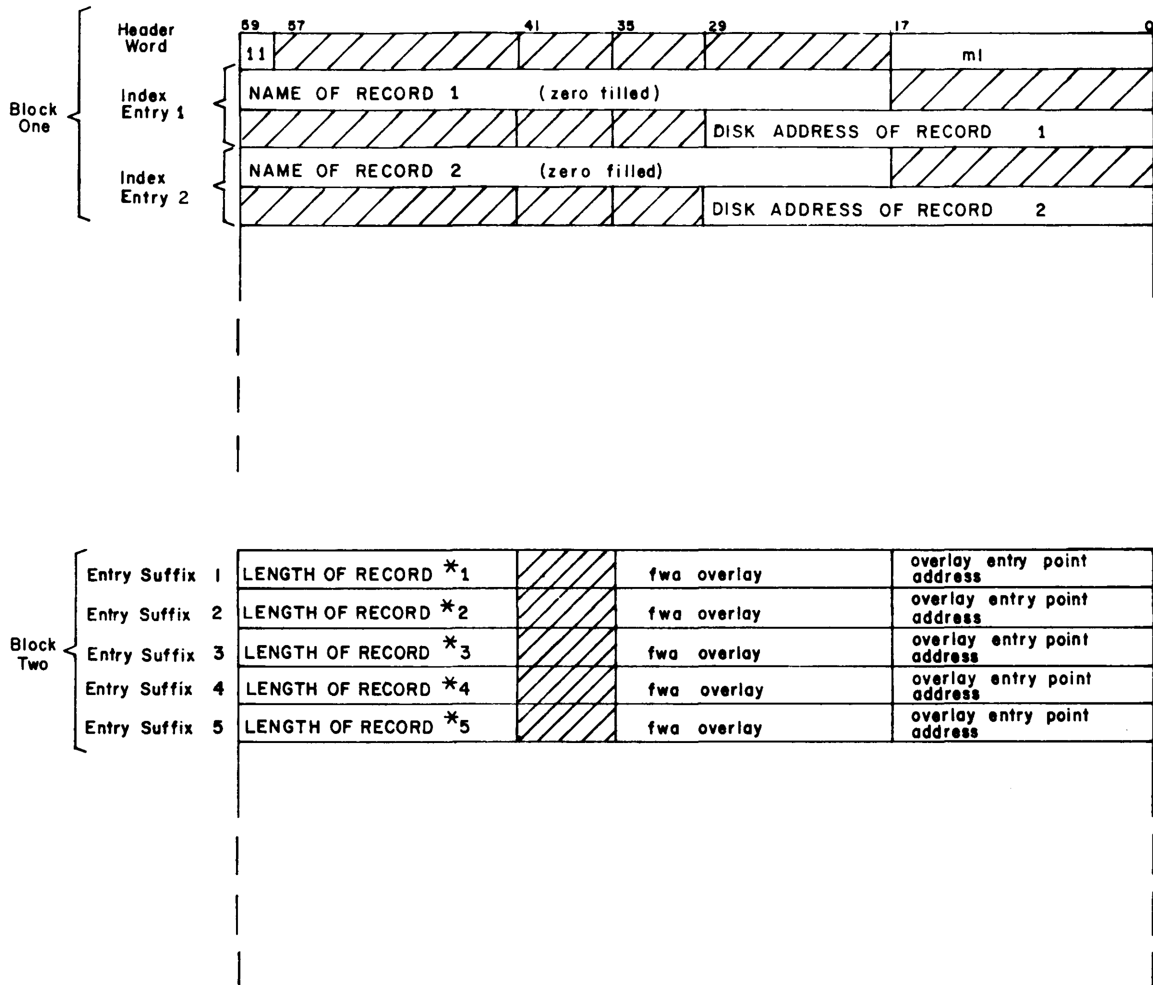


Figure 2-9. Task Directory

AEFILE ACTIONS

Before AEFILE can create a graphics COMMON file, it must make FET/FNT entries for both the COMMON file and the loader-created overlay source file; AEFILE uses SCOPE library macros and the contents of RA+2 and RA+3 to do this. If AEFILE detects an error in the Table entries when the macros finish, it produces a dayfile message (see Appendix B) and aborts the job.

The graphics COMMON file entry defines the file as a system COMMON file and associates the programmer's graphics COMMON file name with it. The source file entry is used to save that file on the disk after the graphics COMMON file is written; the source file is treated as a local file and is destroyed when program execution ends.

After the entries are made, AEFILE uses SCOPE library macros to open the COMMON file, read the overlay source file, write the COMMON file, and close the overlay source file. These SCOPE macros write the graphics COMMON file on the most easily accessed allocable device (usually the system disk).

If AEFILE finds that FET/FNT entries already exist for the graphics COMMON file, it opens the file, saves the index, adds or inserts the contents of the overlay source file to the COMMON file, then writes a new task directory containing the latest index entries.

TASK FILE MAINTENANCE

If AEFILE is used to replace a task in an existing graphics COMMON file, it performs the action logically but not physically. This means that the old copy of the task still occupies storage space in the file, but is not listed in the new task directory index.

For example, the file OBJECT created by the decks in Figures 2-2 and 2-5 would contain:

TASK1
TASK2
•
•
•
TASK77
Old Index
New TASK1
New Index

A file like this should be purged after several debugging or updating runs, to keep it from wasting mass storage and becoming unwieldy. Purging is done with the AEDUMP and AELOAD routines at a regular batch processing control point.

AEDUMP ROUTINE

AEDUMP is a system library routine that is called by a control card; it requires 15K words of memory. AEDUMP reads the indexed random file named by the first parameter on its control card, and writes a new sequential file with the name specified by its second control card parameter.

The sequential file created by AEDUMP contains the index of the random file as its first record. Although the disk addresses in the index are meaningless, the record names and index suffix entries do not have to be altered to recreate a random file.

The other records of the sequential file are the binary text task overlay records; these records are written in the order that they are listed in the index. Only those records from the random file that are listed in the index are written into the new file. Unlisted records are skipped, so that the file created from the records in the example above would contain:

New Index
New TASK1
TASK2
•
•
•
TASK77

This sequential file could then be written on tape for storage outside of the system, or it could be used immediately to recreate a random task file – using the AELOAD routine.

AELOAD ROUTINE

AELOAD is also a system library routine, and is called by a control card. AELOAD reads the sequential-access file named in the first parameter of its control card and creates a random-access COMMON file with the name specified by the second control card parameter.

The sequential-access files used by AELOAD need not be located in mass storage; AELOAD will call a tape driver to read the file if the programmer has supplied a valid REQUEST control card in his job deck.

The file created by AELOAD is structured exactly the same as one produced by AEFILE. The new task directory contains new disk addresses; the name of each task record is checked against the sequential file index as the record is written in the new file (if the names do not agree, a diagnostic message is produced and the job is aborted).

The AELOAD graphics task COMMON file can be used for program execution by the card deck shown in Figure 2-8.

AELOAD requires about 15K words of memory.

GRAPHICS PROGRAM ABORTING

If an applications programmer wants to cause a program abort, he usually creates a light button at the graphices console to call GIABRT (see Section 7). This routine displays a dayfile and console message, and calls the SCOPE Abort routine.

When a 6000 Basic Graphics Package routine finds a programming error, it produces a dayfile and console message; the program's Application Executive routine then issues the messages and calls the SCOPE Abort routine.

If the 6000 Series computer detects an error condition during program execution, it sets a control point flag which calls the SCOPE Abort routine and produces a dayfile message.

If the 1700 Computer detects an error condition or an illegal request, it generates IMPORT directive code 23 (the 1700 operator can also generate this code with a type-in command – see Section 9). This sends a message to the affected console and informs EXPORT to flag the program for abortion. The Scheduler detects EXPORT's flag during the next rollin of the program, issues a dayfile message, and calls the SCOPE Abort routine.

EXPORT removes an aborted graphics program from the Scheduler's input queue and disconnects any graphics consoles assigned to it.

The SCOPE Abort routine releases all of the job's files to the system and sends output files to the 1700 if the program originated there. It will dump a core listing with the output file if the program requests it by using a control card.

After a graphics abort, the dedicated memory assigned to the program is not released to batch jobs as is the normal system procedure, but is retained for future graphics programs.

SCHEDULER

The Scheduler is a PPU program that is called into its peripheral processor by EXPORT whenever necessary to provide dynamic scheduling and time-sharing for graphics jobs running at graphic control points. Graphics jobs are not queued. If a job is on console #1, another job for console #1 cannot be read in until the current graphics job on that console is aborted and detached from the console.

Initially, a graphics job enters the Interactive Graphics System as a batch job, and is assigned to a batch-processing control point for execution (batch job scheduling is done by SCOPE, not by the Scheduler). At some point in its execution as a batch job, the graphics job calls the graphics reformatter (see Application Executive, Section 7).

GRAPHICS REFORMATTER

The graphics reformatter is a Scheduler subprogram; it puts a graphics job into the graphics rolled out format so that the job can be scheduled at a graphics control point.

After a program's initial call to the Scheduler/reformatter, the Scheduler drops the CPU and clears the program's EXPORT communication word at RA+76_8. The Scheduler then rolls out the program, its control point field, dayfile, and all of its associated File Name Table Entries The program is then assigned an initial priority and placed in a special graphics input queue.

When the program is rolled back in, the File Name Table entries are replaced to reflect the new control point number.

SCHEDULING OF GRAPHICS CONTROL POINTS

ROLLIN PRIORITY

The execution priority of each program in the Scheduler's graphics input queue is determined by the program's current field length and whether or not it has any unsatisfied graphics input requests; short programs with no unsatisfied requests have the highest priorities.

LONG PROGRAMS

At the beginning of graphics operation, the 6000 operator dedicates a fixed amount of core memory to each graphics control point. A graphics program that requires more memory than is available at the larger control point is then forced to wait in the graphics program input queue until the space it requires becomes available; such a program would wait only when there are batch jobs requesting or occupying all of the available core memory.

If a graphics program is larger than either dedicated graphics control point area,but has been rolled in, and batch jobs request more memory after it is rolled out (see time-slicing, below), then the program may be affected adversely. It will take on the lowest priority, because other graphics programs fit in the dedicated area and they will be scheduled ahead of the large graphics program. After all other rolled out and new graphics programs are executed, a storage move request is made to allow the longer program back into the computer. Therefore, the application programmer should write programs shorter than or equal to the designated length of the dedicated area, or else suffer the consequences of longer response time.

SCHEDULER ROLLOUT STRATEGY AND TIME-SLICING

Because the amount of CPU and PPU time required by a program varies widely from task to task and application to application, Interactive Graphics uses a form of time-slicing that allows several programs to time-share a control point without destroying the console user's real-time environment.

The Scheduler determines how long each program will remain at a graphics control point. This length of time, called the frame time, can never be less than a guaranteed minimum value chosen by the installation when the Scheduler is assembled.

The frame time is computed by an algorithm, using the rollin/rollout time as the variable that must be equalized between programs. The algorithm includes the following variables:

R_{final}	Calculated resident time (frame time)
I	Minimum resident time (installation parameter)
N	Current program field length (multiples of 100_8 central memory words last rolled out)
M	Maximum allowable percentage of frame time to be used for rollin/rollout

The average hardware overhead time for reading or writing data in mass storage is 60 milliseconds, and each transfer of 100_8 central memory words requires .4 milliseconds; therefore, the Scheduler calculates that the last rollout of the program required 60+.4N milliseconds.

Since the next rollin of the program will require the same amount of time, the total rollin/rollout overhead is 120+.8N milliseconds. Because the total overhead should not use up an appreciable amount of the frame time, the Scheduler chooses R_{final} so it has one of two values. If:

$$120+.8N \geq MI, \text{ then } R_{final} = \frac{120+.8N}{M} \;;$$

however, if:

$$120+.8N < MI, \text{ then } R_{final} = I.$$

This formula allows large programs to remain in core longer than small programs, since small ones are faster to roll in and out. If the difference in program lengths is not very great, a word in the BATCHIO control point area can be altered so R_{final} is not changed after every rollout but has a pre-set value $\geq$ I. The value of I assembled into the Scheduler can also be varied through this word (the larger the value of I chosen, the more uniform each time-slice is).

NORMAL SCHEDULING

The frame time is calculated at the graphics control point; the Scheduler determines the new R_{final} for each program after every rollout. When the program is rolled back in, the Scheduler writes this R_{final} in the job's graphics control point area. EXPORT then monitors the length of time the program resides at the control point (see Figure 2-10).

If the program's actual resident time exceeds its frame time (or if the program terminates, asks for graphics input, or aborts), EXPORT calls the Scheduler into an idle PPU; the Scheduler then rolls out the program, calculates a new R_{final} if processing is to continue, and rolls in another program.

Rollout/rollin is not performed if there is only one graphics program in the system for each graphics control point.

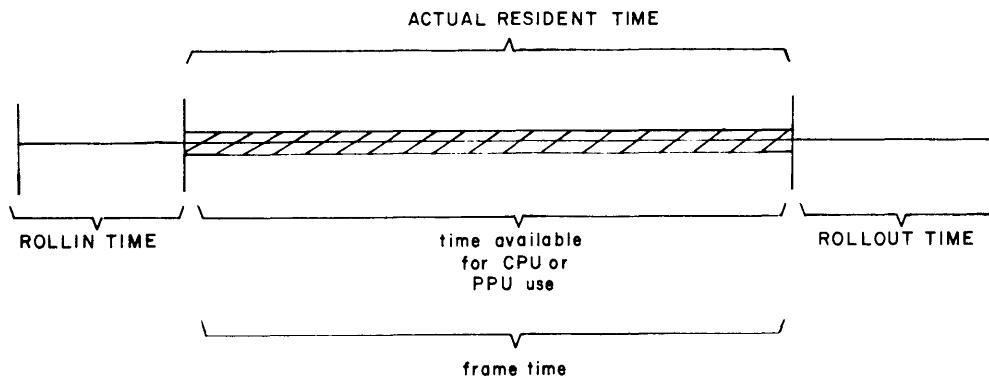


Figure 2-10. Typical Time-Slice

ROUTINE COMMUNICATION AND HOUSEKEEPING

The Scheduler rolls out a program by asking SCOPE Monitor to place the CPU in recall. Monitor is then free to turn the CPU over to another graphics control point, or if there is no other active graphics control point, to turn the CPU over to batch jobs.

While a graphics job is running, its control point asks SCOPE for PPU and CPU service in the normal manner; graphics control points have the highest system priority for such service (except when the Storage move or RESPOND programs are present).

The rollin/rollout program of the Scheduler keeps track of graphics priorities, and of where rolled out programs are to be found in mass storage. When the program is to be rolled in from mass storage, the Scheduler will adjust the control point field length if a storage change occurred during the time the program was rolled out. All the rules of 6000 Series program protection keep each program independent.

GRAPHICS CONTROL POINTS

INITIALIZATION

The 6000 operator assigns one or two graphics control points manually, using the procedure given in Section 9. The type-ins that he uses enter the control point numbers in a table at the BATCHIO control point area (BATCHIO must be assigned first); this table identifies which control points EXPORT must service for graphics processing.

STRUCTURE

Figure 2-11 shows the general structure of one graphics control point area. The uses of the various words and subdivisions are described in other sections of this manual. Minimum field length of a control point area is about 7000 octal words.

RA	SYSTEM AND GRAPHICS COMMUNICATIONS AREAS
	LABELLED COMMON (IF PRESENT)
	ONE OR MORE STANDARD ROUTINE FILE BUFFERS
	SYSTEM (FORTRAN ERROR TRACING ROUTINE)
	MAIN (APPLICATION EXECUTIVE ROUTINES), ASSUMING NOTHING ELSE IN 0,0 OVERLAY
	BLANK COMMON AREA (IF PRESENT)
	SIOS (FORTRAN INPUT/OUTPUT ROUTINE)
	TASK OVERLAY
	IN-CORE DATA BASE (DATA HANDLER AREA)
RA + FL	

Figure 2-11. Graphics Control Point Field

NUMBER

In order to best use the dedicated space available plus the idle time when graphic tasks are being rolled in or out to mass storage, two graphics control points should be used. While one control point is accessing the mass storage device, the other can be executing and/or performing input and output to the graphics console.

SIZE

The total dedicated graphics area should be small (on the order of 10K - 20K) because of the suggested method of application programming. Each installation determines how much core storage will be dedicated to graphics. Thus, it is possible to assign a minimum amount of memory to a graphics control point and make each graphics program request storage when needed. However, this may also slow down the response time at the graphics console.

FILES

All files used by a graphics program must be attached to it by the programmer, using standard SCOPE control cards. A maximum of eight files per program can be handled by the Scheduler. This number includes all local scratch files, the job's graphics COMMON file, the overlay source file named in the zero-level overlay card parameter field, and all Data Handler files (see Section 7).

Once a file is attached to a graphics program, the file is not available to other programs.

GRAPHICS COMMON FILE

The file created by the AEFILE and AELOAD routines is a graphics COMMON file.

Until there are permanent files in the graphics SCOPE system, COMMON file names will have to be unique for each user. Using the last two digits of the file name to designate a user graphics console would eliminate possible duplications.

LOCAL FILES

All files that are local are rolled out with a program, so that either graphics control point can be used (if available).

INPUT FILES

Tape and card files other than the FORTRAN input file must be put in mass storage before being used by a graphics program. These files are read in and made COMMON with names different from that of the graphics task COMMON file.

OUTPUT FILES

All tape output is through a disk file. After a graphics job completes, a SCOPE utility program can be used to transfer the data to magnetic tape.

EXPORT/IMPORT AND DATA COMMUNICATION 3

INTRODUCTION

The EXPORT/IMPORT package is the set of routines on which all intercomputer communications of the Interactive Graphics System are based. Besides being a functional communications package, EXPORT/IMPORT has the advantage of allowing both graphics and non-graphics remote jobs to enter the 6000 Series computer job queue.

The communications scheme of EXPORT/IMPORT is excellent for graphics requirements. The EXPORT program optimizes the use of the intercomputer communications line by using an asynchronous method of communicating with the 1700, and by allowing variable length data transfers from IMPORT. In addition, EXPORT informs the 1700 Computer of the optimum time to send specific classes of data buffers. This tends to synchronize the lengths of transfers.

The EXPORT/IMPORT package also contains an error recovery routine which handles such errors as transmission noise bursts and sequence errors (produced when one of the computers fails to receive a complete buffer). All errors that are detected by the hardware are recovered by EXPORT/IMPORT; the error detection ability is very close to 100 percent.

Special features of SCOPE aid these two programs in the processing of remote jobs.

EXPORT

EXPORT (Executive Processor of Remote Tasks) resides in one 6000 Peripheral Processor Unit (PPU), assigned to remote communication by central computer.

EXPORT consists of a resident program with several overlays. The resident program handles communications and the processing of data; the overlays perform housekeeping. In addition, one complementing SCOPE system overlay, 2TJ, which performs job card translation, is called as needed to assist in the processing preparation.

INITIALIZATION

When BATCHIO is assigned to a control point, its initialization routine requests enough central memory storage for the EXPORT counter area. Then, whenever a 1700 communication line becomes active, BATCHIO automatically loads the EXPORT resident program into an idle PPU.

PROCESSING CONTROL

Processing accomplished by EXPORT is controlled by the EXPORT PPU resident program. The resident program remains in PPU memory until all communication lines become inactive. The main program loop cycles between several activities in the resident routine and the functions performed by the specific central memory overlays. These activities include:

- Handling all communications with IMPORT
- Processing all directives requesting output data
- Processing card data directives
- Processing special request directives
- Scanning for new output files
- Making all CIO requests or stack entries
- Handling file manipulation requests
- Monitoring actual resident time for a program at a graphics control point
- Calling the Scheduler
- Processing input and output requests for graphics data

EXPORT employs the Circular Input/Output package (CIO), the Stack Processors and the Close Files routine (CLO) to aid in performing its tasks. Using data received from IMPORT, EXPORT prepares input files for processing under SCOPE and then intercepts output files for return transmission to the remote terminal. Direct central operator communication with the remote site is accomplished through the 6612 System Display Console of the 6000 Series computer.

COMMUNICATION CONTROL

Although IMPORT initiates communication with the 6000, EXPORT controls all data communication operations. The 1700 terminal has a buffered data channel dedicated to the hardware interface so that information from the 6000 can be received at any time, even when other input/output or IMPORT processing operations are in progress. EXPORT, however, accepts data only at specific time intervals because the 6000 does not have a hardware buffer large enough for an entire transfer.

EXPORT requests data for the central computer by entering an output/input routine, which transmits a status word to all active terminals. While in this routine, EXPORT expects to receive a directive word from each active terminal. Total time required for one output/input operation is dependent on the terminal which has the longest transfer length for the given operation (see Figure 3-1).

EXPORT sequentially services up to four IMPORT terminals. If one of the active terminals fails to transmit, EXPORT attempts automatic recovery of communication; if immediate action fails to maintain communication, EXPORT considers the terminal inactive and requires it to re-initiate communications.

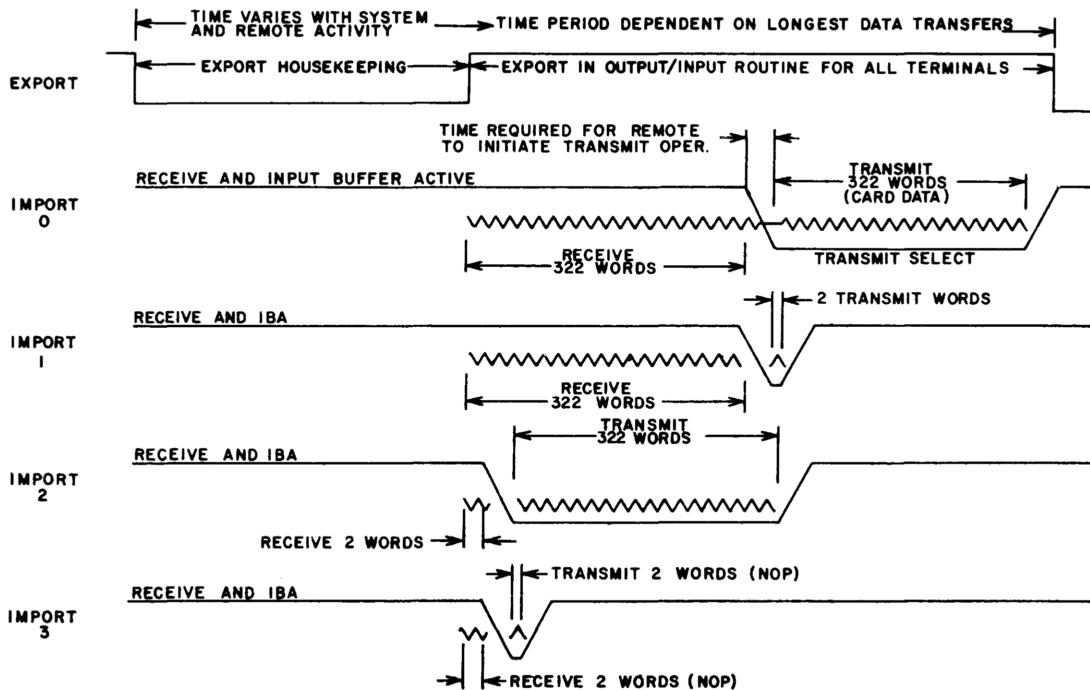


Figure 3-1. Conditions Present During One EXPORT Service Cycle

EXPORT SERVICING CYCLE

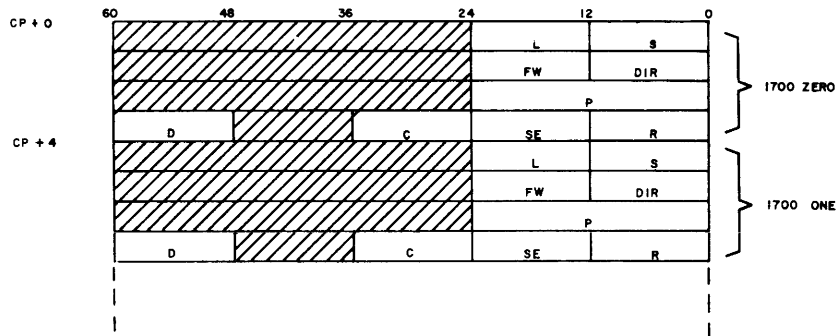
Although all 1700 terminals possess the same line speed, not all transfers require the same amount of time because of differences in transfer lengths. The EXPORT hardware and software compensate for these variations by transmitting to one terminal while receiving from another during each terminal servicing cycle.

The user can expect the best performance of EXPORT when all transfers are of the same length.

Figure 3-1 illustrates one set of possible terminal conditions which may exist during one EXPORT servicing cycle.

EXPORT COUNTERS

Space has been allocated in central memory to maintain information needed for execution of EXPORT/IMPORT. Beginning at the BATCHIO control point, RA, four central memory words are reserved for each connected terminal; these contain the first two words of the current transfer to the 1700, the first two of the current transfer from the 1700, and the number of sync errors for this terminal. The counters are cleared only when EXPORT is dropped from the BATCHIO control point. Figure 3-2 is an illustration of this area.



CP — CONTROL POINT ADDRESS
L — TRANSFER LENGTH FROM CURRENT EXPORT STATUS TRANSFER
S — STATUS WORD FROM CURRENT EXPORT STATUS TRANSFER
FW — TRANSFER LENGTH FROM CURRENT IMPORT DIRECTIVE TRANSFER
DIR — DIRECTIVE FROM CURRENT DIRECTIVE TRANSFER
P — 24-BIT TERMINAL TOTAL I/O PASSES COUNTER
D — ACTUAL STATUS FROM DSC IF ERROR DETECTED IN STATUS WORD
C — 12-BIT TOTAL CYCLIC ERROR COUNTER
SE — 12-BIT TOTAL SYNC ERROR COUNTER
R — 12-BIT TOTAL RETRANSMISSION COUNTER

Figure 3-2. EXPORT Counters

DATA TRANSFER

When EXPORT and IMPORT are not actively communicating, an idle bit pattern continually flows between the data set controller at the central site and the data set controller at the remote site. Accompanying this data transfer is a sync word issued by the transmitting controller.

The receiving controller automatically acknowledges the sync word while the idle bit pattern is being received; any other type of data transfer would occur after sync word acknowledgement.

At the end of each data transmission, the controller generates and transmits a 12-bit cyclic code word - which is appended to the transferred data. When the complete data block is received, the receiving controller checks the cyclic code word against one it has generated. If they do not agree, the receiving controller sets the cyclic code-error status bit.

RULES

Transfers of data between the 6000 Series computer and the 1700 terminal include an exchange of control information in a fixed format. EXPORT sends information to IMPORT in a 12-bit status word specifying the types of data that EXPORT is prepared to handle. A 12-bit directive is returned to EXPORT, selecting one or more of the options offered to IMPORT in the status word, or making a special request. Appropriate data may accompany the status word or the directive word.

Non-graphics data included with a status word transfer must always correspond to the latest directive received; the data in a directive transfer will always be of the type indicated in the accompanying directive word.

EXPORT TRANSFER FORMATS

All data, except message and idle transfers, is sent in sector blocks.

EXPORT sends graphics data in the format shown in Figure 3-3. If there is no graphics data available, EXPORT sends only the transfer length and status header words to IMPORT. If data is sent to the 1700, it is transferred in a block with a fixed length of 322 12-bit words; i.e., 320 words of data (the EXPORT output buffer) and the two communication header words.

The communication header words in a graphics data transfer are similar to those of a non-graphics transfer. The transfer length word tells IMPORT how many words of the fixed block contain valid information; this length does not include the hardware-generated cyclic code word.

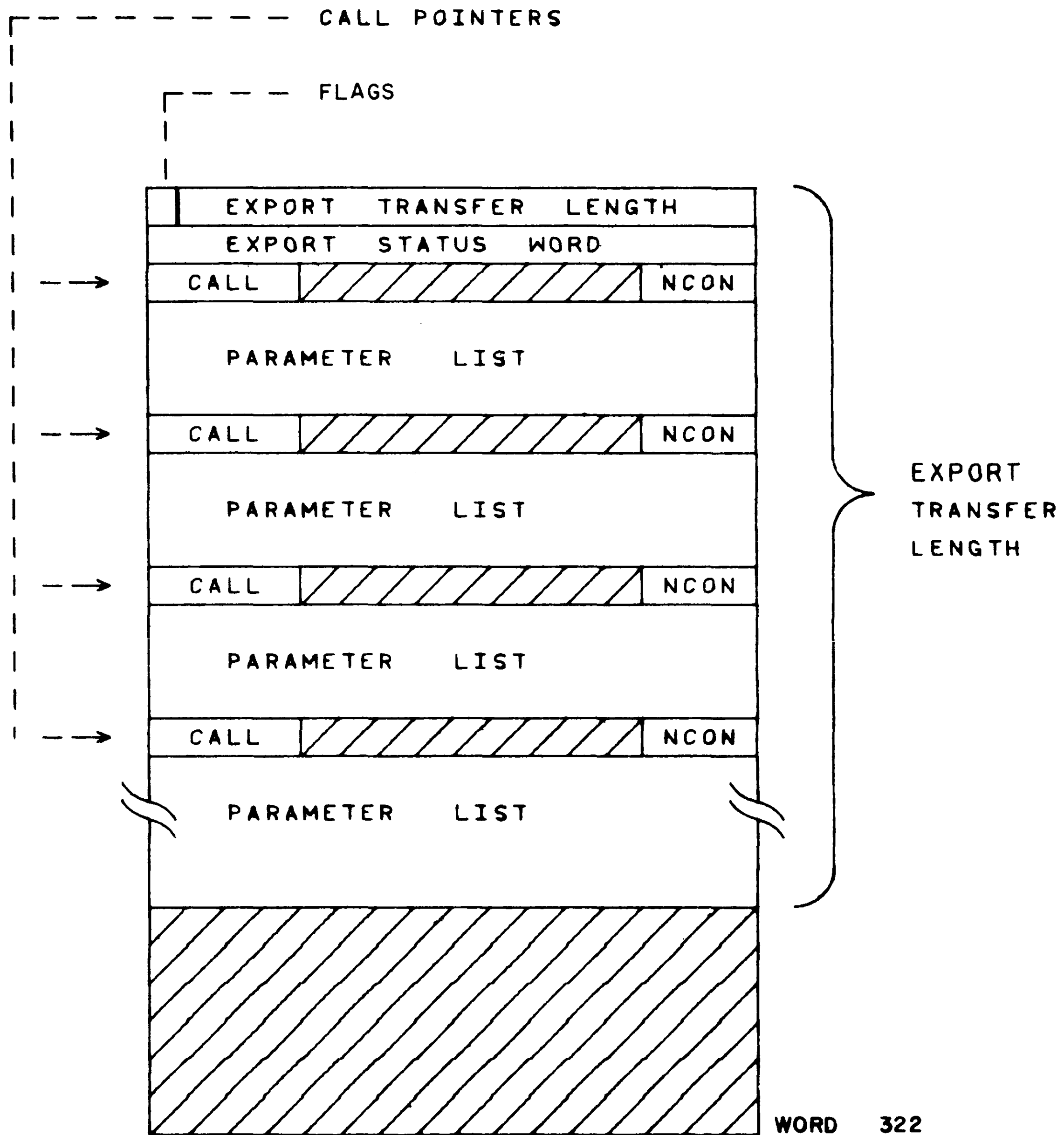


Figure 3-3. EXPORT Graphics Transfer Buffer

Table 3-1 illustrates the meanings for the status word in both graphics and non-graphics transfers.

TABLE 3-1. STATUS WORD CODES

Bit	Assignment*
11	Card reader buffer empty
10	Output data stream 1 is available**
9	Output data stream 2 is available**
8	Typewriter message available
7	Central display message buffer empty
6	JOB card error
5	Last output buffer
4, 3	Used to indicate special conditions of output data
2	Output data stream 3 is available**
1	Output data stream 4 is available**

*Not all of the status bits are used in every system, but they are designated and reserved in order to retain systems compatibility.

**Graphics data is assigned to one of the data streams by the installation.

GRAPHICS DATA

Application program requests for output from a graphics console and output to a console are queued at the BATCHIO control point (see Section 2). If such requests arrive before EXPORT transfers previous requests to the 1700, they are stacked until the maximum length of the EXPORT graphics buffer is reached. The 6000 Basic Graphics Package routines perform the program buffer setup for the actual input and output to the EXPORT graphics buffer area (see Section 7).

EXPORT continually asks the 1700 for graphics input data, but the 1700 will not release any of this data until the application program specifically requests it. Thus, data can never originate at the graphics console and reach the 6000 Series computer without the knowledge of the application program.

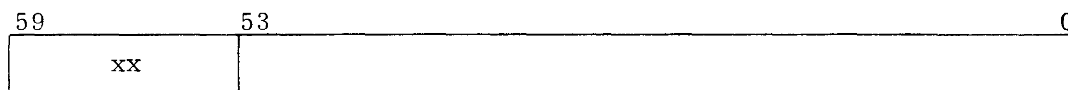
EXPORT periodically scans RA+76_8 of each graphics control point for a data transfer request. There are four such calls (see following page), each formatted by the 6000 Basic Graphics Package routines and placed in the application program's EXPORT communication word at RA+76_8.

EXPORT checks the console number specified in each of these four requests against the numbers of those consoles which are assigned to the program making the request. If the console designated is illegal, the job is aborted with an appropriate message in the Dayfile.

When EXPORT finds a request call with a valid console number at RA+76_8, it performs the function and clears RA+76_8 to notify the application program that it has completed the request.

REQUEST TO ATTACH CONSOLE

The format for this request is:



xx Console number NCON in octal (see GICNJB, Section 7)

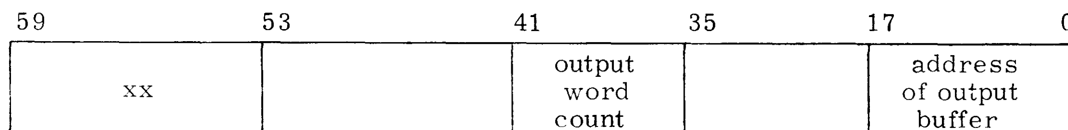
REQUEST TO DETACH CONSOLE

The format for this request is:

xx Graphics console number NCON in octal (see GICNRL, Section 7)

REQUEST TO OUTPUT DATA

The format for this request is:



xx Graphics console number NCON in octal

The application program uses this call to request that data from its IBUF buffer be sent to the graphics console. The application program will go into recall when this call is issued; EXPORT recalls the CPU after the data transfer is completed.

The EXPORT program transfers the data block (starting at the designated address and continuing for the number of words indicated by the word count) to the EXPORT graphics output buffer, and to the 1700 when it requests graphics data. The console number in the format tells EXPORT which 1700 should receive the data.

REQUEST FOR INPUT

The format for this request is:

59 xx	53 Input Word Count	47	41 Output Word Count	35 Input Address	17 Output Request Buffer Address 0

xx Graphics console number NCON in octal

This call to EXPORT generates a request to the 1700 for those light buttons and legal picks that the program is concerned with. When a request for input is sensed by EXPORT, the output request buffer is put in the EXPORT output area and the application program is put into a recall state, subject to being rolled out if another graphics program is ready to be rolled in.

When the input from the graphics console is received at the 6000 Series computer, the application program is reactivated, its input buffer is filled, and it is notified that it has the requested input. If the program is rolled out, the input request will still be associated with the program; EXPORT does not have to store the input buffer addresses, since they are at RA+76_8.

BUFFER PASSING

The data stream is the flow of a file or buffer of data from one computer to another. Input and output, as defined below, are used separately to refer to all data transmissions between the remote terminal and the central site.

The two types of data streams are:

- Output data stream - the data flow from the central computer to the remote computer.
- Input data stream - the data flow from the remote computer to the central computer.

Each data stream is reserved by the installation for the use of a specific input or output device.

CHARACTER SET

The EXPORT program processes all teletypewriter, card (except binary), and line printer data in 6-bit display code using the standard SCOPE character set (Appendix C). Two display-coded characters are packed into one 12-bit PPU word. Before transmission, IMPORT converts card reader and teletypewriter input to display codes. IMPORT formats input data according to SCOPE system requirements. Printer, punch, and teletypewriter data are converted from display code to ASCII code for output to the peripheral equipment. Output data is

received in the same manner as it is generated within the SCOPE system prior to output. Binary information is accepted in either of the following formats and is transmitted to EXPORT as column binary card images:

- 80 columns of free-form binary
- 6000 Normal mode binary

FILE PROCESSING

At the 6000 Series computer, files associated with jobs entered remotely are identified by the remote bit (the highest-order bit) of the 12-bit disposition code and the fifth character of the file name in the FNT/FST. The remote bit in the second byte of the third word of the FNT/FST is normally not interpreted until the job has been completely processed. EXPORT is responsible for disposing of all output files when the remote bit is set. Consequently, SCOPE ignores these files. Remaining bits of the disposition code field identify the type of output file. The fifth character of the file name contains the terminal identification. In all other considerations, the job is processed the same as a nonremote job.

TERMINATION

Each terminal notifies EXPORT when communication is finished. When all lines become inactive, EXPORT enters PP RECALL.

JOB FLOW

The steps listed below illustrate the flow of data and control from the beginning to the end of a job in one terminal system.

INITIALIZATION

Remote operation:

- Loads IMPORT program
- Sets up to read cards

INPUT FROM CARDS

- IMPORT reads cards
- IMPORT converts Hollerith card data to display code and inputs 6000 formatted or free-form binary card data
- IMPORT packs data into buffers equal in size to one disk sector (64 central memory words)
- IMPORT transmits to EXPORT full data buffers with EOR and record level, or data buffers with EOF

- EXPORT inputs data and writes it to central memory buffers
- EXPORT requests SCOPE to write data from buffers to disk
- At EOF, the file is set to type INPUT and released to SCOPE for processing

INPUT FROM GRAPHICS CONSOLE

- IMPORT detects a 1700 Basic Graphics Package processing flag, and calls the Buffer Translator
- The Buffer Translator repacks the queued information
- IMPORT reads the packed information into its data buffer and transmits the buffer
- EXPORT writes the data into the program's central memory buffers

OUTPUT TO GRAPHICS CONSOLE

- EXPORT scans $RA+76_8$ of the graphics control point and interprets any requests found there
- EXPORT packs program buffers into its output data buffer
- EXPORT transmits the buffer to IMPORT
- IMPORT calls the Buffer Translator
- The Buffer Translator repacks the data
- The 1700 Basic Graphics Package acts on the data

OUTPUT TO PRINTER AND PUNCH

- EXPORT scans the File Name Table/File Status Table for remote files of type OUTPUT. Output files are returned on a highest-priority and lowest SCOPE sequence number basis. Sequence numbers are used only when priorities are identical. (The standard SCOPE 3.1 header pages are provided on output.)
- EXPORT disposes of punch card data in the manner indicated in the Equipment Status Table entry for EXPORT. Disposition of data is handled in one of three ways:

  a. Data may be punched at the remote site when equipment is available.

  b. Data may be punched at the central site in the event the remote site does not have a card punch.

  c. Punch data may be dropped completely.

- EXPORT transmits output to IMPORT in one-sector blocks.
- IMPORT converts output data to the proper code.

- IMPORT deblocks print lines.
- IMPORT prints output in lines with carriage controls.
- IMPORT recognizes the Print mode control character (see Section 9).

ERROR DETECTION SCHEME

If, during transmission, data received does not correspond bit-for-bit with the data sent, an error has occurred. Following are descriptions of several error types and conditions along with methods of error detection. (Only errors in the communication facility are considered here.)

- Message Lost Completely – The leading sync words may be lost or mutilated so that the entire message is passed over as noise on the line.
- Message Garbled – Lightning, electrical disturbances, and random noise may introduce errors on the communications line or between a modem and the Data Set Controller (DSC).
- Bit Lost – Inaccurate timing in the modem may cause the entire message to be shifted forward by one bit.
- Microwave Transmitter Switching – All or part of a message may be lost while microwave transmitters are switched.

DETECTION PROCEDURE AND CAPABILITIES

The 6673 or 6674 DSC and the 1747 DSC hardware provide the primary error detection capability in the form of a 12-bit cyclic code encoder. The transmitting data set controller continuously and automatically generates a 12-bit cyclic code word that is appended to the transmission. Similarly, the receiving DSC generates a 12-bit word which is compared to the last word received. The transmission is assumed to be correct as received if the two cyclic code words are identical.

In some situations, detection of a transmission error occurs apart from the cyclic code check. This happens when analysis of the control data received shows an impossible condition, as an invalid transfer length. In these cases, the transmission is treated as if a cyclic code error has been detected.

EXPORT/IMPORT has the following capabilities in cyclic code error detection:

- Any odd number of errors
- All error bursts 12 bits or less in length
- 99.95 percent of all error bursts 13 bits long
- 99.98 percent of all error bursts longer than 13 bits

An error burst is defined as any pattern of errors whose length is the number of bits between the first and last errors of the transmission. (See Appendix G.)

ERROR COMPENSATION

An error recovery scheme provides a means of recovering from a sync word error. In the event both computers are in the Transmit mode, the remote computer is placed in the Receive mode when it has attempted to transmit 120 times without receiving a sync word acknowledgement.

If for any reason the communication line goes down between transfers, the DSC's will detect and inform the affected computers of the malfunction. If communications go down, EXPORT and IMPORT also output messages to their respective operators notifying them of malfunctions.

If either computer goes down before or during an input or output operation, the other computer detects this condition and informs its operator of the malfunction. For example, when a computer goes down while transmitting, the receiving computer inputs idle characters for the remainder of the transfer. Then, at the conclusion of the input, the cyclic code indicates that a malfunction has occurred. Further attempts to retransmit will not fault the sending computer, but the receiving computer continues to detect the down or off-line condition of the other computer.

IMPORT

IMPORT (Input/Output Monitor for Processing of Remote Tasks) acts as a system monitor program for the 1700 Computer. It interprets all input/output requests made to the 1700 and calls those parts of the 1700 Basic Graphics Package needed to service the application programs running in the 6000 Series computer.

IMPORT consists of a series of processing routines, subroutines, and related operating system routines. The major processing routines are:

- Main status loop
- Communications active check
- Line printer data conversion
- Card reader data conversion
- Teletypewriter data conversion
- Card punch data conversion
- Interrupt processing
- Determine directive code

ROUTINES

MAIN STATUS LOOP

The main status loop of IMPORT enters each of the major routines in sequence, executing those routines from which particular functions are currently required. The Interrupt Processing routine is entered automatically whenever the 1700 computer senses a controller interrupt.

The IMPORT program will recognize graphics data from both the 1700 and the 6000 Series computer. The main status loop of the program checks to see if data is available for the typewriter, card punch, card reader, printer, and graphics buffers. There are double-buffered output buffers for Interactive Graphics use; these are loaded by drivers contained in IMPORT.

COMMUNICATIONS ACTIVE CHECK

IMPORT designates to the Communications Active Check routine the following duties:

- Checking for remote site shut down
- Determining if data is still being received
- Checking expiration of allowed input time

LINE PRINTER DATA CONVERSION

The Line Printer routine is responsible for:

- Converting data from display code to ASCII and placing it in the printer buffer

CARD READER DATA CONVERSION

The activities of the Card Reader routine include:

- Acquiring card reader status
- Processing the job card
- Processing Hollerith, binary, EOR, and EOF cards
- Converting card image to display code

TELETYPEWRITER DATA CONVERSION

The teletypewriter routine responsibilities include:

- Controlling input and output with the remote operator
- Checking for manual interrupt by the operator
- Checking for error stack entries
- Checking for message(s) from the central computer
- Initiating appropriate display messages to the 6000

CARD PUNCH DATA CONVERSION

This routine is responsible for:

- Converting data from display code to ASCII and placing it in the punch buffer

INTERRUPT PROCESSING

When the 1700 Computer recognizes a DSC interrupt, an Interrupt Processing routine within the system is entered. After this routine saves the registers, control is transferred to the proper routine within IMPORT for processing. Interrupts are disabled for a brief period while the DSC is being serviced.

When a graphics console entry produces a 1705 Controller interrupt, control of the 1700 is turned over to the 1700 Basic Graphics Package Digigraphics Interrupt Processor.

Input from the communications line is initiated immediately after the output is complete by placing the DSC into Receive mode. When the interrupt processing routine is entered, a cyclic code error check is made, and the output routine is initiated (if necessary) to retransmit the data.

Output to EXPORT is similar for both a normal request and a retransmission. After acquiring DSC status, the Transmit mode is selected, and the output is initiated. If no retransmission is required, the following routine is entered prior to the next transmit operation.

DETERMINE DIRECTIVE CODE

The Directive Code routine's activities consist of:

- Examining last directive to determine if data was requested or sent
- Setting pertinent flags to indicate a course of action to the IMPORT data conversion routine
- Determining next directive

GRAPHICS DATA TRANSFERS

Unlike the normal mechanics of an EXPORT/IMPORT data transfer, there is no status bit for each graphics console. Instead, there is a status bit in one output stream for all graphics consoles, so that it is possible to turn off output from the consoles as a group, but not individually.

Unlike the data transfers from EXPORT, data transfers from IMPORT can be of variable length; the smallest transfer is two words.

The amount of data, the data itself, and information about the disposition of the data are passed from IMPORT to EXPORT using the format shown in Figure 3-4.

- Transfer Length (one 12-bit word)
- Directive Word Code (one 12-bit word)
- Up to 320 12-bit words of data

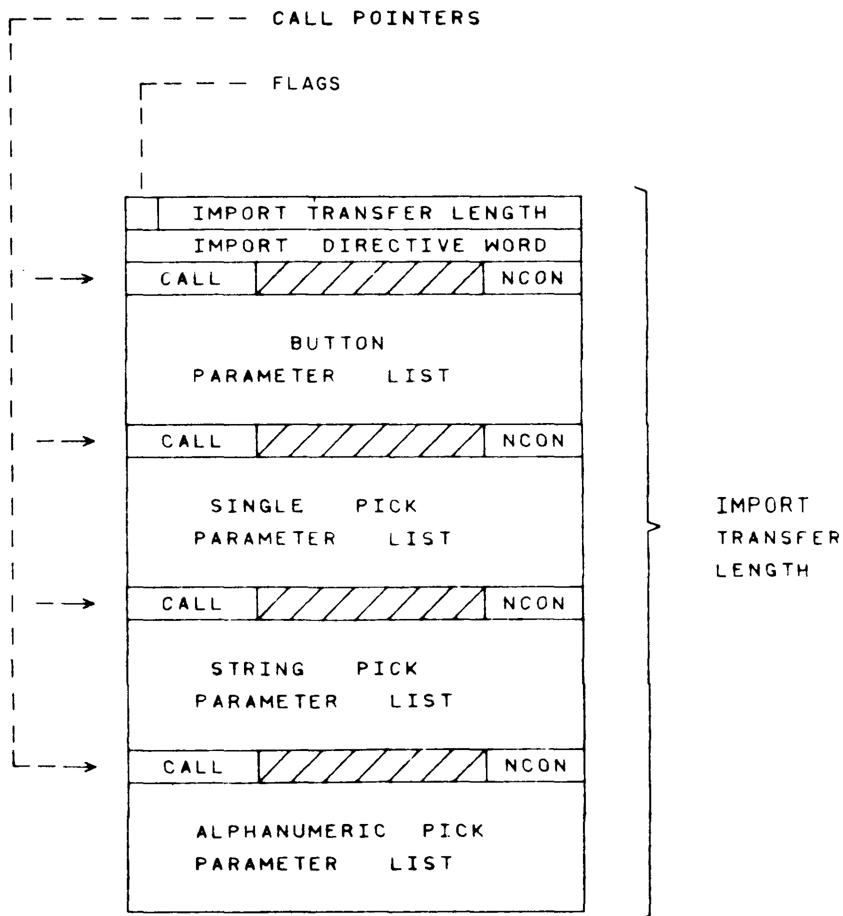


Figure 3-4. Sample IMPORT Graphics Transfer Buffer

IMPORT DIRECTIVE CODES

A summary of the directive word codes, with an abbreviated description indicating the meaning of each, is given in Table 3-2. Not all of the directive codes are used by every system, but they are designated and reserved in order to retain systems compatibility.

Directive codes in which the lower five bits equal 02, 07, 10, or 42 may take the form of xx02, xx07, etc. to form a double directive indicating that the remote terminal is sending input data and expects output data returned by the next transmission from the 6000 Series computer. The xx is the directive word code that is to apply to the returned data.

TABLE 3-2. DIRECTIVE WORD CODES

Directive Code	Accompanying Data	Description of Instruction
0001	None	No operation
0002	Packed card data	Load card buffer
0102	Card data (optional)	Load card buffer with End-of-Record (EOR)
0142	EOR level; Card data (optional)	Load card buffer with EOR and EOR level
0202	Card data (optional)	Load card buffer with End-of-File (EOF)
0003	Job Name	Request for job status
0004	None	Send output data stream 1
0005	None	Send output data stream 2
0404	None	Send output data stream 3
0405	None	Send output data stream 4
0006	None	Send message(s)
0007	Message	Load remote to central message
0010	Graphics Data	Load graphics data
0011	Job Name Time	Change time limit of named job to the time supplied
0012	Job Name Priority	Change the priority of the named job to the priority supplied
0013	n, x	If n=0, rewind current file of output stream x. If n≠0, backspace n sectors on output stream x. If x=2 or 3, the output stream is 2 or 3; any other value of x implies output stream 1.
0014	None	Repunch the current job
0015	Job Name	Abort the job if it is at control point
0016	dt, x	If dt=LP, terminate output 1 If dt=CP, terminate output 2 If dt=0, terminate output x

TABLE 3-2. (Cont'd)

Directive Code	Accompanying Data	Description of Instruction
0017	None	Shut down remote, rewind any files being output
0020	Disposition codes (8 words)	Word 1 = disposition of output data stream 1 Word 2 = disposition of output data stream 2 Word 3 = disposition of output data stream 3 Word 4 = disposition of output data stream 4 If word 8 = 10_8, divert all files to central site If dt=LP, divert line printer file to central computer If dt=CP, divert card punch file to central computer If dt=5555_8, divert all output to central computer (dt=LP or CP are valid only if the named job is in the output stack.)
0022	Reserved	
0023	Console number	Abort the graphics job whether it is in mass storage or at a control point

GRAPHICS HARDWARE INFORMATION 4

Proper use of the 6000 Basic Graphics Package routines by the applications programmer requires a general knowledge of the graphics hardware. This section describes those system characteristics which are used by the 6000 Series Interactive Graphics System applications interface routines.

GENERAL DESCRIPTION

The graphics system provides an interface for the handling of graphic or alphanumeric information; entries or modifications made at the console are placed into the 1700 Computer in digital form, and become available for use by the 6000 Series computer system. This graphics input becomes visible on the cathode ray tube and can be used for information processing by an applications program under console operator control. Results of such processing can be immediately displayed on the screen. Static display of graphic and alphanumeric data at the consoles is provided by buffer memories, so that the consoles are essentially off-line devices. The 1700 is used to process display-change information, thus saving transfer time from the 6000 Series computer.

GRAPHICS CONSOLE

The graphics console is the input/output and control center for the Interactive Graphics user. The complete range of system graphics capability can be controlled from the console without recourse to other points of control. The console is designed for maximum operator utilization and comfort, and can be used efficiently at normal room light levels.

The console cabinet is a desk-size unit which mounts a rectangular housing assembly, off-centered to the left, and provides a writing surface to the right. The housing assembly contains a magnetic shield and a 20-inch diameter cathode ray tube centered on the front panel housing.

The cathode tube is a precision, 52-degree, high-resolution unit and has a nearly flat display surface to minimize parallax error. The tube is equipped with an implosion shield for the protection of the operator, and is coated with a two-layer P-7 phosphor. One layer produces blue-violet light with a short persistence to facilitate light-pen tracking. The other layer produces yellow-green light, and has a longer persistence to eliminate flicker. With a continuously refreshed display, the light from both phosphor components combines to appear light blue to the human eye. The deflection yoke and driving circuitry of the console are designed to make the entire 314 square inches of cathode ray tube surface available for display. The tube has a resolution of 1000 lines in 20 inches.

Data can be entered on the cathode ray tube via the light-pen or one of three optional keyboards.

CONTROLS

The controls available to the console operator include the keyboards, light-pen, light registers, and light buttons. The light registers and light buttons are defined by the application program and formed for display on the screen by the 1700 Basic Graphics Package routines.

FUNCTION KEYBOARD

The 16-key function keyboard can be used to tell the application program that an operation is requested (see Figure 4-1).

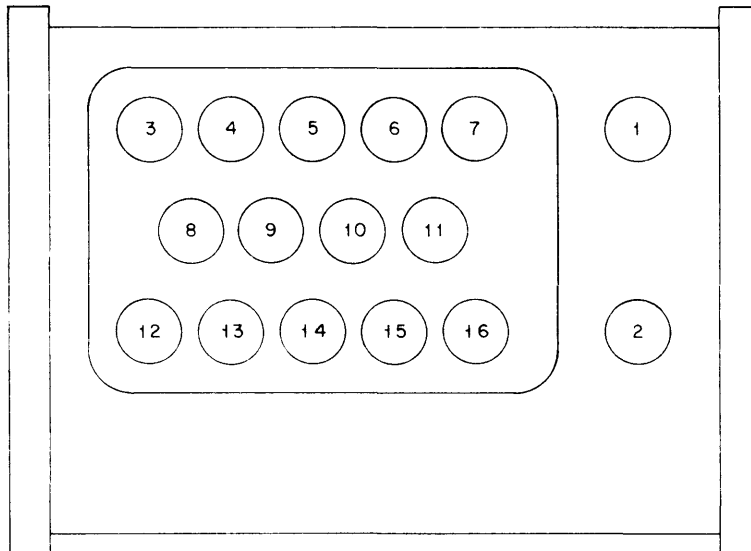


Figure 4-1. Function Keyboard

Fourteen buttons contain a snap-action switch that remains on after an initial press, and off after being pressed again. The remaining buttons must be held down to give an "on" status. Each button has an internal light that shows the operator when the button is on. Removable plastic cards may be placed over the keys to label the function of each. All keys can be given new functional assignments by the application program through the 6000 Basic Graphics Package.

Any change in the status of a key produces an interrupt at the 1744 Controller. The 1700 Basic Graphics Package then fetches the on/off status of all 16 keys as bits in a status word. These status bits are placed in the IH and IV coordinate locations of a display item ID block (see Section 6) created for the keyboard by the application program through a call to the GIKYBD routine of the 6000 Package. Table 4-1 shows the relation between the coordinate bits and the keys; a 1 in a coordinate bit indicates that the button is on.

TABLE 4-1. FUNCTION KEYBOARD STATUS IN IH, IV

Coordinate Bit		Keyboard Button
IV	0	1
	1	2
	2	3
	3	4
	4	5
	5	6
	6	7
	7	8
	8	9
	9	10
	10	11
	11	12
IH	0	13
	1	14
	2	15
	3	16

The application program retrieves the ID block through the Application Executive, GIFID, GIFSID, GIBUT, or AELBUT routines of the 6000 Package, and then determines the function requested by testing the values of the coordinates

ALPHANUMERIC KEYBOARD

The alphanumeric keyboard (see Figure 4-2) provides typewriter-like symbolic input to the application program. The keyboard layout is similar to that of a conventional teletypewriter. Each key causes an interrupt at the 1744 Controller, and enters an 8-bit ASCII character code in the left-hand portion of a status word that is fetched by the 1700 Package. The characters are collected into line images and displayed on the 274 Console screen in the currently defined light register.

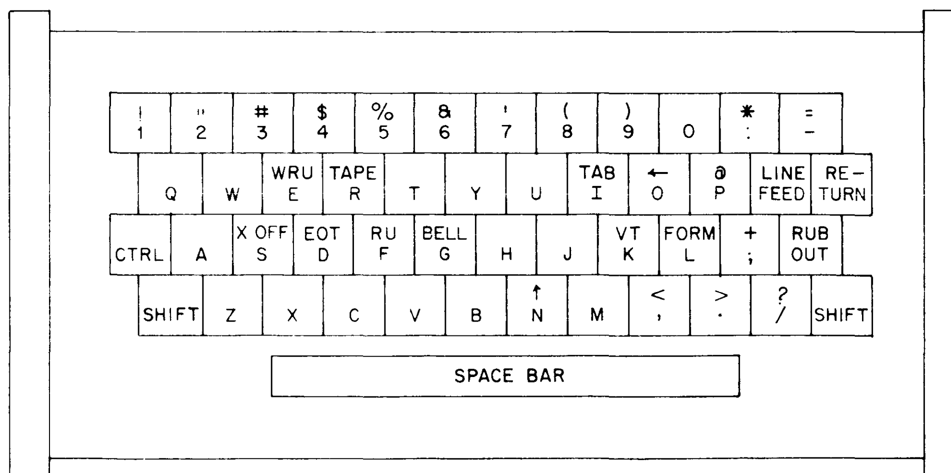


Figure 4-2. Alphanumeric Keyboard

The application program can acquire the console's input through calls to the 6000 Package GIANS and GIANE. If the Package GIEOM routine has been used to assign an ID block to a particular keyboard character, that character will clear the register when it is entered.

NUMERIC KEYBOARD

The numeric keyboard is used in the same manner as the alphanumeric board.

LIGHT PEN

The light pen has two functions: tracking, and picking. Tracking may be used to place a light source (the tracking cross) at any desired position on the console screen so that a graphic entity may be created there, or to designate that position as an area of interest to the user. Picking may be used to select an entity currently being displayed, or to define points on a displayed entity, and to select a light button or tracking cross.

LIGHT REGISTERS

The light registers allow the user to input and retrieve alphanumeric information, and permit the Interactive Graphics System to display error diagnostic messages. The number and locations of the registers are lefined by the application program through 6000 Basic Graphics Package GUAN calls. If none have been defined, the System defines its own at the center of the screen (for error messages); otherwise, the last one defined by the program is used for System messages.

LIGHT BUTTONS

The light buttons are light spots on the console screen that are identified by a letter, digit, symbol, or instruction code specified by the application program. Any displayed entity or physical control key can also be defined as a light button. Buttons are used to control ID block queueing (see Section 6) and to initiate tasks.

DISPLAY PRESENTATION

The entire 20-inch diameter cathode ray tube screen can be used for display presentation. Points on the screen are addressed by a Cartesian coordinate system called the display grid.

DISPLAY GRID

The display grid (see Figure 4-3) consists of 4095 addressable points on the horizontal (H) axis, and 4095 addressable points on the vertical (V) axis; coordinates can be given either octally or decimally when addressing a point. Coordinate 7777_8 equals coordinate 0000 on both axes.

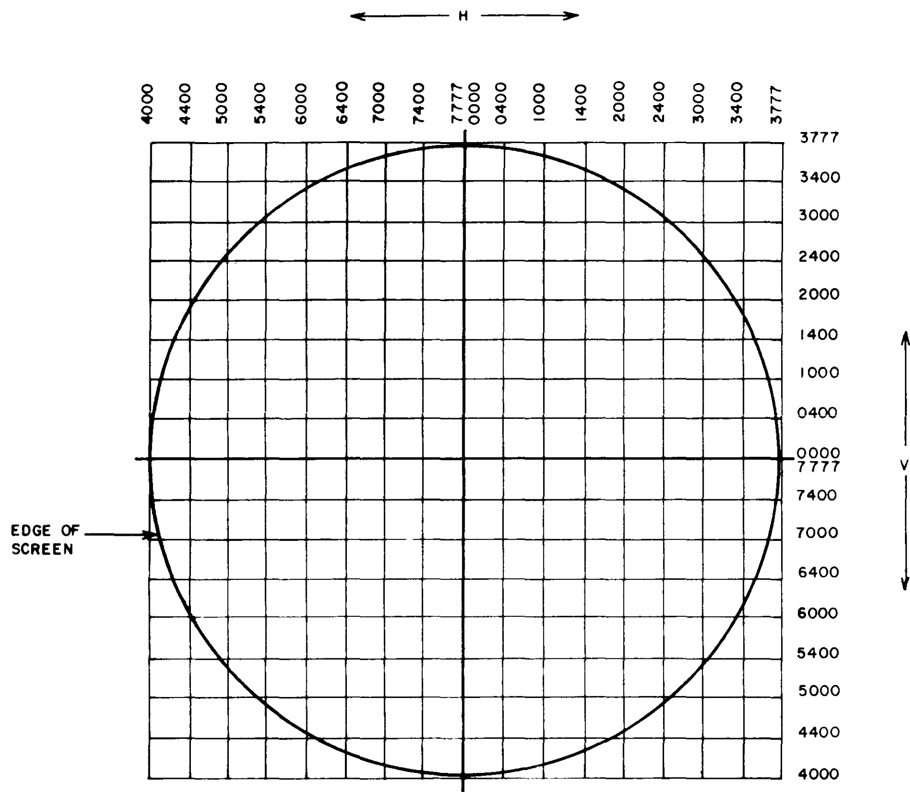


Figure 4-3. Display Grid System

The grid is larger than the screen so that all points on the screen can be addressed; points beyond the edge of the screen can be addressed by a programmer, but are invisible to a user directly in front of the screen (if viewed at an angle, such points can be seen reflected off the side of the tube). There are 200 grid points per linear inch; however, because the cathode beam is wider than the distance between adjacent points, the console controller drops the least significant bit from each coordinate address of a point. Note: The console controller may vary a plus or minus five DGUs depending on the C. E. 's set adjustments.

SCREEN ORGANIZATION

The organization of the screen is completely up to the programmer. However, certain conventions may be used for a wide variety of applications. These conventions allow a programmer to make maximum use of the screen area, yet help him avoid addressing grid coordinates off the screen.

WORKING SURFACE

One convention is to divide the screen into a working surface and a control surface. The working surface is reserved for the display of graphic forms, and is contained within an undisplayed frame or frames defined by the programmer (see GULINE and GUARC, Section 7).

CONTROL SURFACE

The control surface is defined as the area outside of the frame or frames, and is normally reserved for light buttons, light registers, and the tracking cross (when it is not in use on the working surface). Figure 4-4 shows a sample of one type of screen organization.

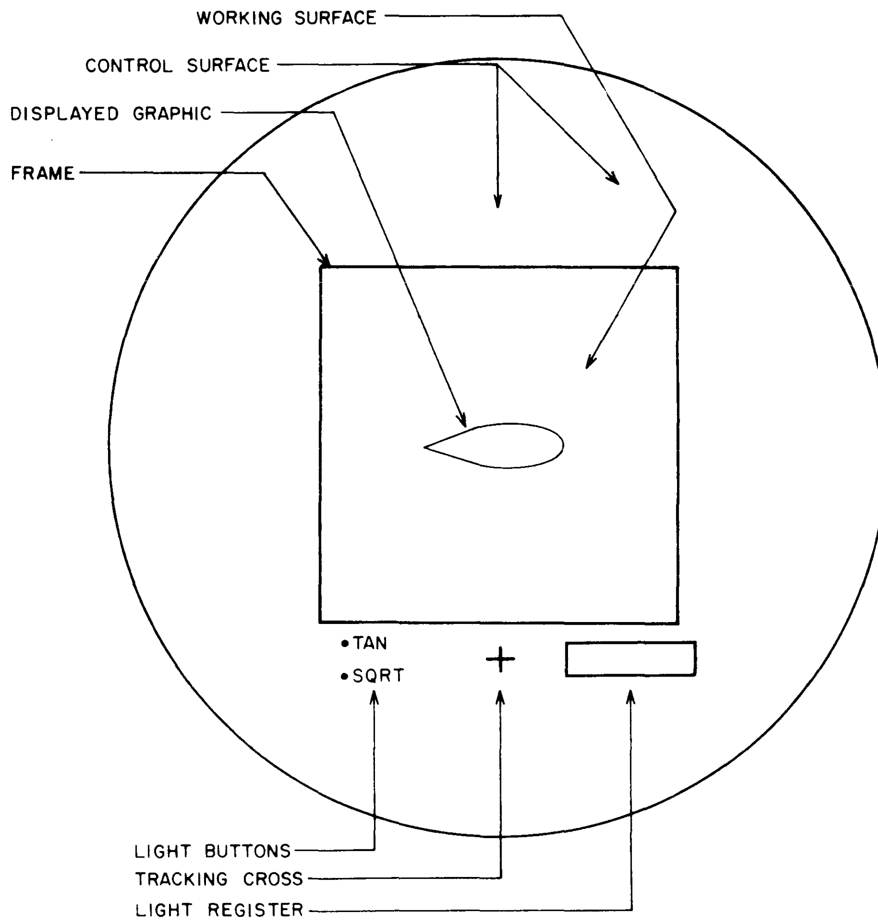


Figure 4-4. Sample Display Surface Organization

FRAMES

Table 4-2 defines possible frames within the screen area of the display grid. Several frames may exist on the screen at the same time; they may overlap, or each figure may have its own frame. The system software defines all frames as right rectangular areas, and frames may be centered anywhere on the screen.

TABLE 4-2. SAMPLE FRAMES

Frame Size	Center Coordinates	Right Corner Coordinates
Maximum square, 14 by 14 inches	IHCEN = 0000B IVCEN = 0000B	IHCOR = 2570B 1434 IVCOR = 2570B 1434
Horizontal rectangle, 11 by 17 inches	IHCEN = 0000B IVCEN = 0000B	IHCOR = 3244B 1741 IVCOR = 2114B 1126
Vertical rectangle, 17 by 11 inches	IHCEN = 0000B IVCEN = 0000B	IHCOR = 2114B 1126 IVCOR = 3244B 1741

POTENTIAL PHOSPHOR DAMAGE

It is possible for programming errors to cause endlessly repeated or excessively intense display of an item at the same location on the screen. This may cause damage to the cathode ray tube phosphor. When one of these conditions is detected by the console operator at program debugging time, the console must be turned off immediately by use of the console power switch above and to the right of the screen. Console power status does not affect the operation of the computer or of the Interactive Graphics System.

1744 DIGIGRAPHICS CONTROLLER

The controller uses a standard 1700 memory module (1708) of 4096 16-bit words for a buffer memory, with an option for an additional 4096 words. A 1700 programmer may use the buffer memory as a display buffer or as an auxilliary 1700 Computer storage device.

As a display buffer, only bits 00 through 12 (with the exception of function and status codes) contain meaningful data. As an auxilliary random access storage device, all 16 bits can be used.

REGISTERS

The controller's 13-bit S register is used to address the memory module(s) and select the locations which are to be read or written; its contents can be incremented by one or jumped to a new value. Thirteen bits are required to address all 8192 memory locations.

The P register is also a 13-bit register. Its function is to maintain the address of a program within a main program. This register can also be incremented by one or jumped to a new value.

The Z register is a 16-bit register and is the central data holding register of the controller. All data except function codes and interrupt status passes through the Z register. The S, P, and Z registers act and interact via sequence control logic in the controller to execute the S jump, P jump, M jump, RTM, and end-of-display-byte-stream commands described below.

COMMAND BYTES

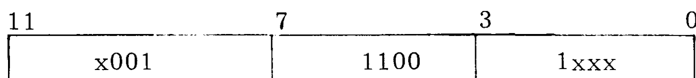
The Graphics Hardware Interface and Graphics Utilities routines of the 6000 Package in effect write or erase two classes of bytes within the 1744 memory — memory command bytes and control bytes. These bytes are right-justified in the 16-bit memory word and make up the display byte stream.

Command bytes are roughly equivalent to program instructions in a computer. There are eight kinds of command bytes:

- S jump
- P jump
- M jump
- RTM
- EDB
- ROD
- sense
- no-sense

S JUMP

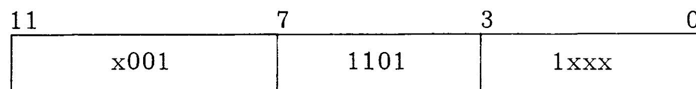
The S jump command byte is equivalent to an unconditional jump instruction. Its format is:



x Any value; this can be written as 0710B

P JUMP

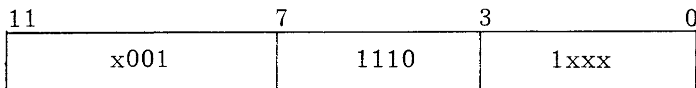
The P jump byte is used to exit from one macro routine to another. The format for this byte is:



x Any value; this can be written as 0730B

M JUMP

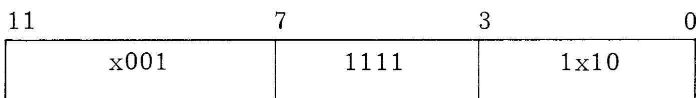
The M jump byte corresponds to a return jump instruction. It's format is:



x Any value; this can be written as 0750B

RTM

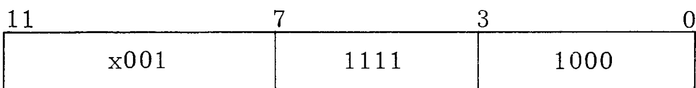
The RTM (return to main) command byte is used to exit from an M jump and return to the main display program in the buffer memory. The format of RTM is:



x Any value; this can be written as 0772B

EDB

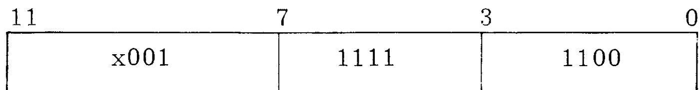
The EDB (end-of-display-byte-stream) byte signals the 1700 that on-line editing by the 1700 Basic Graphics Package can begin. The EDB format is:



x Any value; this can be written as 0770B

ROD

The ROD (return-to-off-line-display) byte terminates 1700 editing and returns control of the 1744 to the contents of its buffer memory. The ROD byte has the format:

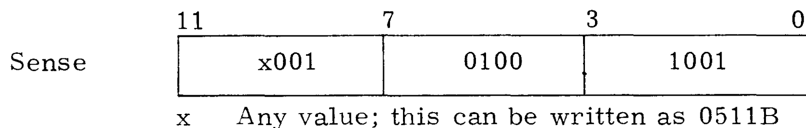


x Any value; this can be written as 0774B

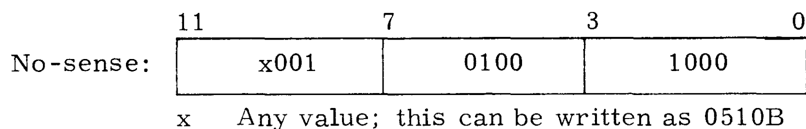
SENSE/NO-SENSE

The sense or no-sense command byte precedes each item's display byte stream to enable or disable the item's sensitivity to a light-pen pick. The pick of an item with a sense byte

preceding its display byte stream causes the controller to send an interrupt signal to the 1700. The pick of an item having a no-sense byte is ignored. The formats for these bytes are:



x Any value; this can be written as 0511B



x Any value; this can be written as 0510B

CONTROL BYTES

While the command bytes determine the order of item display, the control bytes form the description of each item and determine how it appears on the screen.

There are three kinds of control bytes processed by the 1744 Controller:

- Reset
- Control
- Increment

RESET

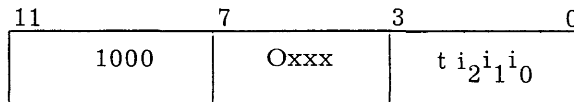
The reset byte (see GURSET, Section 7) is a secondary beam-moving byte; it is usually the first byte in any series, and is a positioning byte which moves the cathode ray tube beam to an approximate position on the display surface. Reset bytes are usually followed by increment bytes within the byte series and establish the precise point for display initiation. The reset byte resets the beam to one-half normal intensity.

The reset byte controls beam intensity, light pen sense, blink and terminate to reset. The reset byte initiates a 25 μsec time delay and processes the bytes as follows:

- The first byte is interpreted as a 12-bit X location which will be transferred to the X console interface register.
- The second byte is interpreted as a 12-bit Y location which will be transferred to the Y console interface register.

The 30 μsec time delay has to elapse before the first increment byte is processed.

The format for the reset byte is:



11	7	3 0
1000	Oxxx	t $i_2 i_1 i_0$

where:

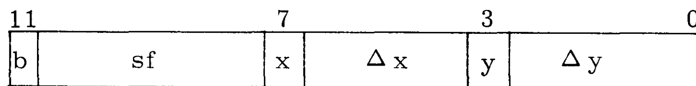
S	T	i_2	i_1	i_0	Condition
0					Light pen sense disable
1					Light pen sense enable
	1				Terminate to nest reset
		0			Blink disable
		1			Blink enable
			0	1	Intensity dim
			1	0	Intensity medium
			1	1	Intensity bright

When bits 11 through 8 equal 1000, bits 7 through 4 are placed in the high order bit positions of the controller's X accumulator, with the remaining bits set to zero. Bits 3 through 0 are placed in the high order positions of the controller's Y accumulator, with the remaining bits set to zero. (The X and Y accumulators contain the H and V coordinates used to aim the cathode beam.) The reset byte does not affect the on/off state of the beam, which remains in the same state as specified in the previous byte.

INCREMENT

The increment byte is the primary beam-moving control byte. It is used for fine positioning of the cathode ray tube beam. In normal operation, an initializing reset byte is used for coarse positioning of a displayed item. With the beam off, increment bytes establish the precise display starting point (followed by an intensity byte if a change of intensity is desired). Once the desired starting point is established, increment bytes (the first of which turns the beam on) display the desired graphics.

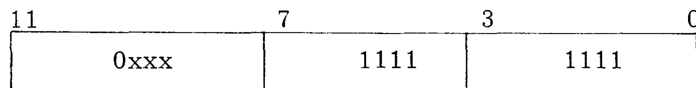
The format for the increment byte is:



Bit 11 controls the beam state (0 = off, 1 = on); bits 10, 9, and 8 are weighted scale factors, used to multiply the Δx and Δy values that specify the precise position of the beam. The scale factors range from two to seven, corresponding to changes of 1:1 to 32:1, respectively.

Bit 7 is the X sign bit and bits 6, 5, and 4 are added to the current X accumulator value after being shifted right the number of bit places specified by the scale factor. Bit 3 is the Y sign bit and bits 2, 1, and 0 are added to the current Y accumulator value after being shifted right the number of bit places specified by the scale factor.

A delay function is performed by a special case increment byte. The format:



x Any valid scale factor

produces a 25 μsec delay to provide time for beam settling, and is used after a reset byte (see GURSET, Section 7).

Table of Increments

SF = 7,	32:1 =	0.16 inch	per 1 unit in Δx or Δy
SF = 6,	16.1 =	0.08 inch	
SF = 5,	8:1 =	0.04 inch	
SF = 4,	4:1 =	0.02 inch	
SF = 3,	2:1 =	0.01 inch	
SF = 2,	1:1 =	0.005 inch	

The information on command and control bytes supplied in the preceding paragraphs is not necessary for normal programming in the Interactive Graphics System; it is provided here

for the convenience of a programmer using the GUBYTE routine described in Section 7. A more detailed discussion of programming the graphics hardware is beyond the scope of this manual.

DISPLAY MACROS

Often used display items can be placed in the buffer memory as macros to better utilize its core space. The macros can then be called on command. A macro call stores a return address in a hardware register, which leads to a transfer of display control to the addressed macro. The macro either returns directly to the return address or transfers control to another macro addressed from the stored location. Since the return address is held in a hardware register, only one level of macro is provided. A macro may not include a call to another macro. Macros provide efficient access to associated byte-streams such as alphanumeric strings.

Macros are used primarily to conserve memory in the 1744 display buffer, although their use benefits the application programmer as well. If the programmer requires the use of a particular display item at more than one point on the screen, the most convenient way to duplicate it is to classify the item as a display macro by generating its byte-stream with a GIMAC call (see Section 7).

Subsequent calls to GUMACG then generate a short calling sequence for the macro; this sequence contains an M jump to the macro area of the 1744, where the byte-stream of the item is stored for execution. The RTM address byte at the end of the byte-stream returns control to the buffer area for regular display items.

The programmer can use macros in at least two ways. If it is necessary to display an item intermittently at the same location on the console screen, the item can be defined as a macro and can contain a reset sequence (see GURSET, Section 7). Each time a regular display item calls the macro, it will be displayed at the same location; each time the calling item is erased, display of the macro stops.

A macro can also be used without a reset sequence to define an item that is easily relocated at any point on the screen. This type of macro is used with the tracking cross (see GITIMV and GITMMV, Section 7).

DISPLAY BUFFER MEMORY LAYOUT

The first two-byte sequence of the display buffer contains an S jump when alphanumeric information is being entered at the console. The jump is made to the Alphanumeric Pick Display Area to display the characters that have been entered. If there is nothing to display, an S jump is made to continue processing the main display. Figure 4-5 illustrates the memory via a block diagram.

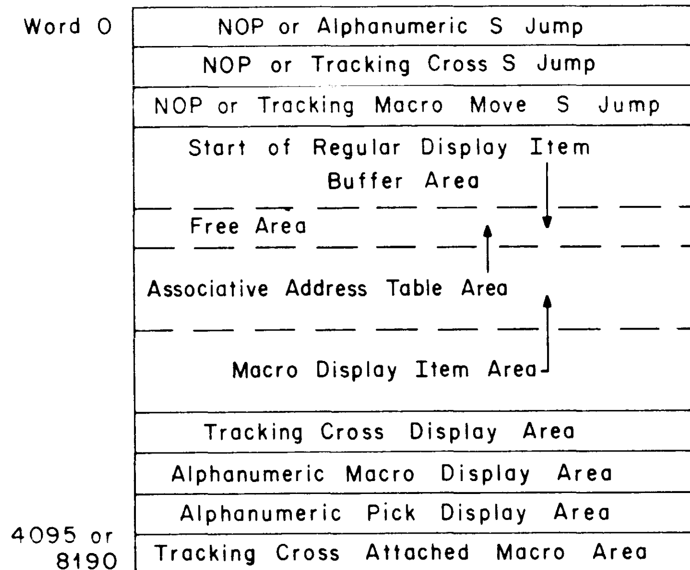


Figure 4-5. Display Buffer Block Diagram

The second two-byte sequence of the buffer always contains an S jump when the tracking cross is being displayed and no tracking is taking place. If the tracking cross is not up, an S jump is made in the tracking display area to prevent the tracking cross bytes from being processed.

The third two-byte sequence contains an S jump only when a macro is attached to the tracking cross (see GITMMV, Section 7).

The first six bytes of the display buffer contain NOP (zero ID control) bytes for all other cases. This allows the memory scan that produces the display to proceed into the regular display item area without executing a jump.

The regular display item area floats in memory, and is expanded upward as needed.

The Associative Address Table (see Section 7) is stored at the end of the Macro Display Item Area. This table also floats in memory, and is expanded downward as needed.

The Macro Display Item Area is fixed in memory but can be expanded downward as needed, moving the Associative Address Area at the same time. When a macro is erased, no contraction of the macro area occurs.

The last four areas are fixed in memory, and are assigned to the 1700 Basic Graphics Package for use during tracking, alphanumeric output to the console, alphanumeric picking, and tracking macro operation, respectively.

1700 GRAPHICS FUNCTIONS 5

BUFFER TRANSLATOR

The buffer translator is called by IMPORT when the 1700 receives a data buffer from EXPORT or is ordered to send a data buffer to the 6000 Series computer.

The translator program will unpack the EXPORT buffers and put the calling parameters of the 6000 Basic Graphics Package into a format that the 1700 Basic Graphics Package will recognize. The translator also loads buffers for transfer to the 6000 Series computer from the 1700 Basic Graphics Package. All alphanumeric characters are code converted by the translator into or from 1700 internal code. Floating-point conversions are done in the 6000 Series computer by the 6000 Package routines.

PROGRAM ABORTING

The translator is also responsible for aborting graphics programs at the 1700. If a 1700 Package routine attempts to communicate with a console but the console's driver routine detects a communication error or failure, the Package routine sets a flag to inform the translator of the condition. The translator then displays an appropriate message (see Table 9-6) on the teletypewriter and sends IMPORT directive code 23 to the 6000 (see Graphics Program Aborting, Section 2).

The translator also aborts programs if it detects an invalid IDDAD, IDDADI, or MAD programming parameter while it is processing a buffer from EXPORT. In this case, the translator returns a 1700 ABORT message to the 6000, displays an appropriate message (Appendix B) on the screen of the affected consoles and at the teletypewriter, and sends the IMPORT directive code to EXPORT.

If the Digigraphic Interrupt Processor of the 1700 Package detects an error condition while attempting to process console input or output, it also sets a flag for the translator. The translator then types out one of the two reject messages given in Table 9-6, and aborts the job in the same manner as given above for a console driver error.

1700 BASIC GRAPHICS PACKAGE

The 1700 Basic Graphics Package contains a set of graphics routines and a queue handler to process light-pen/keyboard picks and save tracking-cross positions.

The functions and philosophy of the 1700 Basic Graphics Package routines are similar to those of the Graphics Utilities and Graphics Hardware Interface routines of the 6000 Basic Graphics Package. The calling statements for both sets of routines are identical; two Packages are used solely to prevent tying up the 6000 Series computer with the detail work necessary to service a display console.

The applications programmer is concerned only with the 6000 Basic Graphics Package. He writes his programs in parametric form, and the 6000 Package then passes these parameters (via EXPORT/IMPORT and the Buffer Translator) to the 1700 Basic Graphics Package, which uses the data to actually drive the cathode ray tube of its associated graphics console.

Specific information regarding the functions of the 1700 Package routines is beyond the scope of this manual.

SYSTEM EXPANSION

The Interactive Graphics System can be expanded by the addition of routines to the Graphics Utilities library of the 6000 Basic Graphics Package (Section 7). Although such additions could be made without corresponding changes in the 1700 Package, the efficiency of the System would be increased by the addition of a corresponding 1700 Package routine for each routine added to the 6000 Package. This approach would simplify 1700 error processing.

Additions to the two Packages can be made without changes in any of the other parts of the System software.

DISPLAY ITEMS AND PICK PROCESSING 6

Every item that the programmer creates on the display screen has identification information associated with it in the 1700 Computer's memory, as does every console input device that he wishes to have his program service. This information includes parameters from the 6000 Basic Graphics Package calls which the programmer uses to create and manipulate the item or to define the functions of the device. These parameters (and other pick processing information) are organized into a structure called a display item ID block.

DISPLAY ITEM ID BLOCK

The 1700 Basic Graphics Package maintains a buffer of the item ID blocks created by the programmer (see Section 7) as shown in Figure 6-1.

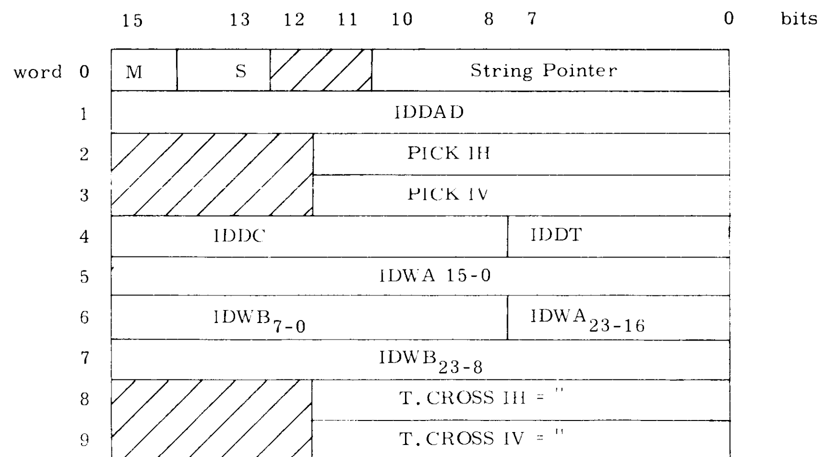


Figure 6-1. Display Item ID Block in 1700

The ID block is the basis of all graphics input processing. The four ID quantities IDDT, IDDC, IDWA, and IDWB are defined and used by the programmer. The display item type code IDDT is also used by the queue handler and the 1700 interrupt processor mask comparison routines (see GIMASK, Section 7).

The IDWA/IDWB of a light button would normally contain the name of a task to be called by the Application Executive AETSKR routine; the task name is left-justified, beginning in IDWA. The IDWA/IDWB of a graphic figure would contain a data bead address (see Data Handler, Section 7) as its last five characters.

The contents of IDDT and IDDC cannot exceed 8 bits (377B) each; IDWA and IDWB cannot exceed 24 bits (77777777B) each. IDDT = 0 is reserved for alphanumeric input only.

ID blocks may be associated with other graphic input devices, as well as the items on the display. These are:

- The console function keyboard.
- An alphanumeric End-of-Message character.
- The switch on the light-pen.
- The pick of some display item of a particular type; this results in two ID blocks being queued (for example: a regular display item may also be conditioned to act as a button; see GIPBUT, Section 7).

To have an ID block from one of these devices input to the application program, the IDDT of the device must classify it as one of the three types of pick information processed by the queue handler:

- Single pick information
- String pick information
- Button pick information

QUEUE HANDLER

Since the console operator will get ahead of the application program's execution, it is necessary to have a means of allowing the operator to use the light-pen, keyboard, and tracking-cross at his own speed, but still enable the graphics software to keep track of the picks and tracking-cross coordinates for later use by the application program. The queueing mechanism which accomplishes this,reduces the time a console operator must wait after making a request until he can make another request.

PICK TYPES

The picks made by the console operator are queued as four types of ID blocks before being passed to the application program:

- Single pick type - only the copy of the ID block for the latest single pick display item chosen is kept in the queue, regardless of how many such items are picked.
- String pick type - one copy of a string pick display item ID block is kept in the queue for each time such an item is picked.
- Alphanumeric type - includes alphanumeric characters picked by either the light-pen or a keyboard key; queued in the same manner as a string pick type.
- Button pick type - one copy of the ID block for a light-button is kept in the queue for each time such an item is picked. The button pick ID is similar to the string pick ID except that a button pick may reactivate an idle application task.

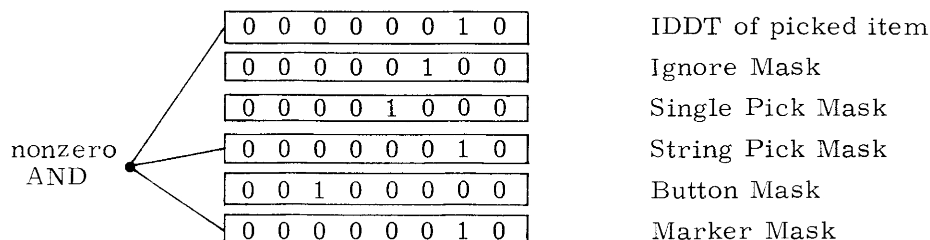
The single pick ID block and the string pick ID block are associated with the button pick ID block, and may contain tracking-cross coordinates along with other ID information.

QUEUE HANDLER FUNCTIONS

When the 1700 Basic Graphics Package interrupt processor detects a light-pen or keyboard pick,it turns control of the 1700 Computer over to the queue handler. The queue handler then performs three actions:

1. Does an ID read of display memory to determine which item has been picked.
2. Logical ANDs the IDDT of the pick ID block with the set of ID processor masks (see GIMASK).
3. Performs the queue operation specified by the result of the ANDing.

If the logical AND of the IDDT and the set of masks is nonzero, the queue handler will place the ID block of the picked item on the end of the appropriate queue string. For example, after the AND of the following IDDT and masks values, the ID block involved is placed at the end of the set of queued string picks and the blink byte in the reset sequence is complemented. That is, a nonblinking item will blink and a blinking item will no longer blink.



In each of the following cases, the ID read processing differs from that of a normal light-pen strike. However, steps 2 and 3 above remain the same:

- If GILPKY has been called and a light-pen switch interrupt occurs, the queue handler will read the assigned ID block from a table in 1700 memory.
- If GIKYBD has been called and a keyboard interrupt occurs, a 1700 memory ID read will be performed.
- If GIEOM has been called and an EOM key press or an EOM font pick causes the interrupt, a 1700 memory ID read will be performed.
- If GIPBUT has been called and a prime button pick causes an interrupt, both a 1700 memory and a 1744 display memory ID read will be performed.

The queue handler also retrieves ID blocks from the FETCH queue (see below) when they are requested by a 6000 Basic Graphics Package GIBUT or Application Executive AETSKR call.

FETCH AND WAIT QUEUES

There are actually two separate queues maintained in the 1700's memory for each graphics console — the FETCH queue and the WAIT queue.

The WAIT queue serves as a temporary console input buffer in which to arrange and complete a set of picked ID blocks. The WAIT queue is not accessible to the application program; this prevents the program from receiving an incomplete set or string of pick ID blocks if it requests transfer of the blocks to the 6000 while the console user is still building a string or editing a set of queued blocks.

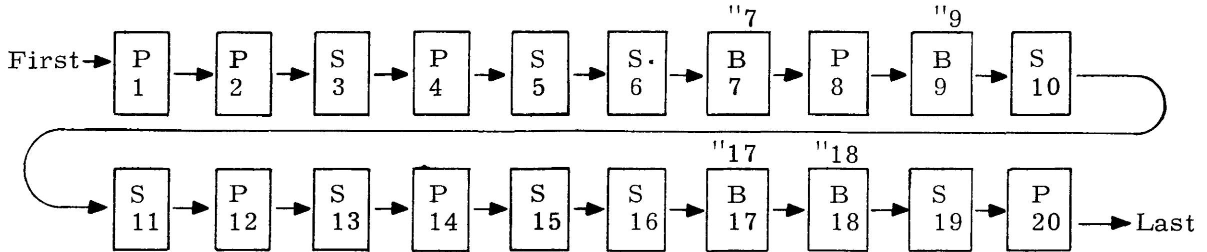
A button pick automatically transfers the ordered blocks from the WAIT queue to the FETCH queue. The blocks are then passed to the program in the 6000 Series computer from the FETCH queue.

Whenever an item is erased from the display, both queues are scanned for a pick of the erased item. If the item is a single pick type or string pick type and is erased, the ID block is spliced out of the WAIT queue. If the erased item is in the FETCH queue as a button, the button ID block and its associated single pick and string pick ID blocks are all removed.

QUEUE MECHANISM OPERATION

The following set of diagrams illustrates the logical mechanism used by the queue handler to queue picks and buttons. Each square represents a core block of ID information, pointers, and coordinates. The queueing of ID blocks is controlled by the application program through the setting and clearing of type code masks (see GIMASK, Section 7).

Time History if maintained as a simple queue:



" Tracking cross coordinates

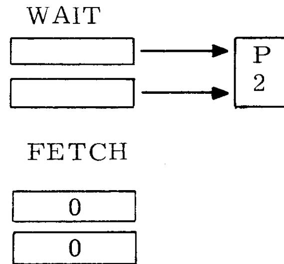
→ Flow of execution

P Single Pick ID

S String Pick ID

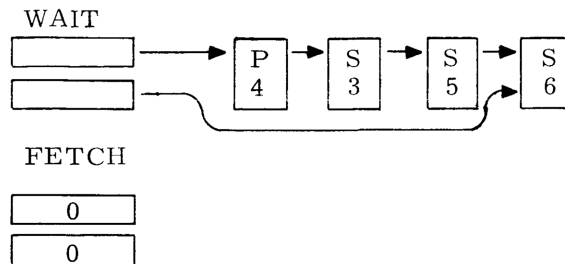
B Button Pick ID

Representation after Single Picks 1 and 2:



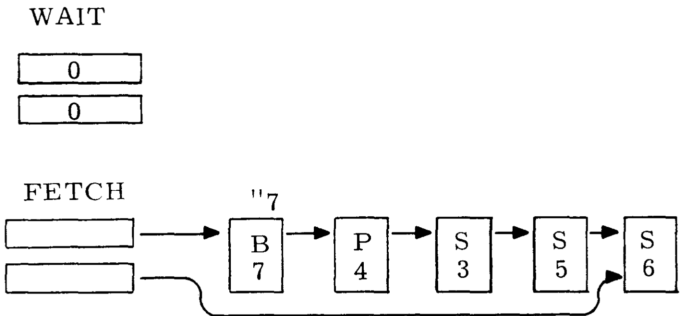
Single pick ID blocks are always placed at the start of the WAIT queue, and replace whatever single pick ID block may have been there.

Representation after String Picks 5 and 6:



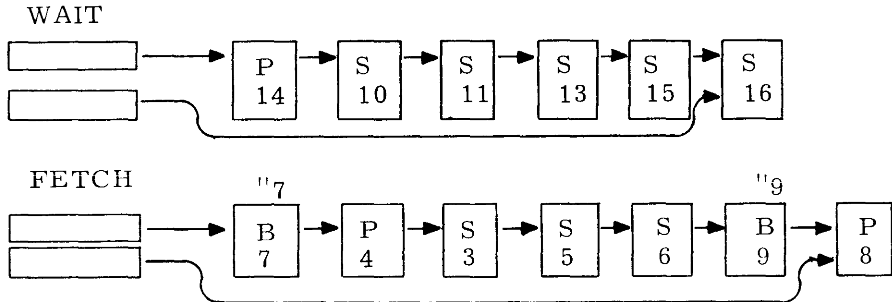
String pick ID blocks are always collected at the end of the WAIT queue.

Representation after Button Pick 7:



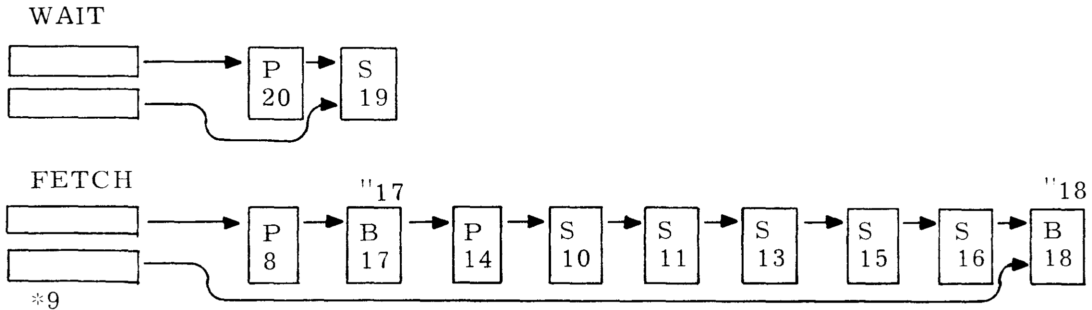
Button pick ID blocks are always placed at the end of the FETCH queue followed by the contents of the WAIT queue (which is then cleared).

Representation after String Picks 15 and 16:



If two string pick ID blocks are the same (IDDAD all identical) both blocks are removed. If string picks 11 and 16 were the same in the above example, block 11 would be spliced out of the WAIT queue and block 16 would not be saved. This feature allows limited editing of picked items without application program intervention.

Representation after Pick 20 Is Input and Button 9 Is Fetched by the Application:



The tracking cross coordinates of each button ID are saved for application reference. The tracking cross coordinates fetched by GITCOF are those from the last button ID pressed to the application program (*9 in the example above).

ID blocks passed to the application program are always picked up from the start of the FETCH queue (block 8 in the last example above), and are passed only as the result of a specific request from the application. If a button fetch (CALL GIBUT) is requested at the time of the above representation, ID blocks 8 and 17 are made available to the application.

Next, the single, string and button masks are checked in that order. The first nonzero product causes the designated queueing operation and a check of the marker mask. If the marker mask also causes a nonzero product, the queue handler will perform the marking function on the item. The blink byte will be reversed in order to change the appearance of the item while it is queued until it is fetched.

As each ID block is removed from the queue, the marker bit is checked and the blink byte is restored to the status it had before it was queued. This notifies the console user that his pick has been sent to the 6000 Series Computer.

6000 COMPUTER PICK PROCESSING

The 6000 Series computer receives ID blocks from the 1700 Computer only after sending it a GIBUT call, or an equivalent call from the Application Executive. Only the first button ID block, and any single or string pick blocks preceding it in the FETCH queue, is sent to the 6000's EXPORT program.

The ID blocks are all sent by EXPORT to the Application Executive area of the calling graphics job, where GIBUT unpacks the button ID block information and stores it for later use by the 6000 Basic Graphics Package AELBUT and GITCOF routines. The other ID blocks are stored for later use by the GIFID and GIDSID routines.

Each time a button pick is fetched from the 1700, the new ID blocks are written over the blocks stored in the 6000 by the previous fetch. The queue handler masking procedure is ordered. The ignore mask is anded with IDDT first. If the product is nonzero, no further action is taken.

6000 BASIC GRAPHICS PACKAGE 7

The 6000 Basic Graphics Package is a set of subroutines, written in COMPASS assembly language, designed to provide an interface between the applications programmer and the graphics hardware. The Package coexists with SCOPE; an applications programmer has full access to both.

This Basic Graphics Package has four main functions: to provide the ability to manipulate display items, to control light buttons, to input and output alphanumeric data, and to supply the necessary tools for creating and handling a data structure.

Using the Basic Graphics Package routines, the simplest application program can send display items to the consoles. These display items are described in a language one level higher than the standard display language. For instance, a circle in display language is a stream of DXs and DYs; however, the Basic Graphics Package, using the 6000 Computer, describes a circle in parameter form. The 1700 has the ability, which it is more suited to accomplish, to convert this parameterized data into display language — by using the 1700 Basic Graphics Package (see Section 5).

ROUTINE TYPES

The 6000 Basic Graphics Package routines are divided into four categories:

- Graphics Hardware Interface
- Application Executive
- Graphics Utilities
- Data Handler

GRAPHICS HARDWARE INTERFACE

The Graphics Hardware Interface is a set of library subroutines that permit application program control of the display hardware. The functions performed by the Graphics Interface define the graphics capabilities available to a user. The interface includes routines to edit the display buffer display items, control light-pen and keyboard inputs, control light-pen tracking, and collect alphanumeric text input. All interface routine names begin with GI.

APPLICATION EXECUTIVE

The Application Executive controls the residence, sequencing, and execution of tasks; it includes the equivalents of SCHEDR, GIBUT, and GIABRT.

The Executive is written as a single, eight-part program called MAIN. When a programmer uses MAIN as part of his zero-level overlay, his subsequent calls to AETSKC and AETSKR in any task overlay result in calls to the appropriate part of the MAIN program.

FUNCTIONS OF MAIN

MAIN is entered as a FORTRAN subroutine from the application program's zero-level overlay, using a CALL MAIN card (see Section 2), during both the file creation and execution runs of the job.

MAIN first reads the file name parameter cards in the application program's next data record.

If the data record contains two cards, MAIN:

- Writes the graphics task COMMON file name (from the first card) in RA+2 of the program's current control point area
- Writes the overlay source file name (from the second card) in RA+3
- Terminates the LGO portion of the job so that AEFILE can create the program's graphics COMMON file

If MAIN finds only one card in the first data record, it assumes that the job is to be executed during the current run. MAIN then:

- Opens the task file named on the card
- Reads the task directory pointer to determine the amount of central memory needed to load the longest overlay in the task file
- Changes the field length
- Calls the Scheduler to assign the program to a graphics control point

The Scheduler then rolls out the job (see Section 2). When the Scheduler rolls the job back in, MAIN reads the first record of the task file into central memory. Control of the central processor is then transferred to the task in that record.

MAIN is again entered when an AETSKC call occurs; it then locates the requested task within the task file, reads it into central memory, and transfers control to it.

When an AETSKR call occurs, MAIN requests a button fetch from the 1700 FETCH queue, and waits until one has been returned. When a button pick type ID block (and its associated string and single pick blocks) is returned, control of the central processor is turned over to the task overlay named in the IDWA and IDWB parameters of the button's ID block.

MAIN also contains an abort processor that is entered any time a 6000 Basic Graphics Package routine produces an error message. The abort processor enters all diagnostic messages supplied to it in the system dayfile; the processor aborts the application job only if a fatal error or a GIABRT call has occurred.

GRAPHICS UTILITIES

The Graphics Utilities are an expandable library of subroutines for general graphics applications. Included as Graphics Utilities are routines to frame-scissor graphic figures, generate graphic figure descriptions, and collect figure descriptions for display. The utilities routine names supplied with the 6000 Basic Graphics Package all begin with GU.

DATA HANDLER

The Data Handler is a set of routines that optimize access to mass storage and perform in-core list processing. The handler permits an application programmer to efficiently create and manipulate his own unique data structure. The form of data organization used is a plex data structure.

PLEX DATA STRUCTURES

Graphics interaction places stringent demands upon the application programmer in the allocation and handling of data. In general, graphics application data is completely random in the order of its manipulation, and in the amounts of each data type stored. Conventional allocation and management schemes, such as FORTRAN arrays or card image files, are usually inappropriate and inefficient.

A concept of storage management has been defined* that meets all the requirements of interactive applications. The concept, called the Modelling Plex, involves the data, data structure, and data manipulating algorithms required to represent the physical actions required of the application. The requirements of the data and algorithms are determined by the needs of the application on one hand, and by the data structure on the other.

---

*Douglas T. Ross, AED-O Programming Manual, Section 2.2 Data Structure Language, Preliminary Release No. 2, MIL-ESL, October 1964.

A plex data structure is the most general form in a broad class of data management techniques called list structuring. In a plex data structure, all data is contained in variable length beads of contiguous computer words. The length, format, and data content of any bead is completely under control of the application programs.

The Data Handler provides a pool of empty beads (free storage) from which the application may obtain new beads and to which it can return those no longer needed. Each bead has a unique addressing parameter (IBEAD) that is supplied by the system, and used by the application programs as data. This bead address is used for referencing the data within the bead, and may be used as data within other beads as a pointer to specify related information. In general, a plex data structure contains a greater number of pointers than do more conventional storage techniques. (See Figure 7-1 for typical bead arrangements.)

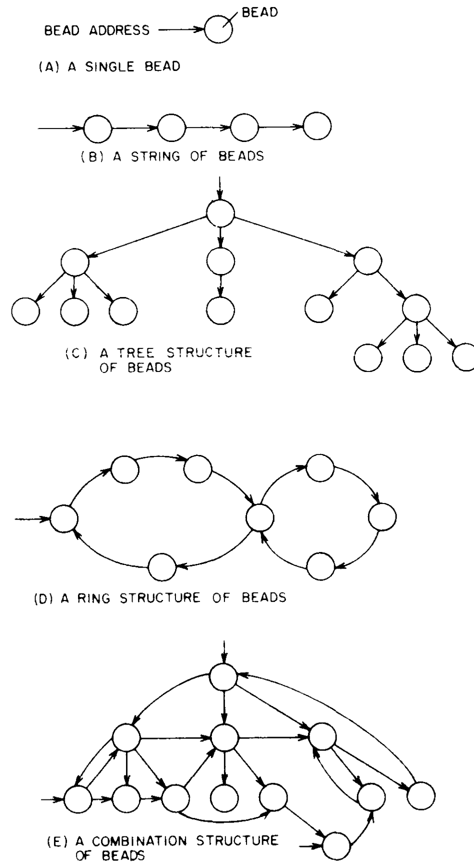


Figure 7-1. Typical Bead Arrangement

For a specific application, it is most efficient to include only the beads and pointers needed. A formal structure that includes all possible relationships of a rigid class introduces inefficiencies that cannot be tolerated in an interactive system.

For a simple example of a plex data structure, consider the dynamic parts of an automobile engine: the crankshaft, the connecting rods, and pistons. Each bead of the representation contains the needed information about a particular item. Each bead of a particular type has exactly the same length and format, but different data values; i.e., the mass parameter in each connecting rod bead — CR in the example — would appear in the same position within the bead, but would reflect the actual mass of the particular connecting rod. Note that by proper design of the plex data structure, the calculating algorithms may be quite independent of the actual representational model. In the example (Figure 7-2), a 12-cylinder engine could be handled with the same structure and programs by allowing the connecting rod string of beads to be of variable length. By convention, the initial bead address in a string of beads is called the state variable of the string, and the last bead of the string holds a zero string pointer. In a plex data structure any number of strings may be passed through a bead.

CS
CR CR CR CR
P P P P
105 MAX. H.P.
CS – CRANKSHAFT
CR – CONNECTING ROD
P – PISTON

Figure 7-2. Four Cylinder Engine

Beads are floating within blocks. The bead address IBEAD contains the block count and an index to an array of pointers within the block as follows:

23	13	8 0
Block Count	Index	(cross-hatched)

For use as a string pointer, the location within the bead of the next pointer (hook) may be placed in the cross-hatched area.

The 9-bit hook limits string pointers to the first 511 locations in a bead. The Data Handler accepts full 24-bit addresses as a bead address, and will ignore the low order 9 bits on all but string operations.

The Data Handler allows simple FORTRAN programming string operations. Figure 7-3 is an example of list structuring. Hooks are shown with a broken line.

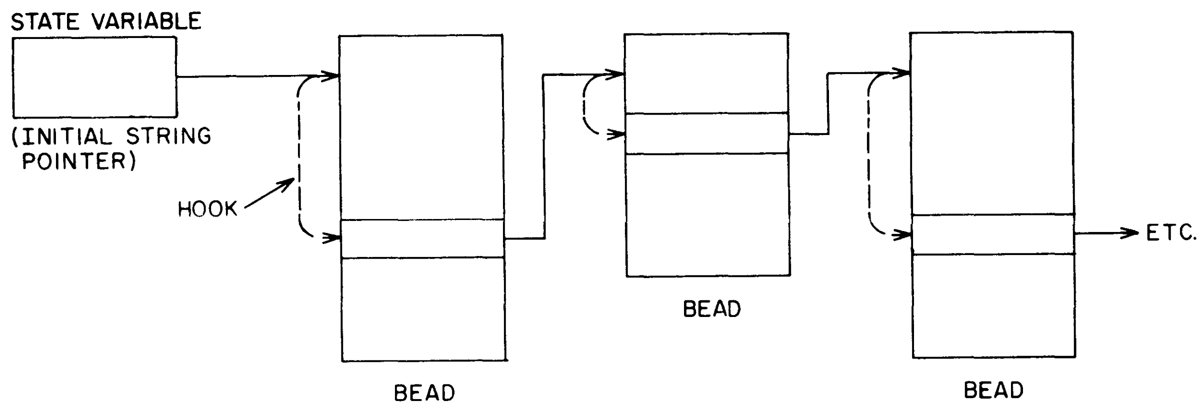


Figure 7-3. List Structure Example

BLOCK STRUCTURE AND ACCESS

Data resides in standard SCOPE random files (in logical blocks). Specification of the block length is an application programming function (see DMINIT, page 7-50).

The Data Handler maintains in-core duplicates of those blocks needed to allow efficient access to the data. The number of in-core blocks is specified by the application programmer and may be changed dynamically.

The in-core blocks reside as IFILE in the application job's global data area of the graphics control point. IFILE is rolled out and in automatically as a local file with the program.

Data is confined to beads within the blocks. Data Handler subroutines are provided to create and destroy beads, as desired by the application programmer.

The data content of a bead is broken into components. A component is a specific bit or word space within a bead and has a unique address code. Data Handler subroutines are provided to set or fetch values of components of specified beads.

The application program does not reference mass storage blocks directly, so the block accessing process and format details are not a programming function.

The Data Handler provides efficient automatic access to the mass storage blocks through an algorithmic optimization procedure. Three decision parameters, kept for each in-core block, are used in the algorithm:

- UC Usage count of the current in-core block from the time of the last decision
- ES Amount of empty space within the in-core block
- WE Indicator of in-core block content change

Three additional values are used to modify the decision parameters:

- NB Number of blocks in core
- TUC Total usage count of the Data Handler from the time of the last decision
- BS Block size

The decision process involves finding the in-core block with the minimum or maximum value of the algorithm:

$$B \times \frac{UC}{TUC} + C \times \frac{ES}{BS+NB} + D \times WE$$

The weighing factors B, C, and D are integers between 0 and 100 with a combined sum of 100. These factors are chosen by the installation. The algorithm is used in each decision to optimize use of the in-core block space (IFILE) and minimize mass storage references.

Figure 7-4 shows the structure of a Data Handler file block as it is stored in central memory.

The first word of the duplicate block contains the amount of empty space in the block and a pointer to the empty space. When a bead is entered into a block, it is associated with a bead pointer to which the bead address is related. The bead pointer is fixed in a block but its contents can vary, since beads are floating in the block. The beads in a block move when a bead is deleted in a block and the Data Handler closes up any space previously occupied by a bead to maintain empty space contiguity. The bead address IBEAD, however, remains inviolate for the life of a bead.

Beads are entered into a block starting from the bottom of the empty space. If a bead is too large to fit in a single block, it is continued onto as many other blocks as necessary. (The continuation process is designed to minimize the number of blocks per bead.) A continuation bead address is added to the end of the bead segment to point to the next segment of the bead. Bit F is set in the bead pointer to indicate that its segment is the first segment of a bead. Bit G indicates the continuation of a bead.

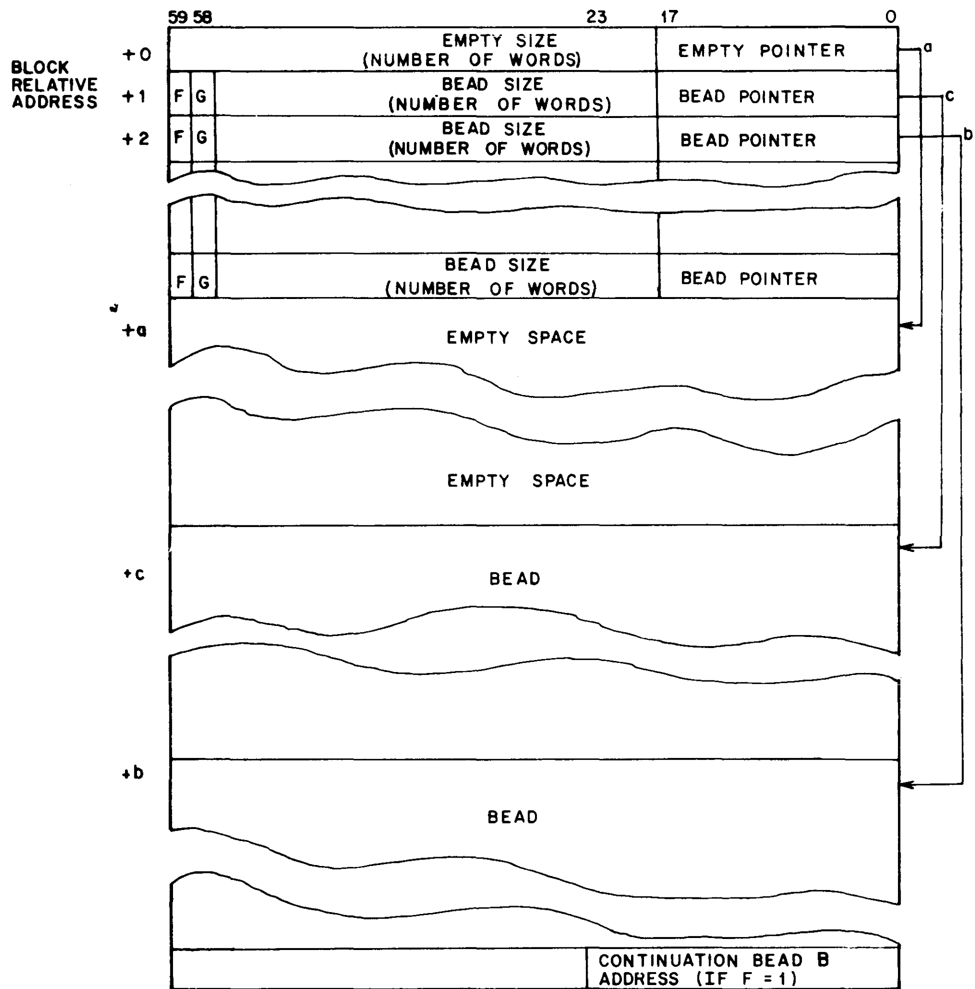


Figure 7-4. Data Handler File Block Structure

MAXIMUM DATA UNIT SIZES

The addressing scheme used by the Data Handler limits the size of files, beads, and blocks. The limits are:

- Maximum number of blocks per file 1,023
- Maximum number of beads per block 31
- Maximum number of words per bead 1,048,575 (2^{18} -1)

The number of words in a block depends on the device the system uses for mass storage. The block size can be specified by the programmer in his DMINIT calls; if the programmer omits the block size parameter from his calls, an installation parameter is used.

Another installation parameter (MAXBLKSP) defines the maximum number of words that can be allocated for the in-core blocks. Since the Data Handler requires at least two in-core blocks to function efficiently, this actually limits the maximum block size to MAXBLKSP/2 central memory words.

General Summary

a. Components
   - are bit or word spaces
   - contain values
   - reside in beads
   - are addressed by a unique code

b. Beads
   - are contiguous computer words
   - contain components
   - reside in blocks
   - are addressed by a unique bit pattern

c. Blocks
   - are mass storage logical blocks
   - contain beads
   - reside on mass storage and in core as IFILE
   - are addressed by count

d. All Data Handler routine names begin with DM

ASSOCIATIVE ADDRESSES

The Basic Graphics Package also does internal bookkeeping, controlled by bit patterns called associative addresses that are supplied to or by the application programmer. The major associative addresses are:

- The console address NCON that is associated with the particular console(s) assigned to an application program. More than one console address may be used by a program to control several consoles at once. NCON is a two digit number; the first digit is the number of the 1700 to which the console is connected (1-4), and the second digit is the number of the console itself (1-6). Thus, NCON can vary from 11 to 46. Both digits are defined by the installation and supplied by the programmer.

- The display item address IDDAD that is associated with a particular graphic item being displayed. IDDAD is used for editing functions and is the relative address of the item within a table containing the actual 1744 display addresses of all such items (see Section 4).

- The macro address MAD, which serves the same function for macro item information as IDDAD,serves for display items.

- The bead address IBEAD, which is associated with a particular set of contiguous computer words supplied by the Data Handler. The bead address is used for all references within the bead, and is defined as the relative address of the first word of the bead within the IFILE.

- The application task name NAME, is used to control program execution. A typical program may consist of over 100 individual tasks or overlays, each performing a function(s) or a computation(s). Each task resides in mass storage and is randomly accessible. NAME is used by the Application Executive to associate the task with its actual location in mass storage (see Task Directory, Section 2).

PROGRAMMING CONVENTIONS

To reduce application programming errors, the following calling sequence conventions are imposed on all Basic Graphics Package routines:

- All externally supplied values are passed between the routines as parameters in the calling statements. No specific COMMON configurations are imposed on the applications programmer.

- Needed values are specified as separate calling statement parameters. The code inefficency of loading and unloading formatted arrays justifies the use of the longer calling sequences that are produced by this convention.

- Separate subroutines are provided for each function of the Package. Code parameters are not used for function selection.

SUMMARY OF USER FORTRAN CALLABLE ROUTINES

These routines are all part of the 6000 Basic Graphics Package; all perform parameter checking functions and may cause the system software to abort an application program if its parameters are illegal. If a display item buffer exceeds the maximum length of the EXPORT/IMPORT input or output buffers (320 12-bit bytes each), it is considered a fatal error. Diagnostic messages for these and other errors are given in Appendix B.

All of the Package routines can be accessed through standard FORTRAN CALL statements. Unless otherwise specified, all parameters in the statements are passed to the routines as programmer-supplied arguments; integers may be either decimal or Boolean octal in form. The programmer may choose his own parameter names, although use of the names supplied in this manual would eliminate confusion when interpreting the diagnostic messages listed in Appendix B. Because many of these diagnostics contain the parameter names used in this manual, all parameter names throughout the book have been capitalized — a convention normally used to indicate words or letters whose presence is required by the system.

PROGRAM INITIATION

SCHEDR rolls the program out to mass storage so that it can undergo real-time scheduling and be rolled into a graphics control point for execution. A call to this routine must precede all Graphics Interface calls if the Application Executive MAIN program is not used. When the SCHEDR call is made, the system Scheduler program rolls out the entire control point and all associated files.

When MAIN is used, a call to SCHEDR serves no useful function.

Call Statement Format:

```
CALL SCHEDR
```

PROGRAM CONSOLE CONTROL

The subroutines GICNJB and GICNRL flag the 1700 Basic Graphics Package interrupt processor to establish or break the correspondence between a console and the calling job. The two routines also perform such housekeeping duties as clearing the 1744 display buffer and resetting interrupt tables.

Good programming practice dictates that a call to the console release subroutine GICNRL be made before terminating the program. However, it is not mandatory to do so since a call to GICNJB from a later job (in the time sequence of job runs) will perform the same function.

The functions performed by GICNJB and GICNRL are console-oriented. Any task of any job may request initialization of the console/job correspondence for a particular console number. More than one console may be initialized for a job.

Once console/job correspondence is made, any task of that job may address that console. If a task addresses a console that has not been initialized through GICNJB for the job of that task (or if a task addresses a console that has been initialized for some other job), the task and its job will be aborted.

A console may be in one of three states with respect to a particular job:

1. Not attached to any job
2. Attached to some other job
3. Attached to a particular job

The purpose of GICNJB is to go from state 1 to state 3. The purpose of GICNRL is to go from state 3 to state 1.

GICNJB

This subroutine assigns a programmer-specified graphics console to the calling program and performs such initial clean-up duties as clearing the display buffer.

GICNJB aborts the calling job if the console number, NCON, is invalid, or if the console is not available (i.e., has been declared out of service by the 1700 Computer operator or is assigned to another job). GICNJB clears the tables and masks that have been set.

Call Statement Format:

```
CALL GICNJB (NCON)

     NCON     Number of the graphics console that should be assigned to
              this job; only one console can be assigned through each call
```

NCON can easily be changed by loading data cards with the application program through either the remote or local card reader.

GICNRL

This routine releases the specified graphics console from the control of the calling job. GICNRL terminates internal display for console NCON and clears console-oriented tables kept by the 1700 Basic Graphics Package interrupt processor.

Call Statement Format:

```
CALL GICNRL (NCON)

     NCON     Console number; the same constraints apply here as to
              NCON in the GICNJB statement
```

PROGRAM TASK CONTROL

AETSKC and AETSKR establish the linkage between the Application Executive MAIN program and/or individual tasks of the application job.

AETSKC

This routine can be called from the zero-level overlay, any task overlay, or from any subroutine within an overlay. A call to AETSKC causes the named task overlay to be loaded into core memory from the graphics task COMMON file. AETSKC then turns control of the 6000 Series computer over to the new task; there is no return from a call to AETSKC.

Call Statement Format:

```
CALL AETSKC (NAME)
```

NAME — Name of the task to be called; this is the 1 to 7 character identifier on the PROGRAM card at the beginning of each task overlay. The name in this call must be written in 6000 internal display code, left-justified within NAME, and blank or zero-filled.

AETSKR

An AETSKR call terminates execution of the current task, then performs the functions of AETSKC for the task overlay named in the IDWA and IDWB parameters of the next button pick type ID block in the 1700 FETCH queue.

AETSKR determines which task to load by requesting that a button pick type ID block be fetched from the 1700. If no button ID block is queued there, AETSKR waits until one is entered, then loads and executes the task indicated by the button picked. There is no return from a call to AETSKR.

If a STOP or END card is encountered within a task before a call to AETSKR or AETSKC occurs, the card will cause normal termination of the entire application job.

There is no console argument in the AETSKR calling sequence. AETSKR asks for a button from the graphics console number used as the argument of the last call to a GIBUT or GICNJB routine.

Call Statement Format:

```
CALL AETSKR
```

SPECIAL ID BLOCK ASSIGNMENT

ID blocks similar to those described in Section 6 can be assigned to various input devices at each console. These special ID blocks give the devices queuing and input significance that they would not otherwise possess.

One such block may be assigned to a console for each of the following:

- All of the buttons on the function keyboard
- The switch on the light-pen

- A specific alphanumeric character, which will be used to terminate the console's current alphanumeric input

- One display item that is not defined as a light-button, but is to be treated as one

GIKYBD

GIKYBD associates an ID block similar to that of Figure 6-1 with the function keyboard of a particular graphics console. This block provides a means to examine the status of the keyboard's keys or to call a task overlay when a key is pressed.

Once GIKYBD has been called, a copy of the keyboard ID block is queued every time a keyboard key is pressed. Queueing is done according to the IDDT of the block.

Key status is contained in the IH and IV parameters of the block (see Table 4-1).

A GIKYBD call can also be used to change the ID parameters of an existing keyboard ID block.

GIKYBD cannot be used for a graphics console that is not equipped with a function keyboard.

Call Statement Format:

```
CALL GIKYBD (NCON,IDDT,IDDC,IDWA,IDWB)
```

	NCON	Number of the console to which the block should be assigned; only one console can be referenced by each call
	IDDT	ID type code; used to specify how the queue handler will treat the ID block
	IDDC	ID code word; the contents assigned by the programmer can be $0 \leq \text{IDDC} \leq 2^8-1$
Refer to 7-41	IDWA	ID information word A; contents are arbitrary unless block is referenced by an Application Executive routine
	IDWB	ID information word B; contents are arbitrary unless block is referenced by an Application Executive routine

The ID block assigned to console NCON by GIKYBD contains the representation of the input parameters IDDT through IDWB.

Only one keyboard ID block can be associated with a particular console; if several calls are made to GIKYBD with the same NCON value, the parameters of the latest call will replace all of the parameters previously entered in the block.

GILPKY

GILPKY assigns an ID block to the switch on the light-pen of a particular graphics console. If GILPKY has been called, the effect of releasing the key on the pen is identical to the act of pointing to an item on the display; the ID block assigned to the key is processed by the queue handler as if it were the ID block of a display item. This allows the programmer to detect the use of the switch.

Call Statement Format:

```
CALL GILPKY (NCON,IDDT,IDDC,IDWA,IDWB)
```

NCON	Number of the console to which the block should be assigned; only one console can be referenced with each call
IDDT	ID type code; used to specify how the queue handler will treat the ID block
IDDC	ID code word; the contents assigned by the programmer can be $0 \leq \text{IDDC} \leq 2^8-1$
IDWA	ID information word A; contents are arbitrary unless the block is referenced by an Application Executive routine
IDWB	ID information word B; contents are arbitrary unless the block is referenced by an Application Executive routine

The ID block generated by a GILPKY call contains the representation of the input parameters IDDT through IDWB. If NCON is the only nonzero input parameter (or the only parameter given in the call), the existing ID block for the light-pen switch of console NCON will be removed from the 1700 Computer's memory.

When the light-pen key is released, the copy of the ID block queued in the 1700 will contain the current H-V coordinates of the tracking cross in the IH and IV words — which are used for the coordinates of a light-pen pick in the ID block of a display item.

Only one light-pen key ID block can be associated with a particular console; if several calls are made to GILPKY with the same NCON value, the parameters of the latest call will replace all of the parameters previously entered in that block.

GIEOM

This routine assigns an ID block to a single alphanumeric character at a specified graphics console; the character may be part of the display font or on the alphanumeric keyboard. When the character associated with a GIEOM call is pressed (or picked, in the case of the display font) during an alphanumeric input operation, the ID block assigned to it is queued as if it were the ID block of a display item. This gives the programmer a means to detect an End-of-Message condition.

An End-of-Message character is displayed on the screen and returned through a GIANE call like any other character.

Call Statement Format:

```
CALL GIEOM (NCON,IBCD,IDDT,IDDC,IDWA,IDWB)
```

NCON	Number of the console to which the ID block should be assigned; only one console can be referenced with each call
IBCD	A right justified display code character which is to act as an End-of-Message indicator and to which the ID block should be assigned
IDDT	ID type code; used to specify how the queue handler will treat the ID block
IDDC	ID code word; the contents assigned by the programmer are $0 \leq \text{IDDC} \leq 2^8-1$
IDWA	ID information word A; contents are arbitrary unless the ID block is referenced by an Application Executive routine
IDWB	ID information word B; contents are arbitrary unless the ID block is referenced by an Application Executive routine

The ID block created by a GIEOM call contains the representation of the input parameters IDDT through IDWB.

Only one End-of-Message character ID block can be associated with a particular console; if several calls are made to GIEOM with the same NCON value, the parameters of the latest call will replace all of the parameters previously entered in the block.

GIPBUT

GIPBUT will create an ID block and a Queue Handler mask for a prime button at a particular console. Anything for which an ID block exists may be defined as a prime button, but the prime button ID information and its associated mask are usually used to allow a display item to activate a task when picked, even if the item is not defined as a button pick type.

When an item is defined as a prime button, two ID blocks for it exist in memory, and both are queued according to their type code values when the item is picked. This allows the programmer to simultaneously classify the item as two different types.

Call Statement Format:

```
CALL GIPBUT (NCON,IIDDT,IDDT,IDDC,IDWA,IDWB)
```

NCON	Number of console to which the block should be assigned; only one console can be referenced with each call
IIDDT	Value to be used as a mask to determine if an item is a prime button type
IDDT	ID type code; used to specify how the queue handler will treat the prime button ID block
IDDC	ID code word; the contents assigned by the programmer are $0 \leq IDDC \leq 2^8-1$
IDWA	ID information word A; if the IDDT of this ID block classifies it as a button pick type, this parameter should contain a portion of the name of the task overlay to be called by MAIN
IDWB	ID information word B; contents are arbitrary unless the ID block is referenced by an Application Executive routine

Only one prime button ID block can be created for a given console; if several calls to GIPBUT occur with the same NCON value, the parameters from the latest call will replace all of the parameters previously entered in the block. If NCON is the only nonzero parameter used, the existing prime button ID block for console NCON will be removed from the 1700 Computer's memory.

The Prime Button mask is used in the following manner (the other queue handler processing masks are explained in the paragraphs on GIMASK, below).

When a string pick or single pick entry is made at console NCON, the queue handler processing mask comparisons are made. If the pick is not a button type and is not ignored, then the following comparison is performed by the 1700 interrupt processor:

$$IIDDT \not\vee IDDT$$

IIDDT	Prime button mask value
IDDT	ID type code of picked item
$\not\vee$	Logical exclusive OR

If this algorithm equals zero, the picked item is considered to be a prime button. The IDDT value in the prime button ID block is then compared with the programmer-defined masks to see how the prime button ID block should be processed; the item's regular ID block is processed separately, according to its own IDDT value.

The prime button ID block IDDT value can be any one (or none) of the valid mask values; it does not have to equal the mask value established for buttons.

CONTROL OF QUEUE HANDLER AND PICK PROCESSING

When an entry is made at a console, the 1700 interrupt processor compares the IDDT value in the entry's ID block with the value that the programmer has previously placed in the Ignore mask. If the result equals zero, the entry is ignored. If the entry is not zero, the IDDT value is compared with values in the Single Pick, String Pick, Button, and Marker masks. If the IDDT values correspond to any of these mask values, the queue handler performs the appropriate function; the ID block of the entry is either placed on one of the appropriate queue strings (single pick, string pick, or button pick) or the item on the screen is blinked (marker function). If the IDDT corresponds to more than one Pick mask value, the ID block is queued according to the hierarchy: single, string, button.

The algorithm used for the mask comparisons are given here to further explain the mask concept. In the following paragraphs:

IDDT	ID type code parameter from the ID block of the entry
	Logical AND
	Logical inclusive OR
IGM	Value set in Ignore mask
SPM	Value set in Single Pick mask
STPM	Value set in String Pick mask
BM	Value set in Button mask
MM	Value set in Marker mask

MASK COMPARISONS

The comparisons are listed below in the order in which they are made by the software.

IGNORE MASK

If IGM $\wedge$ IDDT $\neq$ 0, the entry will be ignored, regardless of the contents of any other mask. For example, if IDDT also equals the value in the Marker mask, indicating that the item should be blinked when picked, the item will not be blinked.

SINGLE PICK MASK

If SPM $\wedge$ IDDT $\neq$ 0, the ID block of this entry is a single pick type; the ID block queued for the last single pick type entry is replaced by the ID block of this entry.

STRING PICK MASK

If STPM $\wedge$ IDDT $\neq$ 0, the ID block for this entry is a string pick type; the ID block for this entry is queued after the ID block queued for the last string pick type entry.

BUTTON MASK

If BM $\wedge$ IDDT $\neq$ 0, the ID block for this entry is a button pick type; the ID block for this entry and any associated tracking cross coordinates, single pick ID blocks, and string pick ID blocks are queued after the information queued for the last button entry.

MARKER MASK

If (SPM $\vee$ STPM $\vee$ BM) $\wedge$ MM $\wedge$ IDDT $\neq$ 0, the picked display item reverses blink status until its queued ID block is fetched by the application program.

GIMASK

This routine sets and clears the bits in the pick processing masks defined above. Each graphics console has its own set of masks, and the programmer establishes the values of each according to the IDDT parameter values that he wishes to use in his current application program.

Call Statement Format:

```
CALL GIMASK (NCON,IDDTC,IDDTS,IMASK)
```

NCON — Number of the graphics console for which the mask values will be used; only one console can be referenced through each call

IDDTC — Value of the bit pattern to be cleared from the specified pick processing masks

IDDTS — Value of the bit pattern to be set in the specified pick processing masks

IMASK — Mask indicator code; may be any one or any combination of the following:

- = 1, set or clear the indicated bits in the Ignore mask
- = 2, set or clear the indicated bits in the Single Pick mask
- = 4, set or clear the indicated bits in the String Pick mask
- = 8, set or clear the indicated bits in the Button mask
- = 16, set or clear the indicated bits in the Marker mask

Several masks can be cleared or set simultaneously by placing the appropriate values in IDDTC and IDDTS, as in the following illustration. The IMASK value used is 24_8, IDDTC is 22_8, and IDDTS is 104_8.

The f bit in ICODE of GURSET, GICOPY and GIMOVE controls the original blinking status of the item. If f is set to 1 (ICODE = s0001bb), the item will blink; if f is not set (ICODE = s0000bb) the item will not blink. However, the original blinking status will be reversed (i.e., a blinking item will stop blinking, a nonblinking item will blink) if the item is queued, provided that the marker mask is set for the item. As soon as the item is fetched, it will resume the original blinking status.

IDDTC	0	0	0	1	0	0	1	0
IDDTS	0	1	0	0	0	1	0	0

Masks Before Call

IMASK									
0	1	0	0	0	0	0	0	0	Ignore Mask
0	0	1	0	0	1	0	1	0	Single Pick Mask
1	0	0	0	1	0	0	1	0	String Pick Mask
0	0	0	1	0	0	0	0	1	Button Mask
1	0	0	0	1	0	0	1	0	Marker Mask

Masks After Call

	1	0	0	0	0	0	0	0	Ignore Mask
	0	1	0	0	1	0	1	0	Single Pick Mask
New mask value	0	1	0	0	0	1	0	0	String Pick Mask
	0	0	1	0	0	0	0	1	Button Mask
New mask value	0	1	0	0	0	1	0	0	Marker Mask

As an example of mask operation, assume that a programmer has defined grid lines as pick type 2. Each grid line has an ID block associated with it that contains an IDDT value of 2. Every time a grid line is picked by the console operator, the programmer wants to place the ID block for that grid line in the queue of string pick blocks and blink the grid line. To do this, he would make a GIMASK call with IDDTC = 0, IDDTS = 2, and IMASK = 20. This call would set both the String Pick mask and the Marker mask equal to two.

GICLR

The GICLR routine clears all ID blocks associated with a particular graphics console from the FETCH and WAIT queues in the 1700 Computer's memory. This prevents the application program from acting upon the queued information after the programmer or console operator has decided that it is no longer needed to solve his problem.

Call Statement Format:

```
CALL GICLR (NCON)
```

NCON Number of the console that should have its queued pick information destroyed; only one console can be referenced with each call

FETCHING ID BLOCKS FROM CONSOLE ENTRIES

ID information that has been queued as a result of console operator action can be retrieved from two areas within the Interactive Graphics System. GIBUT (and AETSKR) fetches ID blocks and ID information from the FETCH queues in the 1700 Computer. AELBUT, GIFSID, and GIFID fetch ID information from the ID blocks stored in the 6000 machine by the last GIBUT or AETSKR action.

Because the ID information is queued in two separate areas, the programmer must be careful when he fetches or uses it after a call to GICLR; the GICLR call erases information from the 1700 queues only. This means that calls to GIBUT will always fetch ID information queued after that last GICLR call occurred, but calls to AELBUT, GIFID, and GIFSID may reference information queued before the last GICLR call occurred. To avoid referencing the wrong ID information, a call to GICLR should be followed by a GIBUT call with IR = 0; after this call, the other four routines can be used without causing confusion.

AELBUT

This routine returns the ID information stored in the last button pick type ID block fetched from the 1700 Computer by a GIBUT or AETSKR call. AELBUT enables the programmer to investigate the parameters of the button which caused the calling of the current task overlay.

Call Statement Format:

```
CALL AELBUT (IDDT,IDDC,IDWA,IDWB,IH,IV)
```

IDDT	ID type code; returned as a result of the call
IDDC	ID code word; returned as a result of the call
IDWA	ID information word A; returned as a result of the call
IDWB	ID information word B; returned as a result of the call
IH	H axis (horizontal) coordinate of the light-pen pick which caused the button to be queued; returned as a result of the call
IV	V axis (vertical) coordinate of the light-pen pick which caused the button to be queued; returned as a result of the call

Parameters IDDC through IV are optional.

If a keyboard key, rather than a light-button, caused the calling of the current task, IH and IV will contain the keyboard status bits (see Table 4-1).

The IDDC parameter of any button referenced by AELBUT can be used to store the NCON of the console to which the button is assigned. This would give the programmer a means of determining which NCON value he should use in subsequent GIFID or GIFSID calls; if an NCON value other than that of the last GIBUT or AETSKR call is given in a GIFID or GIFSID call, a fatal error occurs (see Appendix B).

GIBUT

This routine fetches the first sequential button pick type ID block, and all related string pick type and single pick type ID blocks, from the FETCH queue of a particular graphics console. GIBUT also returns the parameters in the button ID information to the calling task. Once a call to GIBUT has been made, the information in the button ID block can be accessed again only through an AELBUT call, because another call to GIBUT will cause the next set of queued ID blocks to be fetched from the 1700 and will write over the information stored in the 6000 Series machine. If the ID block was created by GILPKY, i.e., queued by a light-pen key interrupt, IH and IV will contain the coordinates of the tracking cross at the time of the interrupt.

Call Statement Format:

```
CALL GIBUT (IR,NCON,IDDT,IDDC,IDWA,IDWB,IH,IV)
```

IR	Code to control return; if IR: = 0, wait for a button pick type ID block to be queued = 1, return to the calling task immediately
NCON	Number of the console from which the information should be retrieved
IDDT	ID type code; returned as a result of the call
IDDC	ID code word; returned as a result of the call
IDWA	ID information word A; returned as a result of the call
IDWB	ID information word B; returned as a result of the call
IH	H axis (horizontal) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call
IV	V axis (vertical) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call

If there is no button pick type ID block queued for console NCON and the call parameter IR equals zero, the application job will be rolled out until such a block is queued. If no such block is queued but IR equals 1, IDDT is returned as a positive zero.

GIFID

GIFID fetches the ID parameters from the last single pick type ID block stored in the 6000 input buffer area by an AETSKR or GIBUT call. This is usually the ID block of the last single pick display item selected by the light-pen of the specified console. The NCON in a GIFID call must agree with the NCON of the last AETSKR or GIBUT call (see AELBUT, above).

Call Statement Format:

```
CALL GIFID (NCON,IDDT,IDDC,IDWA,IDWB,IH,IV)
```

NCON	Number of the console from which the ID block should be retrieved
IDDT	ID type code; returned as a result of the call
IDDC	ID code word; returned as a result of the call
IDWA	ID information word A; returned as a result of the call
IDWB	ID information word B; returned as a result of the call
IH	H axis (horizontal) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call
IV	V axis (vertical) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call

IH and IV contain the keyboard status bits, if a keyboard key, rather than a display item pick, caused the block to be queued. The IH and IV parameters returned are the coordinates of the position where the beam was when the interrupt occurred and are in the vicinity of the display item.

If no single pick type ID block is stored in the 6000, IDDT is returned as a positive zero; the values returned for the other parameters cannot be predicted.

A call to GIFID destroys the queued ID block, so that a second call to GIFID will return IDDT = 0.

GIFSID

GIFSID fetches the ID parameters from the last string pick type ID block stored in the program's Application Executive area by an AETSKR or GIBUT call. This is usually the ID block of the last string pick display item selected by the light-pen of the specified console.

A single GIFSID call can be used to fetch the parameters from several associated ID blocks, but the programmer must dimension the ID parameter and coordinate parameter names that he uses in his calling statement.

The NCON in a GIFSID call must agree with the NCON of the last AETSKR or GIBUT call (see AELBUT, above).

Call Statement Format:

```
CALL GIFSID (NCON,N,IDDT,IDDC,IDWA,IDWB,IH,IV)
```

NCON	Number of the graphics console from which the information should be retrieved
N	The number of string pick type ID blocks from which the programmer wishes to fetch parameters; if fewer than N blocks are queued in the 6000, N is returned equal to the number of blocks from which parameters could be returned. If N > 1, the following calling parameters must be dimensioned
IDDT	ID type code; returned as a result of the call
IDDC	ID code word; returned as a result of the call
IDWA	ID information word A; returned as a result of the call
IDWB	ID information word B; returned as a result of the call
IH	H axis (horizontal) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call
IV	V axis (vertical) coordinate of the light-pen pick which caused the block to be queued; returned as a result of the call

Only 20_8 string pick blocks at a time are queued in the 6000.

IH and IV contain the keyboard status bits, if a keyboard key, rather than a display item pick, caused the block to be queued. The IH and IV parameters returned after a display item pick indicate the position where the beam was when the interrupt occurred and is in the vicinity of the display item.

If no string pick type ID block is associated with the last button pick type ID block stored in the 6000, IDDT is returned as a positive zero; the values returned for the other parameters cannot be predicted.

Once retrieved, ID block parameters are lost, so that a second call to GIFSID will return the ID parameters from the next string pick type ID block in the 6000 queue; this is the next block in the time sequence of string pick type queue entries.

A GICNJB call cannot be made between two GIFSID calls that are intended to return values from the same string of ID blocks; such a call would cause a conflict in NCON and result in a fatal error.

CONTROL OF CONSOLE ALPHANUMERIC INPUT

No alphanumeric information can be entered into the system unless the application program first provides a place on the screen to enter it and then makes a call to the 1700 requesting it.

GIANS

This routine creates a light register on the screen so that the console operator can enter alphanumeric information. The register can contain up to 80_{10} characters at a time, and can appear anywhere on the screen.

GIANS displays a line, beginning at the screen coordinates supplied by the programmer, and extending as far across the screen as he wishes. The area immediately above this line constitutes the light register. When the console operator presses an alphanumeric keyboard key or picks a font character with the light-pen, the individual letter, symbol, or number is displayed in the register and the corresponding portion of the line disappears.

If GIANS is called again while the console operator is entering alphanumeric information, the current contents of the register are destroyed; each call to GIANS defines a new register.

Call Statement Format:

```
CALL GIANS (NCON,NC,IH,IV)
```

NCON	Number of the graphics console on which the light register should be created; only one console can be referenced through each call
NC	Maximum number of characters that will be permitted in the register (defines the line length)
IH	H axis (horizontal) coordinate of the left end of the underline
IV	V axis (vertical) coordinate of the left end of the underline

GIANE

This routine performs three functions, in the following order:

1. It stops the entry of alphanumeric information into the currently defined light register.

2. It then transfers the characters currently in the register to the calling program as an array buffer; this buffer contains 10 characters (in 6000 display code) per word. The characters are left-justified within a word and blank-fill is provided for any word not completely filled.

3. It clears all characters from the register and removes any remaining portion of the underline.

If the number of characters entered in the register is less than the maximum number specified by the NC parameter of this call, the number entered in the register will be returned as a result parameter.

Call Statement Format:

```
CALL GIANE (NCON,NC,IBUF)
```

NCON — Number of the console from which the characters should be fetched; only one console can be referenced through each call

NC — Maximum number of characters in the character buffer. If more than NC characters are entered, only NC characters are returned; if fewer than NC characters are entered, NC is returned equal to the number of characters in the character buffer.

IBUF — Array buffer of picked characters; returned as a result of the call

After GIANE has been called, the programmer must call GIANS before any further alphanumeric information can be entered.

FRAME-SCISSORING DISPLAYS

Before displaying a line or arc on the console screen, the programmer may want to assure that it lies entirely within a specific area (see Display Presentation, Section 4). He can do this by calling either the GULINE or GUARC frame-scissoring routine and then using the results of his call in subsequent calls to Display Item Generation routines. GULINE and GUARC do not display anything on the console screen or create an item description that can be displayed; this must be done by other routines.

GULINE

This routine determines the points at which a given line intersects a given frame. If the given line lies completely within the frame, the display grid coordinates of the end points of the line are returned to the application program. If the line is partially within the frame, the grid coordinates of the end points of that part of the line are returned.

GULINE also scissors out lines that are too small for the graphics console operator to discern. This microscissoring is performed on any line less than six display grid units long. The end point coordinates returned after such an operation are meaningless.

If the given line lies completely outside of the given frame, the end points returned by GULINE are meaningless.

Call Statement Format:

```
CALL GULINE (IHCEN,IVCEN,IHCOR,IVCOR,H1,V1,H2,V2,
   1 KSHOW, IH1, IV1, IH2, IV2)
```

IHCEN,IVCEN	Horizontal and vertical display grid coordinates of the center of the frame
IHCOR,IVCOR	Horizontal and vertical display grid coordinates of the upper right-hand corner of the frame
H1,V1,H2,V2	Horizontal and vertical display grid coordinates of the left and right ends (respectively) of the line that the programmer wants scissored; these should be floating-point values, not Boolean octal integers
KSHOW	Scissor flag, returned as a result of the call; if KSHOW: = 0, the given line is either completely outside the frame or has been microscissored = 1, the given line is completely within the frame = 2, the given line is partially within the frame and has been scissored
IH1,IV1,IH2,IV2	Horizontal and vertical display grid coordinates of the left and right end points (respectively) of that portion of the line within the frame; returned as a result of the call, but meaningless if KSHOW equals zero

GUARC

This subroutine determines the points at which a given arc intersects a given frame. If the given arc lies completely within the frame, the display grid coordinates of the arc's center and end points are returned to the application program. If the arc is partially within the frame, the grid coordinates of the arc's center and of the end points of those parts of the arc within the frame are returned.

GUARC also scissors out arcs that are too small for the graphics console operator to discern. This microscissoring is performed on any arc with end points less than six grid units apart. The end point values returned after such an operation are meaningless.

If the given arc lies completely outside of the given frame, the end point coordinates returned by GUARC are meaningless.

GUARC is used for both arcs and circles, since the Interactive Graphics System defines only circular arcs. If the programmer wants to frame-scissor an arc that is almost a complete circle, the end point values returned to him may represent up to five separate arc segments, as in Figure 7-5.

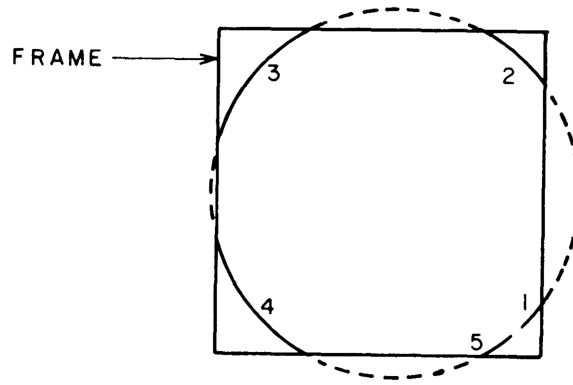


Figure 7-5. Example of a Frame-scissored Arc

Call Statement Format:

```
    1 H2, V2, KSHOW, IHC, IVC, IH1, IV1, IH2, IV2)
CALL GUARC (IHCEN, IVCEN, IHCOR, IVCOR, HC, VC, H1, V1,
```

IHCEN, IVCEN	Horizontal and vertical display grid coordinates of the center of the frame
IHCOR, IVCOR	Horizontal and vertical display grid coordinates of the upper right-hand corner of the frame
HC, VC	Horizontal and vertical display grid coordinates of the center of the circular arc that the programmer wants scissored
H1,V1,H2, V2	Horizontal and vertical display grid coordinates of the right and left ends (respectively) of the arc that the programmer wants scissored; arcs are defined counter-clockwise
KSHOW	Scissor flag, returned as a result of the call; if KSHOW: = 0, the given arc is either completely outside of the frame or has been microscissored = 1, through 5, it indicates the number of arc segments within the frame
IHC, IVC	Horizontal and vertical display grid coordinates of the center of the arc; returned as a result of the call, but meaningless if KSHOW equals zero
IH1, IV1, IH2, IV2	Horizontal and vertical display grid coordinates of the end points of those portions of the arc within the frame; returned as a result of the call, but meaningless if KSHOW equals zero

Each of the last four parameter names is the first word of an array KSHOW words in length. The coordinate value in each word corresponds to the segment of the arc that follows sequentially, counterclockwise, after the coordinate of the segment corresponding to the word before it. The first word in each array contains the coordinate of the first such segment that occurs after the initial end point specified for the programmer's original arc.

DISPLAY ITEM GENERATION

The nine routines that generate display item descriptions can be used to create a figure composed of lines, a figure composed of arcs (or any arbitrary figure combining lines and arcs), to display alphanumeric information, to define an item as a display macro, and to change the 1744 Controller's current control byte values (see Section 4).

These routines do not display anything on the graphics console screen; that can be done only by a separate GIDISP call.

All but one of the nine routines have the following three programmer-defined parameters in common:

IBUF — An array buffer used to contain description bytes produced by the Display Item Generation routines. The contents of each IBUF define one display item or display macro. IBUF must be dimensioned by the programmer; the recommended size is 64_{10} 60-bit words.

MBYTE — Maximum number of 12-bit bytes which the programmer will allow to be packed in the IBUF words.

NBYTE — Number of bytes currently in the IBUF words. NBYTE is set equal to zero by the programmer every time he starts a new IBUF, and its value is automatically corrected after each call to a generation routine.

Each call to a generation routine produces bytes of information in addition to that supplied by the programmer. These bytes are calls to the 1700 Package equivalent of the 6000 Package routine, and are the first bytes packed into IBUF by the call. Because of similar extra bytes, the IBUF used in a call to GIDISP cannot be filled so that NBYTE is greater than 312_{10} before the call; the limit on an IBUF used in a call to GIMAC is 318_{10} bytes before the call.

Under certain conditions, a non-fatal error may occur and cause a generation call to be ignored (see Appendix B); in this case, the extra bytes are not placed in IBUF.

If a generation routine is called and its actions cause NBYTE to exceed MBYTE, IBUF will only include the last MBYTE description bytes placed in it. This condition produces a non-fatal error diagnostic and the overflow bytes are lost. A statement such as:

```
IF (NBYTE.LT.MBYTE) 10, 1000
```

could be used to check for the buffer overflow condition and avoid later problems caused by a truncated IBUF.

GURSET

This routine establishes the initial conditions for a display item which is described in subsequent generation routine calls. GURSET places a reset sequence specified by the programmer in an IBUF description buffer — which is then filled with description bytes by calling other Display Item Generation routines.

The reset sequence consists of bytes that set the cathode beam intensity, item light-pen sensitivity, item blinking capability, and the display grid coordinates to which the beam should be moved (with the beam off) but it may cause flicker or blinking of the display as well as fill up the refresh buffer in the controller if many GURSETS are used.

The reset sequence is the equivalent of several 1744 command and control bytes (see Section 4).

A GURSET call should precede all other generation routine calls when a new display item description buffer is started. GURSET can also be used to place reset information in a partially filled IBUF if the programmer wishes to move the beam or change intensity in the middle of a display item; an example of these uses is given in the paragraphs concerning GUAN, below.

A display macro IBUF does not require a reset sequence.

Call Statement Format:

```
CALL GURSET (IH,IV,ICODE,IBUF,NBYTE,MBYTE)
```

IH, IV — Horizontal and vertical display grid coordinates of the point at which the cathode beam should be repositioned

ICODE — Reset control code bit pattern of the form: s00tfbb B
i.e.: 1 0 3 B

s	=	0,	disable item's sensitivity to light-pen strike
	=	1,	enable item's sensitivity to light-pen strike
f	=	0,	don't blink item when it is displayed
	=	1,	blink item when it is displayed
bb	=	01,	display item with beam at low intensity
	=	10,	display item with beam at medium intensity
	=	11,	display item with beam at high intensity
t	=	1,	terminate to next Reset
	=	0,	no termination, next Reset

IBUF	Description buffer for this display item; contains reset information as a result of this call
NBYTE	Number of bytes currently in IBUF
MBYTE	Maximum number of bytes the programmer will allow in IBUF

GUAN

This routine packs a description of alphanumeric information into a display item description buffer. A subsequent GIDISP call will display the information as lines of characters on the console screen.

Although a single GUAN call will pack up to 255_{10} characters into an IBUF, there is a smaller practical limit for a single call. Each character occupies an area on the screen that is 30_8 grid units square. This limits the maximum length of a line defined by a GUAN call to 170_{10} characters; if more than 170_{10} characters are placed in IBUF for a single line, wrap-around will occur on the display.

Because a GUAN call generates a return jump to a 1700 macro, this routine cannot be used to place alphanumeric information into a display macro IBUF.

Call Statement Format:

```
CALL GUAN (IBCD,NC,IBUF,NBYTE,MBYTE)
```

IBCD	First word of the array of characters which are to be displayed; 10 per array word in left-justified 6000 internal display code
NC	Number of characters from IBCD that should be packed by this call; if NC > 255_{10}, the extra characters will not be packed in IBUF
IBUF	Description buffer for this display item; the packed alphanumeric data is returned as a result of the call
NBYTE	Number of bytes currently in IBUF; updated as a result of the call
MBYTE	Maximum number of bytes the programmer will permit in IBUF

Each line of alphanumeric information should be defined by a separate GUAN call, but at least seven full lines can be placed in one IBUF as a single display item. The following example illustrates this:

```
•
•
COMMON IBUF (63), IBCD (49)
```

```
      NBYTE  = 0
      MBYTE = 312
      ICODE  = 102B
      READ 10, (IBCD (N),N=1,49)
   10 FORMAT (8A10/)
      CALL GURSET (4200B, 400B, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUAN (IBCD (1), 70, IBUF, NBYTE, MBYTE)
      CALL GURSET (4200B, 350B, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUAN (IBCD (8), 70, IBUF, NBYTE, MBYTE)
          •
          •
```

Note that the lines of alphanumeric information in the above example are not 170 characters long. Because the console screen is circular, the maximum line length depends on the point of origin of the line on the screen; a 170 character line would have to originate at (or very near) IH ≐ -2047, IV = 0000. An 88-character line will fit almost anywhere on the screen.

If the programmer wishes to display a character other than those defined for the 274 Console screen (see Appendix C), he cannot use GUAN unless he changes the macro address table in the 1700 Basic Graphics Package equivalent of GUAN; each character is defined as a display macro by the latter routine.

GUSEGS

This routine generates the description of a line segment and packs it in an IBUF description buffer. GUSEGS can be used to generate the description of a single line, or the description of the first line segment in a figure; in the latter case, the parameters in the GUSEGS call can be used to give this first line segment an appearance different from that of the rest of the figure.

Although GUSEGS can be used to initialize a figure (which is generated by later calls to other routines), it does not place a reset sequence in IBUF. If IBUF does not already contain a reset sequence, a GUSEGS call must be preceded by a GURSET call; a macro buffer does not need a reset sequence.

Call Statement Format:

```
CALL GUSEGS (IH1,IV1,IH2,IV2,IBEAM,ISTYLE,IBUF,NBYTE,MBYTE)
```

IH1,IV1 — Horizontal and vertical display grid coordinates for starting point of the line segment

IH2, IV2	Horizontal and vertical display grid coordinates for the end point of the line segment
IBEAM	Beam control parameter that determines the appearance of this line segment only; if IBEAM: = 0, this segment is not displayed = 1, this segment is displayed according to ISTYLE The following values can be used when figure generation is finished; if IBEAM: = -0, turn beam off and leave off after last end point coordinates are processed = -1, turn beam on and leave on after last end point coordinates are processed
ISTYLE	Style control parameter that determines the appearance of this segment and any figure generated by subsequent GUSEG calls; the degree of solidity of the line depends on the number of set bits in this parameter, as in the following sample values: = 0, -0, or 7777B, segment is solid = 5252B, segment is dashed = 6666B, segment is broken = 7272B, segment has appearance called center line by engineers
IBUF	Description buffer for this display item; contents returned depend on the call
NBYTE	Number of bytes currently in IBUF; an updated value is returned as a result of this call
MBYTE	Maximum number of bytes the programmer will permit in IBUF

If the programmer wants to frame-scissor his figure, the IH1, IV1, IH2, IV2 parameters passed to this call should contain the values returned by a GULINE call.

GUSEGI

This routine is used to initialize a figure that is generated by later calls to GUSEG. GUSEGI does not generate the description of a line segment, as GUSEGS does, but merely determines the starting point of a figure and controls its appearance.

GUSEGI does not place a reset sequence in IBUF. If IBUF does not already contain such a sequence, a GURSET call must precede the call to GUSEGI; a macro IBUF need not contain a reset sequence.

Call Statement Format:

```
CALL GUSEGI (IH1,IV1,ISTYLE,IBUF,NBYTE,MBYTE)
```

IH1, IV1 — Horizontal and vertical display grid coordinates for the starting point of the figure

ISTYLE — Style control parameter that determines the appearance of the entire figure; the solidity of the lines in the figure depends on the number of set bits in this parameter, as in the sample values given for GUSEGS

IBUF — Description buffer for this display item; the contents returned depend on the results of this call

NBYTE — Number of bytes currently in IBUF; an updated value is returned as a result of this call

MBYTE — Maximum number of bytes that the programmer will permit in IBUF

GUSEG

Each call to GUSEG generates the description of a single line segment and packs it in an IBUF description buffer. GUSEG does not initialize a figure and must be preceded by either a GUSEGS or GUSEGI call, or else a fatal error occurs.

The appearance of a figure generated by calls to GUSEG depends on the ISTYLE value used in the initial GUSEGS or GUSEGI call, and on the beam control code of each GUSEG call. The last point specified in a preceding call to GUSEGS, GUSEGI, or GUSEG is used as the starting point for the line segment generated by the current GUSEG call.

Call Statement Format:

```
CALL GUSEG (IH, IV, IBEAM)
```

IH, IV — Horizontal and vertical display grid coordinates for end point of this segment

IBEAM — Beam control parameter that determines the appearance of this line segment only; if IBEAM:

- = 0, this segment is not displayed
- = 1, this segment is displayed according to ISTYLE

The following values can be used when figure generation is finished; if IBEAM:

- = -0, turn beam off and leave off after last end point coordinates are processed
- = -1, turn beam on and leave on after last end point coordinates are processed

The IBUF array and MBYTE parameter used by a GUSEG call are the ones specified in the last GUSEGS or GUSEGI call; each GUSEG call also automatically updates the last NBYTE value.

GUSEGA

In contrast to the GUSEG routine, which must be used in conjunction with GUSEGS or GUSEGI, the GUSEGA routine performs its own initialization and then generates the description of an entire figure. One GUSEGA call can thus be used to replace many GUSEG calls if none of the parameters defining the figure depend on a console operator's actions.

GUSEGA does not place a reset sequence in the IBUF description buffer. If IBUF does not already contain such a sequence, a GURSET call must precede the call to GUSEGA; a macro IBUF need not contain a reset sequence.

Call Statement Format:

```
CALL GUSEGA (IH,IV,IBEAM,N,ISTYLE,IBUF,NBYTE,MBYTE)
```

IH, IV — First words of arrays containing the horizontal and vertical (respectively) display grid coordinates for the end point of each figure segment; this routine uses the end point of the last segment as the starting point of the next, so each segment after the first requires only one pair of coordinates

IBEAM — First word of an array containing the beam control code for each figure segment; if an array word:

- = 0, the segment is not displayed
- = 1, this segment is displayed according to ISTYLE

  The following values can be used when figure generation is finished; if IBEAM:

- = -0, turn beam off and leave off after last end point coordinates are processed
- = -1, turn beam on and leave on after last end point coordinates are processed

N — Number of figure segments to be generated by the current call

ISTYLE — Style control parameter that determines the appearance of the entire figure; the solidity of the lines in the figure depends on the number of set bits in this parameter, as in the sample values given for GUSEGS

IBUF — Description buffer for this display item; the contents returned depend on the results of the call

NBYTE — Number of bytes currently in IBUF; an updated value is returned as a result of the call

MBYTE — Maximum number of bytes that the programmer will permit in IBUF

N should always be one less than the number of values in the IH and IV arrays, because the first two words in IH and IV define only one line segment; i.e., IH(1), IV(1) is the starting point and IH(2), IV(2) is the end point of the first segment in the figure. For the same

reason, $IBEAM_i$ always describes the segment defined by IH_{i+1}, IV_{i+1}; IBEAM (N+1) therefore does not describe a segment, but can be set equal to minus zero to turn the cathode beam off when the figure is completed.

GUARCG

This routine generates a description of several arcs or a circle and packs the information in an IBUF description buffer. GUARCG can define up to five separate or connected circular arcs, deployed counterclockwise around a common center.

If the programmer wishes to frame-scissor a circular figure, the array of end points used in the GUARCG call should be the same as the array produced by a previous call to GUARC.

GUARCG does not place a reset sequence in IBUF. If the description buffer does not already contain such a sequence, a call to GURSET should precede the GUARCG call; a macro IBUF need not contain a reset sequence.

Call Statement Format:

```
CALL GUARCG (KSHOW, IHC, IVC, IH1, IV1, IH2, IV2, ISTYLE, IBUF, NBYTE, MBYTE)
```

KSHOW	Number of arc segments to be generated by this call; must be less than 6
IHC, IVC	Horizontal and vertical display grid coordinates for the common center of the arcs
IH1, IV1	First words of arrays containing the horizontal and vertical display grid coordinates for the starting point of each arc segment
IH2, IV2	First words of arrays containing the horizontal and vertical display grid coordinates for the end point of each arc segment
ISTYLE	Style control parameter that determines the appearance of all the arc segments; the solidity of the lines depends on the number of bits set in the parameter, as in the sample values given for GUSEGS
IBUF	Description buffer for this display item; the contents returned depend on the call
NBYTE	Number of bytes currently in IBUF; an updated value is returned as a result of the call
NBYTE	Maximum number of bytes that the programmer will allow in IBUF

GUBYTE

This routine is a general purpose one. GUBYTE is used to place information into an IBUF description buffer when the information is a type other than that processed by the regular Display Item Generation routines.

The information packed by GUBYTE should consist solely of the command and control bytes described in Section 4.

GUBYTE transfers the lowest 12 bits from each word in an input array to the specified IBUF. Each 12-bit byte is left-justified next to the last byte entered in the buffer, so that IBUF is packed with five bytes in each of its words (this is true of the buffers produced by all of the Display Item Generation routines).

Call Statement Format:

```
CALL GUBYTE (IBYTE, L, IBUF, NBYTE, MBYTE)
```

IBYTE	First word of the array containing one description byte at the lower end of each word
L	Number of consecutive words in IBYTE from which bytes are to be transferred
IBUF	Description buffer for this display item; contents returned depend on the call
NBYTE	Number of bytes currently in IBUF; an updated value is returned as a result of the call
MBYTE	Maximum number of bytes which the programmer will allow in IBUF

GUMACG

This routine places a macro call description into an IBUF description buffer. This allows a display item to use display macros that were previously defined by calls to GIMAC. Each call to GIMAC sends an IBUF to the 1700, where its contents are translated, converted into a display byte stream, and stored in the memory of the 1744 Controller (see Section 4). GIMAC then returns an associative address for that macro to the calling program. The macro is not displayed until a GUMACG call and a subsequent GIDISP call place a calling sequence for it into the display byte stream of a regular display item. This is done by inserting the sequence into the IBUF which describes the regular display item of which the macro is to be a part.

Call Statement Format:

```
CALL GUMACG (MAD1, L, IBUF, NBYTE, MBYTE)
```

MAD1	First word of an array containing macro address MAD parameters returned by previous calls to GIMAC
L	Number of consecutive MAD parameters from MAD1 that are to be placed in IBUF by this call
IBUF	Description buffer for this display item; the contents returned depend on the call
NBYTE	Number of bytes currently in IBUF; an updated value is returned as a result of this call
MBYTE	Maximum number of bytes that the programmer will allow to be placed in IBUF

STORING AND DISPLAYING ITEMS

Once the description of a display item is finished, the filled IBUF is placed in the display buffer of the 1744 Controller through a GIDISP or GIMAC call; GIDISP defines the contents of IBUF as a regular display item, which is then shown on the console screen; GIMAC defines the contents as a display macro, which does not appear on the screen unless a call to it occurs in a regular display item.

After the item is placed in the display buffer, it can be:

- Duplicated on another part of the screen, with a new reset sequence and a new ID block
- Moved to another part of the screen, with a new reset sequence and a new ID block
- Erased from the screen and the display buffer

GIMAC

This routine sends the contents of an IBUF description buffer to the 1700 Computer, where its contents are translated and then converted into a display byte stream by the 1700 Package routines. The 1700 version of GIMAC stores this display byte stream as a display macro in the display buffer of the specified console's controller.

GIMAC does not display the macro on the console screen, but returns the associative address of the macro to the programmer. This address parameter is then used by GUMACG to generate a macro call in the IBUF of a regular display item. A subsequent call to GIDISP for the regular display item also displays the macro.

Note that the ID block entered into the 1700 queue,when a macro is picked,is the ID block of the regular display item which called the macro.

There is only one level of macros within the Interactive Graphics System. If an IBUF is being used for the description of a macro, it cannot contain a call to another macro; this means that GIMAC can never be called to process an IBUF that has been used for previous calls to GUAN or GUMACG.

Call Statement Format:

```
CALL GIMAC (NCON,IBUF,NBYTE,MAD)
```

NCON	Number of the console to which the macro should be sent; only one console can be referenced through each call
IBUF	Description buffer for this macro; contents returned depend on the call
NBYTE	Number of bytes currently in IBUF
MAD	Display buffer associative address of the new macro; returned as a result of the call

GIMACE

GIMACE removes one or more macros from a console controller's display buffer, and frees that area of the buffer for later use.

If GIMACE is called to erase a macro that is used by one of the regular display items, the GIMACE call will have unpredictable — and probably chaotic — results on the screen. The programmer can avoid this problem by preceding a GIMACE call with a call to GIERAS; the GIERAS call erases all regular display items which use the macro that the programmer wants to erase.

Call Statement Format:

CALL GIMACE (MAD_1, MAD_2, . . . , MAD_n)

MAD_i	Display buffer address of the macro to be erased; a right parenthesis or a MAD_i equal to minus zero may be used to end the parameter list

If a MAD_i equal to positive zero occurs in the middle of the call's parameter list, the addresses following it will be ignored and their associated macros will not be erased. A zero is returned in the MAD_i parameter of each macro that has been erased.

GIDISP

This routine sends the contents of an IBUF description buffer to the 1700 Computer, where its contents are translated and then converted into a display byte stream by the 1700 Package routines. The 1700 version of GIDISP stores this display byte stream in the display buffer of the specified console's controller, and associates an ID block with it. GIDISP then returns an associative address to the calling program.

This address is the relative address of the regular display item within a table of actual display buffer addresses maintained by the 1700 Package equivalent of GIDISP; the associative address is used by the programmer for all subsequent references to the display item.

GIDISP is the only routine in the 6000 Basic Graphics Package which can display a new item on the console screen.

The IDDAD associative address parameter has the following structure:

NCON Octal number of the console that the item is assigned to

Call Statement Format:

```
CALL GIDISP (NCON,IBUF,NBYTE,IDDAD,IDDT,IDDC,IDWA,IDWB)
```

NCON	Number of the graphics console on which the item should be displayed; only one console can be referenced through each call
IBUF	Description buffer for this display item; contents returned depend on the call parameters
NBYTE	Number of bytes currently in IBUF
IDDAD	System-defined associative address of the display item; returned as a result of the call
IDDT	ID type code; used to specify how the queue handler will treat the item's ID block (see GIMASK)
IDDC	ID code word; the contents assigned by the programmer are $0 \leq IDDC \leq 2^8 - 1$
IDWA	ID information word A; contents are arbitrary unless the item ID block is used by AETSKR
IDWB	ID information word B; contents are arbitrary unless the item ID block is used by AETSKR

A standard display item identification byte stream is formed from parameters IDDT through IDWB and is added to the end of the display byte stream for the item in the 1744's display buffer; if any of the last four parameters is set equal to -0, that parameter and any subsequent ones are omitted from the identification byte stream. An item defined as a button and processed by AETSKR must have an IDWA; if none exists, AETSKR produces a diagnostic (see Appendix B) and no task is loaded.

A task name used by AETSKR must be right-justified within both the IDWA and IDWB words, but must be left-justified as a whole.

For example, the name TSK should be placed in bits 23 through 6 of word IDWA, with blank fill in bits 5 through 0. This could be done by the statement

```
IDWA = 4RTSK
```

Note that the statement

```
IDWA = 3RTSK
```

would produce an invalid task name by placing 0TSK in IDWA. AETSKR does not handle a task name that begins with a blank, so this condition would abort the job.

A longer name such as TSKNAM would have to be stored so that TSKN filled IDWA and bits 23 through 12 of IDWB contained the characters AM. This could be done by the statements

```
IDWA = 4RTSKN
IDWB = 4RAM
```

Note that the statement

```
IDWB = 2RAM
```

would produce an invalid task name by placing 00AM in IDWB; AETSKR would not recognize the resulting TSKN00AM as the task called TSKNAM.

The programmer cannot allow NBYTE to exceed 312_{10} bytes. The EXPORT graphics output buffer contains space for 320 bytes of information, and the identification bytes fill eight of them (the equivalent GIMAC bytes fill two). Since the first two bytes of every IBUF are reserved for a function code and the NBYTE value, no more than 310_{10} description bytes can be placed in the IBUF of a regular display item, and no more than 316_{10} in a macro IBUF.

GIERAS

The GIERAS routine removes one or more display byte streams from the display buffers of the consoles. This erases the display item associated with each byte stream, and also removes any of the items' ID blocks currently in the FETCH or WAIT queues — regardless of the blocks' pick types.

The programmer uses the associative addresses (produced by previous calls to GIDISP) to indicate the display items that he wants GIERAS to erase from the console screen.

Call Statement Format:

```
CALL GIERAS (IDDAD1, IDDAD2, . . . , IDDADn)
```

$IDDAD_i$ — Associative address of the display item to be erased; an $IDDAD_i$ equal to minus zero, 0, or a right parenthesis may be used to end the parameter list

If an $IDDAD_i$ equal to minus zero occurs in the middle of the call's parameter list, the addresses following it are ignored and their associated display items are not erased; if an $IDDAD_i$ equal to positive zero occurs, it is not processed but subsequent addresses are.

A zero is returned in the $IDDAD_i$ parameter of each display item that has been erased.

GICOPY

The GICOPY routine duplicates an existing display item, assigns a new ID block and a new reset sequence to the copy, and displays the copy at a new location on the console screen or on the screen of a different console. The duplication process does not affect the original display item.

Note that the reset sequence changed by a GICOPY call is the first such sequence placed in the IBUF of the original item; if the description of the original display item contains more than one reset sequence, the values assigned to the reset sequence of the copy should not be changed.

Display of the duplicate item begins at the same point within the item as it begins in the original item; i.e., if the original item was described beginning in its lower left-hand corner, then the duplicate will also begin there.

Call Statement Format:

```
CALL GICOPY (IDDADI,NCON,IH,IV,ICODE,IDDAD,IDDT,IDDC,IDWA,IDWB)
```

IDDADI — Associative address of the item which is to be duplicated

NCON — Number of the graphics console on which the duplicate display item should appear; only one console can be referenced through each call

IH, IV	Horizontal and vertical display grid coordinates for the reset sequence of the duplicate item; these are the absolute coordinates of the copy's point of origin
ICODE	Reset control code to be assigned to the copy; the s000fbb bit pattern has the same meanings as those defined for GURSET
IDDAD	Associative address assigned by the system to the duplicate display item; returned as a result of this call
IDDT	ID type code to be assigned to the ID block of the duplicate item; used to specify how the queue handler will treat the duplicate item's ID block
IDDC	ID code word for the ID block of the duplicate item; the contents assigned by the programmer are $0 \leq IDDC \leq 2^8 - 1$
IDWA	ID information word A for the ID block of the duplicate item; contents are arbitrary unless the item ID block is processed by AETSKR (see GIDISP)
IDWB	ID information word B for the ID block of the duplicate item; contents are arbitrary unless the item ID block is processed by AETSKR (see GIDISP)

Any of the call statement parameters IH through IDWB (not including IDDAD) may be set equal to -0, which will cause that parameter for the copy to be left as it is in the original display item. The parameters IDDT through DIWB may be omitted; this has the same effect as setting them equal to -0.

If there is no controller memory available for the display byte stream of the copy, a buffer overflow message will be produced at the 1700 operator's console (see Table 9-6).

GIMOVE

The GIMOVE routine can change the location, reset sequence, and/or ID block information of an existing display item. This allows the programmer to change such features of a display item as its pick type, intensity, sensitivity to light-pen pick, and whether or not it can be blinked.

Note that the reset sequence changed by a call to GIMOVE is the first such sequence placed in the item's IBUF; if the description of the item contains more than one reset sequence, IH, IV, and ICODE in the following call should be set equal to -0, since the item cannot be moved.

Call Statement Format:

```
CALL GIMOVE (IH,IV,ICODE,IDDAD,IDDT,IDDC,IDWA,IDWB)
```

IH, IV	New horizontal and vertical display grid coordinates for the reset sequence of the item; these are the absolute coordinates of the item's point of origin
ICODE	New reset control code for the item; the s000fbb bit pattern has the same meanings as those defined for GURSET
IDDAD	Associative address of the display item; not changed by the call
IDDT	New ID type code for the item's ID block; used to specify how the queue handler will treat the block
IDDC	New ID code word for the item's ID block; contents assigned by the programmer are $0 \leq \text{IDDC} \leq 2^8 - 1$
IDWA	New ID information word A for the item's ID block; contents are arbitrary unless the block is processed by AETSKR (see GIDISP)
IDWB	New ID information word B for the item's ID block; contents are arbitrary unless the block is processed by AETSKR (see GIDISP)

Any of the call statement parameters IH through IDWB (not including IDDAD) may be set equal to -0, which will cause that parameter to retain its present value. The parameters IDDT through IDWB may be omitted; this has the same effect as setting them equal to -0.

CONTROL AND USE OF THE TRACKING CROSS

Each graphics console in the Interactive Graphics System is equipped with a light-pen tracking feature called the tracking cross. This cross always exists somewhere on the console screen, but can be made invisible if the programmer wishes. The cross is a system-defined display item, described by a byte stream that is automatically placed in the display buffer of each console's controller whenever the console is initialized.

The display grid coordinates of the cross are kept in a fixed location in the display buffer (see Section 4). The 6000 Basic Graphics Package contains routines that set and fetch these coordinates, so that the cross location can be determined or changed by the programmer.

The cross and light-pen are used together in the following manner. The pen is used to pick the cross at some location on the screen. The cross is then automatically "attached" to the pen so that it moves with, or "tracks", the light-pen as the pen is moved across the screen. When the pen is stopped and the cross comes to rest, the location of the cross defines the point of a light-pen pick. If the pen and cross are moved across a display item, no light-pen pick is recorded; the cross must be motionless before a pick can be detected. This feature allows the cross to be moved across the screen without causing unwanted light-pen picks.

Two routines are also provided in the 6000 Package to attach a display item or display macro to the tracking cross. Such an item or macro is centered around the cross and moves with it across the screen until detached by another call.

GITCON

GITCON turns the tracking cross on (makes it visible) and initially locates it anywhere on the screen that the programmer wishes. The console operator can then use the cross for the tracking procedure described above.

A call to GITCON will reposition the tracking cross at the location specified in the call, even if the console operator is using it when the call is made. No repositioning will occur, however, if a button ID block is queued for the specified console; the assumption is made that a queued button will initiate some action which requires the tracking cross to be at its present coordinates.

Call Statement Format:

```
CALL GITCON (NCON,IH,IV)
```

NCON	Number of the graphics console on which the tracking cross should appear or be relocated; only one console can be referenced through each call
IH,IV	Horizontal and vertical display grid coordinates of the point at which the cross should be placed; the cross is centered around this point

The IH and IV parameters may be omitted from any call to GITCON. If IH and IV are not supplied in a call, GITCON will display the cross at the current coordinates.

GITCOF

GITCOF returns the display grid coordinates of the cross associated with the last button pick ID block retrieved from the 1700 FETCH queue. These coordinates represent the location of the cross when that button was picked; they are not necessarily the coordinates of the cross at the time of the call to GITCOF, or the coordinates of the last button picked.

Call Statement Format:

```
CALL GITCOF (NCON,IH,IV)
```

NCON	Number of the graphics console to which the call is addressed: only one console can be referenced through each call
IH,IV	Horizontal and vertical display grid coordinates of the tracking cross from the last button pick ID block fetched; returned as a result of the call

GITIMV

GITIMV attaches a previously defined display item to the tracking cross so that the item moves with the cross across the screen. The item is centered around the point defining the coordinates of the tracking cross; i.e., the cross and item have a common center point.

Call Statement Format:

```
CALL GITIMV (NCON,IDDAD)
```

NCON	Octal number of the graphics console to which this call is addressed; only one console can be referenced through each call
IDDAD	Associative address of the display item which should be attached to the tracking cross

The programmer should assure that the IDDAD value he supplies in his call is defined for console NCON; if the same display item has been created at several different consoles, it will have as many different associative addresses. Use of the wrong IDDAD value aborts the job (see Appendix B).

A call to GITIMV can also be used to detach a display item from the tracking cross. If IDDAD is set equal to zero, GITIMV will detach any item currently attached to the cross, and the item will remain at the place on the screen that it occupied when the call occurred.

GITMMV

This routine attaches a previously defined display macro to the tracking cross so that the macro moves with the cross across the screen. A call to GITMMV displays the macro and centers it around the tracking cross; i.e., the cross and the macro have a common center point.

If the macro contains a reset sequence, it will not be moved when the tracking cross is moved.

Call Statement Format:

```
CALL GITMMV (NCON, MAD)
```

NCON	Octal number of the graphics console to which this call is addressed; only one console can be referenced through each call
MAD	Associative address of the macro which should be attached to the tracking cross

A call to GITMMV can also be used to detach a display macro from the tracking cross. If MAD is set equal to zero, GITMMV will detach any macro currently attached to the cross, and the newly displayed macro will remain at the place on the screen that it occupied when the call occurred while the previous macro is erased.

The same restrictions on MAD/NCON agreement apply to this call as apply on IDDAD/NCON agreement in a call to GITIMV.

USE OF THE DATA HANDLER

Seven of the routines in the 6000 Basic Graphics Package manipulate, store, and retrieve data from files organized in a plex data structure. One or more such local files can be defined for each graphics application job. There is an installation parameter, MAXNFILE, which specifies the maximum number of files that can be used by a single job.

The programmer uses one file at a time, and does not access that one directly. Instead, he uses in-core duplicates of the record blocks within the mass storage file (a block is a fixed-length record; the programmer can specify an approximate length for each block in his DMINIT call). He specifies the number of in-core duplicates to be kept, and the Data Handler selects that number of the most frequently used blocks from the file and duplicates them in central memory. The Data Handler determines on which block space is allocated so that data can be written, and writes one in-core block into mass storage whenever it is necessary to read another into central memory so that the data it contains can be accessed. The programmer does not need to know which blocks have duplicates in central memory at any given time; for the purpose of data storage and retrieval, he can consider that the entire file always resides in central memory.

COMPONENT CODES

Data is stored within the mass storage file in word or bit spaces of variable length; these variable memory areas are called components, and make up the beads of the plex data structure. Each component within a bead is accessed according to the value of bit patterns called component codes.

All component codes begin with an octal component type number (of which there are nine), followed by the addressing needed for that component. These codes permit the programmer to use every bit in every word of each bead — a capability that FORTRAN does not ordinarily have. The codes enable him to enter bit pattern values in the bead without disturbing its other contents.

A given value can be inserted in a bead or retrieved from it in a number of ways; the method and component code used depend on the personal preference of the applications programmer and the requirements of his program. For instance, code 6 can be used to perform the functions of all the other component codes, but not necessarily in the most efficient manner. An operation such as the retrieval of the connecting bead addresses is best done with component code 10 and a call to DMGET.

The programmer should use a shifting operation to dynamically define his component code values, as opposed to the use of an expression such as

ICOMP (I) = ICOMP (I-1) *2**N

which produces unpredictable results.

The component codes and their formats are:

Type 1 code represents a 60-bit word as a bead component. The code can be written as 010000wordxxB in Boolean octal:

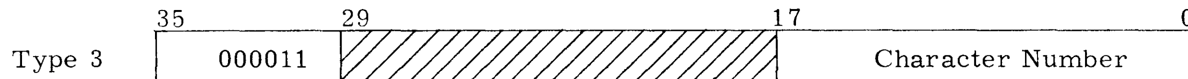
01 Component type

wordxx Position of the word within the bead, expressed as a word number; the first word in a bead is word number 1

Type 2 code represents a 120-bit double-precision floating-point value as a bead component; this value is not checked for validity as a floating-point number when it is stored or retrieved. The code can be written as 020000wordxxB in Boolean octal:

02 Component type

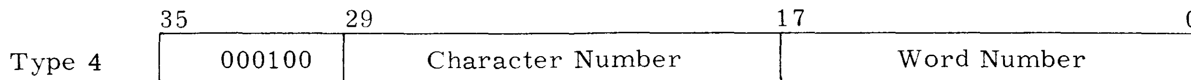
wordxx Position of the first 60-bit word of the value within the bead, expressed as a word number; the first word in a bead is word number 1



Type 3 code represents a 6-bit alphanumeric or special character as a bead component. The code can be written as 030000charxxB in Boolean octal:

03 Component type

charxx Position of the character within the bead, expressed as a character number; the first character in a bead is character number 0

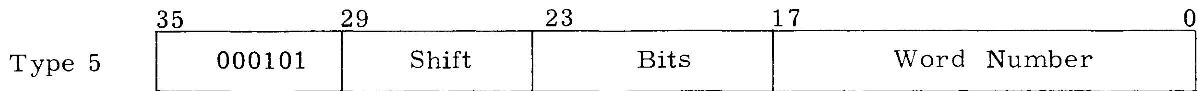


Type 4 code represents a 6-bit alphanumeric or special character within a word or word array as a bead component. The code can be written as 04charwordxxB in Boolean octal:

04 Component type

char Position of the character within the word or word array, expressed as a character number; the first character in the first array word is character number 0

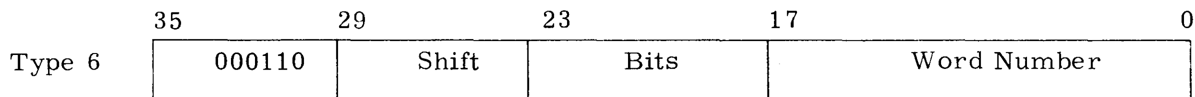
wordxx	Position of the first word of the array within the bead, expressed as a word number; the first word in any bead is word number 1



Type 5 code represents a pattern of bits within a word as a bead component. The code can be written as 05shbtwordxxB in Boolean octal:

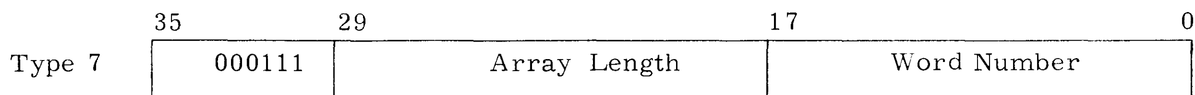
05	Component type
sh	Number of bits to shift right in order to right-justify the bit pattern within the word
bt	Number of bits in the bit pattern
wordxx	Position of the word containing the pattern within the bead, expressed as a word number; the first word in any bead is word number 1

The bit pattern stored or retrieved by the type 5 code is not sign extended; the pattern stored or retrieved by type 6 code is sign extended. In a bit pattern that is not sign extended, the left-most bit in the pattern is part of the octal value of the pattern, while in a sign extended pattern the left-most bit indicates the sign of the value represented by the rest of the pattern's bits. For example: if the bits |1|0|0|0| or the bits |0|1|0|0|0| are retrieved as patterns that are not sign extended, both groups of bits represent the value 10_8; however, if the same bits are retrieved as sign extended patterns, |1|0|0|0| represents the value -7_8 and |0|1|0|0|0| represents $+10_8$. This means that a negative octal value can be stored in its 1's complement form by using type 6 code.



Type 6 code represents a sign extended pattern of bits within a word as a bead component. The code can be written as 06shbtwordxxB in Boolean octal:

06	Component type
sh	Number of bits to shift right in order to right-justify the bit pattern within the word
bt	Number of bits in the bit pattern
wordxx	Position of the first word containing the pattern within the bead, expressed as a word number; the first word in any bead is word number 1

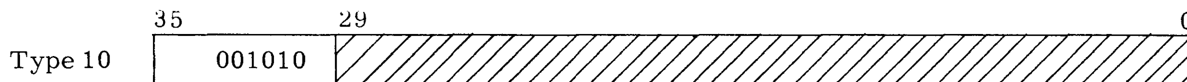


Type 7 code represents an array of 60-bit words as a bead component. The code can be written as 07arylwordxxB in Boolean octal:

07	Component type
aryl	Length of the array in words
wordxx	Position of the first word of the array within the bead, expressed as a word number; the first word in any bead is word number 1

Type 8 code represents the 18-bit address portion of a word as a bead component. The code can be written as 100000wordxxB in Boolean octal:

10	Component type
wordxx	Position of the word within the bead, expressed as a word number; the first word of any bead is word number 1



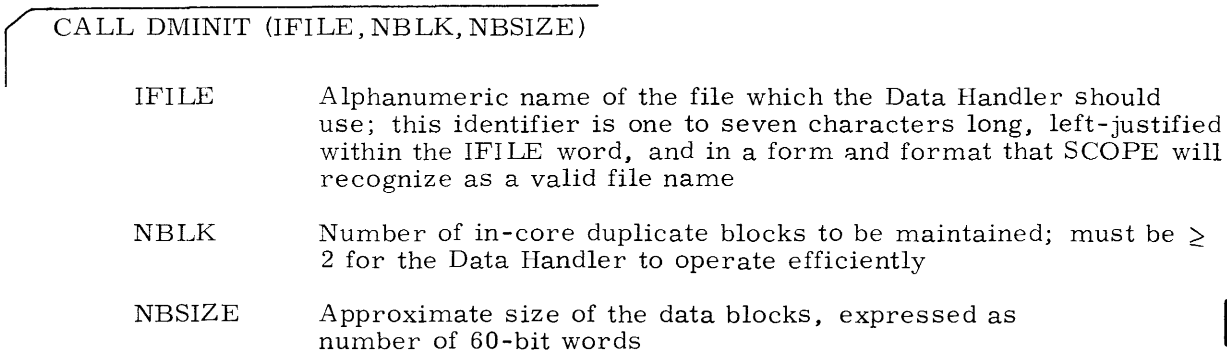
Type 10 code represents the hook (pointer) address of the next bead in a string. The code can be written as 120000000000B.

A fragment of a sample program, showing the use of these component codes, is given after the following routine descriptions.

DMINIT

This Data Handler initializing routine establishes new or changes previously defined mass storage file and core storage parameters. DMINIT is used to control the number of duplicate blocks the Data Handler maintains in central memory, to specify which file the programmer is currently using, and to establish an approximate length for each block in the file.

Call Statement Format:



```
CALL DMINIT (IFILE,NBLK,NBSIZE)
```

IFILE	Alphanumeric name of the file which the Data Handler should use; this identifier is one to seven characters long, left-justified within the IFILE word, and in a form and format that SCOPE will recognize as a valid file name
NBLK	Number of in-core duplicate blocks to be maintained; must be $\geq$ 2 for the Data Handler to operate efficiently
NBSIZE	Approximate size of the data blocks, expressed as number of 60-bit words

The NBSIZE value specified by the application programmer is rounded up to the next highest multiple of 100_8-1; this rounding up provides for the most efficient use of the 6000 system disk space and will not affect program execution. For example, if the programmer specifies NBSIZE equal to 90 (132_8), the blocks will be assigned a size of 127_{10} (177_8).

If the programmer omits NBSIZE from his DMINIT calling sequence, the Data Handler uses an installation parameter to determine block size.

NBLK should be chosen carefully. If too many duplicates are maintained, the program ties up an excessive amount of central memory with its data file; if NBLK is too small, the program's response time deteriorates because the Data Handler must access mass storage so often. The sole purpose of maintaining duplicate blocks in central memory is to avoid these problems.

The programmer can use DMINIT to switch files during program execution. If DMINIT is called with an IFILE different from the one used in a previous call, the Data Handler replaces each mass storage block in the old file with its in-core duplicate if their contents differ.* The Data Handler then uses the new file when processing all subsequent calls from the programmer.

DMINIT can also be used to change the number of duplicate blocks maintained in central memory for the current IFILE. Each call to DMINIT will change the field length of the job as necessary to accommodate any additional blocks.

Any call to DMINIT may change the field length of the job, since the in-core portion of IFILE is appended to the field length of the rest of the job (see Figure 2-11). However, if an IFILE is already open at the time of a DMINIT call, the old file's central memory area is released before space for the new one is allocated.

If the programmer dynamically changes the field length of the job after his first call to DMINIT, the change is nullified by any subsequent calls; the Data Handler always begins allocating space for its file at the same location, and always requests a change in field length just large enough to accommodate it.

DMFLSH

This routine updates the mass storage file by writing the duplicate blocks from central memory into it. *

---

*This is necessary because information entered in the file is actually written in the duplicate blocks in central memory, so that the contents of the blocks in mass storage are not up-to-date until the Data Handler writes the duplicates back into the file.

Call Statement Format:

```
CALL DMFLSH
```

No Data Handler routine can be used after a DMFLSH call, unless another call is first made to DMINIT to re-establish the file-processing parameters.

DMDMP

The DMDMP routine prints an octal dump of the entire IFILE data file. This dump enables the applications programmer to examine the data contained in the blocks and beads of the file, and is formatted for easy reference to beads and string addresses; empty spaces within the file are indicated but not shown (see Appendix D).

The dump is always placed in the standard OUTPUT file.

A call to DMDMP has no effect on the contents of the data file.

Call Statement Format:

```
CALL DMDMP
```

DMGTBD

DMGTBD allocates a specified number of contiguous words from free space in the IFILE data file, and defines those words as a bead. This provides the programmer with dynamic working storage. DMGTBD zero's out each word of the bead (i.e., each word is full of zeros before you do a DMSET).

Call Statement Format:

```
CALL DMGTBD (N,IBEAD)
```

N	Number of 60-bit words to be allocated as a bead; N must be less than 2**18
IBEAD	Relative address of this bead within the block; returned as a result of this call

If there is no space available in IFILE for a bead of N words, IBEAD is returned equal to zero.

DMRLBD

The DMRLBD routine releases the space in IFILE occupied by beads that the programmer no longer needs. This space then becomes available for the allocation of new beads.

Call Statement Format:

```
CALL DMRLBD (IBEAD1, IBEAD2, . . . , IBEADn)
```

$IBEAD_i$ — Relative bead addresses from one or more blocks, indicating the beads that should be released; an $IBEAD_i$ equal to minus zero can be used to terminate the parameter string, in addition to a right parenthesis

$IBEAD_i$ is returned equal to zero when a bead is released.

DMSET

This routine places a given value in a specified position within a bead. If the value used in the call does not occupy a full 60-bit word, the value must be right-justified within the call parameter word.

Call Statement Format:

```
CALL DMSET (ICOMP, IBEAD, VAL)
```

ICOMP — Component code specifying the position within the bead that the value should occupy; ICOMP must contain one of the nine valid type codes described above

IBEAD — Relative address of the first word of the bead in which the information is to be placed

VAL — Component value to be placed in the bead; the contents of VAL must be right-justified

DMGET

This routine retrieves a previously defined value from a specific position within a bead. If the value returned by the call does not occupy a full 60-bit word, it will be right-justified within the returned call parameter word.

Call Statement Format:

```
CALL DMGET (ICOMP, IBEAD, VAL)
```

ICOMP — Component code specifying the position within the bead that the value occupies; ICOMP must contain one of the nine valid type codes defined above

IBEAD — Relative address of the first word of the bead which contains the information

VAL — Component value returned by this call; the value returned is right-justified within VAL

A call to DMGET does not destroy the information within the bead.

EXAMPLE OF BEAD USE

Figure 7-6 illustrates a bead designed to use the nine different component codes; in several cases, more than one code is used to pack a single bead word, as in words six and nineteen.

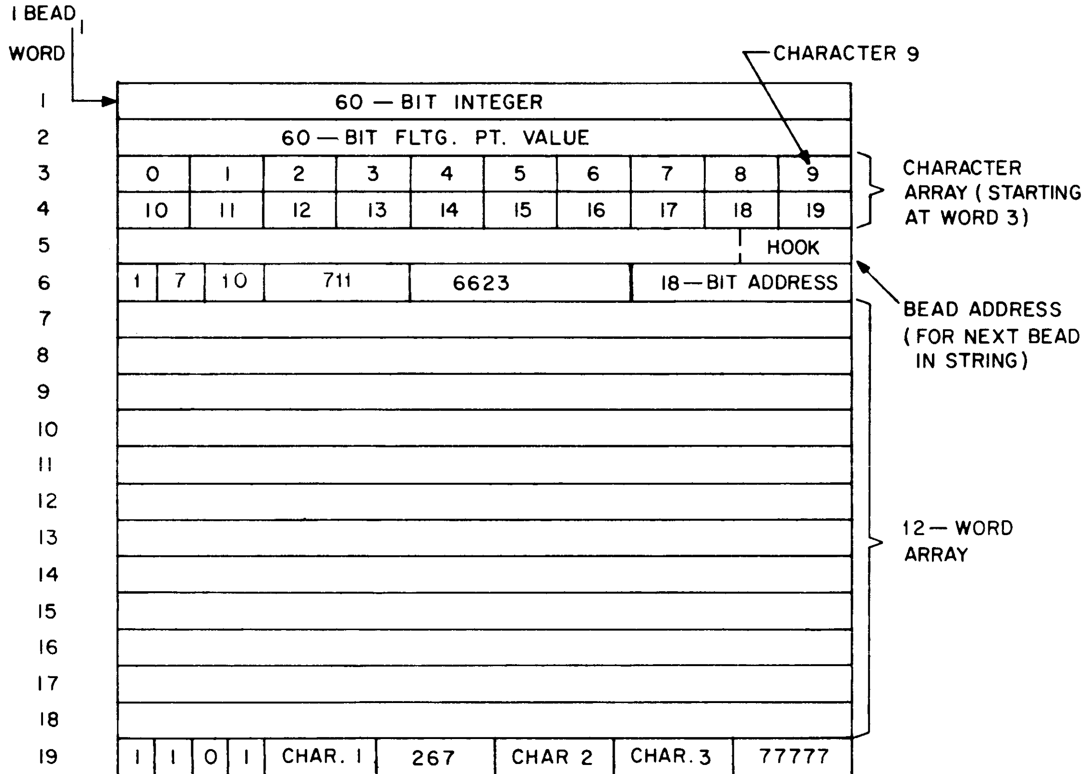


Figure 7-6. Example of Components in a Bead

The bead shown in Figure 7-6 is created and filled by the calls below.

Call	Explanation
•	
•	
•	
CALL DMINIT (7HDMFILE1, 2)	Initializes file DMFILE1, with two duplicate blocks in-core
CALL DMGTBD (19, IBEAD(1))	Establishes a bead 19 words long with a bead address returned in IBEAD(1)
•	
•	

Call	Explanation
CALL DMSET (040011000003B, IBEAD(1), 1RS)	Sets character S in character position 9 in the bead array starting at word three
•	
•	
CALL DMGTBD (50, IBEAD(2))	Establishes a second bead, 50 words long, for a bead string
•	
IHOOK = IBEAD(2) + 7B	Creates string pointer to word seven in the second bead
•	
•	
CALL DMSET (010000000005B, IBEAD(1), IHOOK)	Sets the string pointer in word 5 of the first bead
•	
•	
•	

The other calls used are not shown because of space limitations.

Word six shows six components packed into one bead word by calls using six different component codes. The components include a 1-bit value, a 3-bit value, a 5-bit value, a 9-bit value (eight bits sign extended), a 12-bit value, and an 18-bit address.

Word 19 includes four individual bit states, three alphanumeric characters, an 8-bit value and a 15-bit value. These nine components were placed by nine calls to DMSET.

These two words demonstrate the flexibility and utility of component code usage. As a further example, the character S placed by the call shown above could also have been stored by:

```
CALL DMSET (050006000003B, IBEAD(1), 23B)
```

VOLUNTARY ABORTION OF A JOB

The GIABRT routine allows the application programmer to terminate his job at any point he wishes during execution. GIABRT can be used to abort the job if a non-fatal error or another type of programming problem occurs; it can also be used to abort the job if the console user is not obtaining the results he wishes during an application run.

GIABRT displays an abort message, supplied by the programmer, on the screen of the graphics console; it then performs all of GICNRL's functions, enters the abort message in the SCOPE dayfile, and calls the standard SCOPE abort processor.

There is no return from a call to GIABRT.

Call Statement Format:

```
CALL GIABRT (NCON, IBCD, NC)
```

NCON	Number of the graphics console that should receive the abort message; only one console can be addressed
IBCD	First word of an array buffer containing the abort message
NC	Number of characters in IBCD; must be less than 47_{10}

If the application job is servicing more than one console, one should be considered a master console, so that all GIABRT messages are addressed to it.

HARDCOPY FILE CREATION

A console user often produces a display containing data for which he needs a permanent record. The GIPLOT routine is designed so that such a hardcopy record can be made. Because the type of hardcopy required varies according to the job and the equipment available, GIPLOT does not actually create a hardcopy record.

Instead, it creates a system file (called PLOT) of display information in a device-independent format. This file can then be used by a special driver to duplicate the display. The driver used depends upon the device at the installation, so that no information about it can be provided in this manual.

The following background information is needed to understand the use of GIPLOT.

An Interactive Graphics program intersperses a sequence of calls to the 6000 Basic Graphics Package routines with manipulations of data residing in the mass storage IFILE. The displays produced by the program depend upon the console user's choice of call sequences and call parameters; he chooses these variables by making task selections and data entrys from the console. The display created by one set of choices is usually modified by a subsequent set, until the user finally obtains the graphic forms and information that he wants.

When the user obtains a display for which he wants a hardcopy, he makes a console entry requesting it.

The entry should then cause the program to repeat the sequence of operations that produced the display, without repeating the intermediate steps. By repeating the sequence, the parameter string which resulted in the display is reproduced. This duplicate parameter string is then used in calls to GIPLOT, rather than GIDISP or GIMAC.

An alternative to duplicating the parameter string would be the insertion of coding, similar to the following, at the end of each task which creates a display:

```
      CALL GIBUT (0,NCON, IDDT, IDDC, IDWA, IDWB, IH, IV)
      IF (IDWA.EQ.4RPLOT) GO TO 500
      CALL AETSKR
  500 DO 501 I = 1, IDDC, IDWB
      CALL GIPLOT (NCON,IBUF(I),NBYTE(I),IDENT,ITYPE)
  501 CONTINUE
      CALL AETSKR
      END
```

This coding checks for an entry made by a light button called PLOT and returns control to MAIN if the button has not been picked. If the button has been picked, the contents of several display item buffers are sent to GIPLOT and then control is returned to MAIN and the next task. The display item buffers might be stored in labelled COMMON before each GIDISP call that creates a display item in its final form.

Call Statement Format:

```
CALL GIPLOT (NCON,IBUF,NBYTE,IDENT,ITYPE)
```

NCON	Number of the graphics console containing the display which this file should reproduce; only one console can be specified by each call
IBUF	Description buffer of the item to be entered in the file
NBYTE	Number of bytes contained in IBUF
IDENT, ITYPE	Information used by the programmer to identify himself and his file when it is later processed by the hardcopy driver

ADDITIONAL ROUTINES FOR DISPLAY FONT CREATION

Two routines have been added to the 6000 Basic Graphics Package library to facilitate the creation and use of display fonts. These routines are written in FORTRAN, using the other 6000 Package routines. One routine creates an alphanumeric font display resembling a teletypewriter keyboard, and the other creates a numeric font display resembling a clock face. The two routines actually display the fonts, but return address parameters so that the programmer can manipulate the fonts as he would a regular display item.

GFONTA

This routine creates two alphanumeric font display items, which produce the keyboard-like figure shown in Figure 7-7. The figure is created in two parts because the parameter string describing it exceeds the length of a single EXPORT buffer.

BKSP SPC CLEAR

( ) * / : $ = + - , .

0 1 2 3 4 5 6 7 8 9

Q W E R T Y U I O P

A S D F G H J K L

Z X C V B N M

Figure 7-7. Alphanumeric Display Font

BKSP is a special character for backspace ($5F_{16}$ in 1700 internal ASCII code), SPC is a special character for space (20_{16} in 1700 internal ASCII code), and CLEAR is a special character for clear ($7F_{16}$ in 1700 internal ASCII code). See Appendix C.

Call Statement Format:

```
CALL GFONTA (NCON,IH,IV,IDDA,IDDN)
```

NCON	Number of the graphics console that the font should appear on; only one console can be addressed through each call
IH,IV	Horizontal and vertical display grid coordinates of the approximate center of the display font; the font is displayed from IH - 336 to IH + 336 and from IV + 300 to IV - 200
IDDA, IDDN	First and second associative addresses of the display font items created by this call; returned as a result of the call

GFONTN

This routine creates a numeric font display item like the one shown in Figure 7-8.

BKSP SPC CLEAR

0

9 1

8 2

- . +

7 3

6 4

5

Figure 7-8. Numeric Display Font

Call Statement Format:

```
CALL GFONTN (NCON,IH,IV,IDDAD)
```

NCON	Number of the graphics console that the font should appear on; only one console can be addressed through each call
IH,IV	Horizontal and vertical display grid coordinates of the decimal point in the center of the circle; the figure is located between IH ± 244 and between IV + 310 and IV - 244
IDDAD	Associative address of the font display item; returned as a result of the call

The characters BKSP, SPC, and CLEAR have the same 1700 internal ASCII code equivalents as they have for the Alphanumeric Display Font described previously.

PROGRAMMING CONSIDERATIONS 8

This section contains hints and warnings for the application programmer.

TIME ACCOUNTING

The standard SCOPE accounting procedure is used for all jobs, including graphics jobs. Sufficient time must be requested on the Job card for each job to ensure its completion.

The hardware interrupt handlers of the 1700 Basic Graphics Package operate on a "time stealing" basis (i.e., the 6000 Series computer CPU time record is not incremented during graphics hardware interrupt handling). When graphics consoles are heavily used, the time indications in the SCOPE accounting records can be expected to lag behind clock time. Data channel use time for graphics I/O is not considered CPU time.

MEMORY ALLOTMENT AND LIST PROCESSING EFFICIENCY

The Data Handler is designed to make efficient use of the core space allotted to it by DMINIT. The algorithm used to minimize mass storage references (see Section 7) is designed on the assumption that the data structure for the application will be built and referenced as a local file.

Application programs that use a widely scattered and cross-linked data structure should allot larger amounts of core storage for data handling functions. Improper assignment of core space causes slow response at the console and excessive referencing of mass storage.

DATA HANDLER COMPONENT CODES

The Data Handler offers powerful tools for the general handling of all types of application data. The component codes required in the calls to DMSET and DMGET specify the exact location of particular pieces of data within beads. However, the bit pattern form of the component codes makes them awkward to use directly in FORTRAN programs and causes programming errors.

To avoid this problem, the convention of naming the codes through FORTRAN labeled COMMON may be used. The application programmer can lay out his bead formats and specify a name for each component code. The name can then be typed INTEGER, and assigned a particular value by using a DATA statement. The component code can be transmitted

to each subroutine handling data through use of a COMMON/DATA/statement. All DMSET and DMGET calls may then refer to component codes by name.

The use of COMMON/DATA/ to name component codes also simplifies bead format changes. The technique can be expanded to cover assignment of bead lengths and hook values.

DISPLAY ITEM ADDRESSES

The display item address parameter IDDAD is the link to all display buffer editing operations. The application program should provide disposition for the address of each item it displays. Display addresses of highly transient items, such as prompting messages, value registers, and some lightbuttons, may be kept in programmer-reserved cells in COMMON.

Most display item addresses should be an integral part of the data structure of the application and should reside in components of beads. Display items used for control or communication only, can be linked to an application data structure specifically designed for that purpose. For example, lightbuttons can be represented in a bead containing a specific identifier, class code, and display address (IDDAD):

IBEAD	POINTER TO NEXT BEAD	
IBEAD+1	CLASS	IDENTIFIER
IBEAD+2	DISPLAY ADDRESS	

Simple subroutines can then be written to:

- Display a lightbutton and splice a bead into the lightbutton string
- Erase all lightbuttons of a class and splice out their beads in a string
- Erase a specific lightbutton and splice out its bead

MACRO HANDLING

When a programmer writes display macros, he conserves display buffer space and allows more items to be displayed at one time. However, indiscriminate insertion and removal of macros can lead to an inefficient fragmentation of the fixed address area of the 1744 display buffer. Further, these functions represent the greatest operational load of the graphic interface and frequent use of them may affect response time.

The most frequently used macros should be placed at the beginning of the job coding. Transient macros should be removed immediately after use and before other transients are inserted. Groups of transient macros should be removed in the reverse order of their insertion for the fastest response time.

OPTIMUM TASK LENGTH

One of the prime considerations in programming an Interactive Graphics application is to organize the application as a series of short tasks. Interaction both implies and demands a free flow of information in two directions; from operator to application and vice versa. The operator, in his role as decision maker, should be given the ability to execute the tasks (by way of button selection) in the most meaningful way within the framework of the objectives of the application. For this reason, tasks should be concise and well defined; the operator should be able to skip quickly ahead if the interactive processes show an obvious path to the solution of the problem at hand. Similarly, the operator should be able to jump back and forth through the application when divergence occurs, until convergence to a solution is assured or it is apparent that major parametric changes are required. In either case, note that it is the operator, not the computer or the application, that makes the decisions. It then becomes obvious that the operator cannot make full use of his decision making capacity if the application programmer does not provide the operator with a means of exercising that capacity.

A job consisting of many small tasks (where many could be 300 tasks) permits SCOPE and the Application Executive to operate with maximum efficiency and provides the best task execution response.

This does not imply that one should make a job with 300 extremely short tasks if the logic of the application best suits a configuration with 50 tasks that are somewhat longer but logically more correct. What might be considered a short task on one job might be a long task on another job. The best length for any task is the length consistent with what is required to perform one phase of a job.

NON-GRAPHICS DATA HANDLER USE

The 6000 Basic Graphics Package Data Handler routines can be used by batch jobs that require a data file with a plex data structure. The programmer should bear in mind, however, that the CM parameter on the Job card does not control the allocation of central memory when the Data Handler is used; the Data Handler always appends the in-core data base to the end of the job's current field length during job execution, so that the data base would begin at the end of the memory field specified by the CM parameter.

Because the CM parameter is usually made arbitrarily large to assure enough space for both the program coding and the loader, a great deal of central memory space could be wasted when the Data Handler is used.

To eliminate the unneeded space between the regular coding and the data base, the programmer can use either a REDUCE or an RFL control card (see SCOPE Reference Manual).

Figure 8-1 shows a sample deck that uses the RFL card. In this example, the field length of the job is initially $60,000_8$ central memory words to provide space for the compiler. The field length is then changed to $30,000_8$ prior to execution; during execution, eight in-core Data Handler file blocks are created beginning at RA + $30,000_8$.

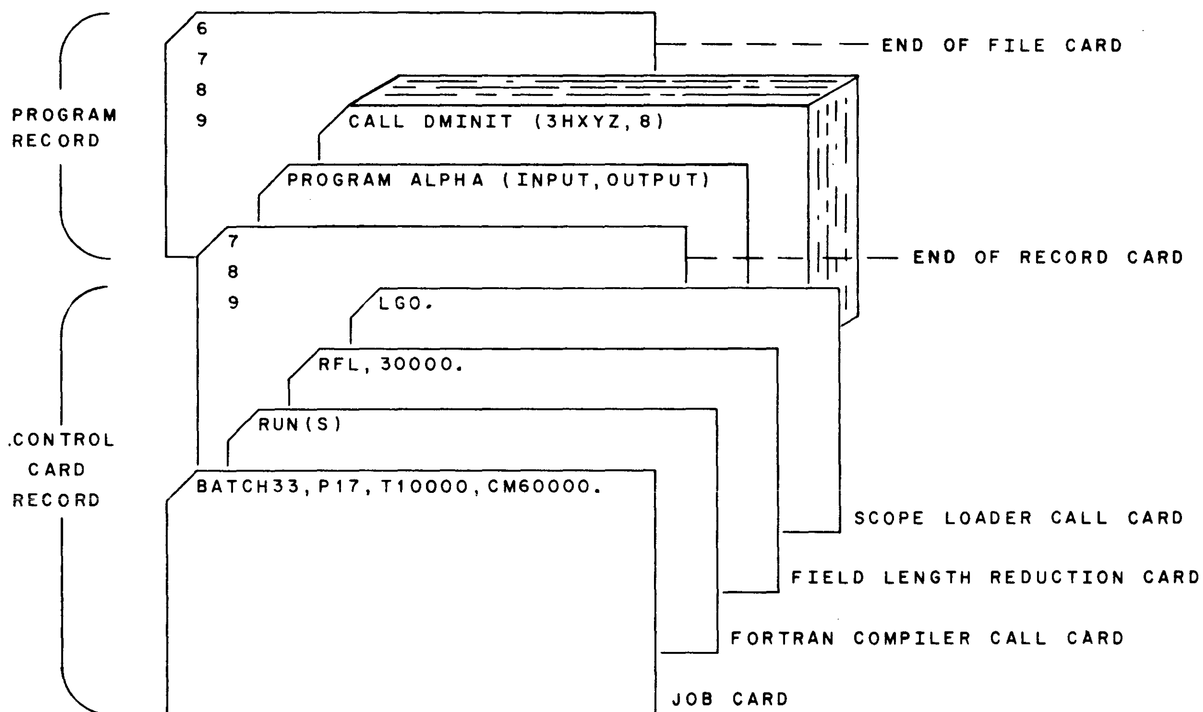


Figure 8-1. Sample Data Handler Batch Deck Using RFL

When the programmer uses an RFL card, he must be careful to leave enough space for the loader and program, yet not permit too much unused space. An easier method is shown by the sample deck in Figure 8-2, which uses a REDUCE card.

In this example, the initial field length is the same, but it is shortened before execution so that it is just large enough to accommodate the application program and the loader. The in-core data base is then appended to that field length during execution, so that almost all wasted space is eliminated.

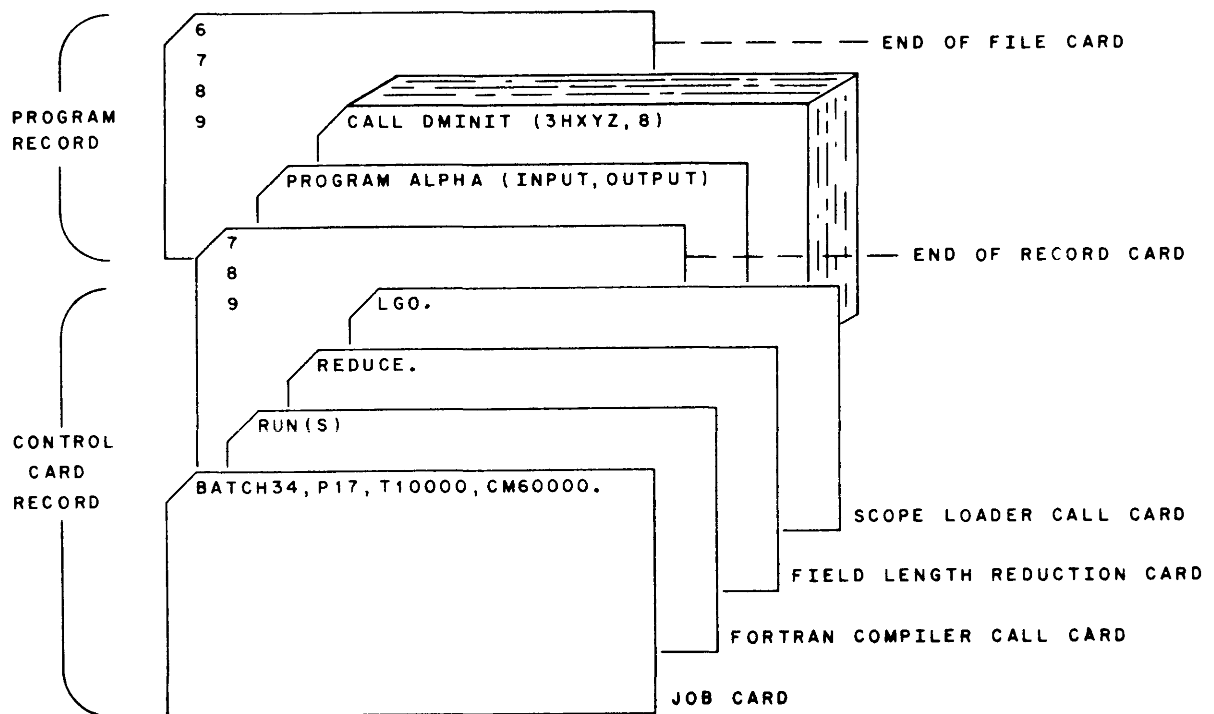


Figure 8-2. Sample Data Handler Batch Deck Using REDUCE

DATA HANDLER COMMON FILES

The files created by the Data Handler during the execution run of a job are local files and are usually destroyed when the job is finished. However, these files can be declared COMMON and subsequently used by other graphics or batch jobs.

The following steps are the suggested method for using a COMMON file as a Data Handler file. First, to create the file:

1. Open the file by calling DMINIT.
2. Before allocating space in the file for data, use DMGTBD to obtain space in the file where all bead addresses can be saved; this call should return a bead address of 41000B.
3. On each subsequent call to DMGTBD, save the bead address in the area allocated by the first DMGTBD call.
4. At the end of the job run, call DMFLSH to update the file in mass storage.
5. After the LGO card in the control card record, insert a COMMON card with the name of the Data Handler IFILE on it.

Then to use the file during a different run:

1. Insert a COMMON card, naming the proper file, in the control card record before the LGO card.
2. Open the file using a DMINIT call, and assure that the NBSIZE parameter has the same value as during the run that created the file.
3. Get the bead addresses for data from the first part of the file, using a DMGET call with a bead address equal to 41000B (see above).
4. If any new data is stored in the file during the run, call DMFLSH at the end of the job to assure that the mass storage version of the file is up-to-date.

The following programs are examples of using a Data Handler file as a COMMON file.

Sample program to create the file:

```
JOB1.
RUN(S).
LGO.
COMMON, DMFILE.
7
8
9
      PROGRAM DMTEST1 (INPUT,OUTPUT)
      COMMON IBD (500), IPTR
C     OPEN FILE DMFILE WITH 4 IN-CORE BLOCKS
      CALL DMINIT (6LDMFILE, 4)
C     GET SPACE ON FILE WHERE BEAD ADDRESSES WILL BE SAVED
      CALL DMGTBD (500, IPTR)
C     IPTR NOW EQUALS 41000B SINCE THIS IS FIRST CALL TO DMGTBD
               •
               •
               •
      CALL GETBEAD (N1, J1)
               •
               •
               •
      CALL GETBEAD (N2, J2)
               •
               •
               •
      CALL DMSET (ICOMP, IBD(J2), VAL)
               •
               •
               •
      CALL DMFLSH
      END
      SUBROUTINE GETBEAD (NUM, INDEX)
      COMMON IBD(500), IPTR
C     ALLOCATE "NUM" NUMBER OF WORDS IN DMFILE
      CALL DMGTBD (NUM, IBD(INDEX))
      IC = 10000000000B + INDEX
C     SAVE BEAD ADDRESS RETURNED IN IBD(INDEX) IN FILE DMFILE
      CALL DMSET (IC, IPTR, IBD(INDEX))
```

```
          RETURN
          END
               •
               •
               •
```

Sample program to use the file:

```
     JOB2.
     RUN(S).
     COMMON, DMFILE.
     LGO.
     7
     8
     9
          PROGRAM DMTEST2 (INPUT, OUTPUT)
          DIMENSION IBEAD (500)
     C    OPEN FILE DMFILE WITH 4 IN-CORE BLOCKS
          CALL DMINIT (6LDMFILE, 4)
               •
               •
               •
     C    SET IPTR = 41000B SO BEAD ADDRESSES CAN BE RETRIEVED FROM
     C    DMFILE
          IPTR = 41000B
          ICOMP = 070764000001B
          CALL DMGET (ICOMP, IPTR, IBEAD)
     C    ARRAY IBEAD NOW CONTAINS ALL BEAD ADDRESSES SET DURING
     C    CREATION RUN
               •
               •
               •
          CALL DMGET (ICOMPA, IBEAD(J), VAL)
               •
               •
               •
          CALL DMFLSH
          END
               •
               •
               •
```

6612 CONSOLE

CONTROL POINT ASSIGNMENT AND RELEASE

AUTOMATIC INITIAL ASSIGNMENT

The type-in command:

AUTO. (CR)

structures the 6000 Series system with BATCHIO at control point 1 and NEXT at 2 through 6. Control point 7 remains blank.

MANUAL ASSIGNMENT AND RELEASE

BATCHIO can be assigned to control point n by typing in:

n. BIO. (CR)

EXPORT is automatically loaded by BATCHIO when needed.

To dedicate a vacant control point to graphics processing, make sure EXPORT is up, then type in:

GRAPH, n, fl. (CR)

n	Control point number reserved for graphics use
fl	Octal field length - the amount of core memory to be reserved for control point n (fl is the actual field length divided by 100) Fl cannot be zero.

To dedicate two vacant control points to graphics, type in:

GRAPH, n, fln, m, flm. (CR)

n	First control point reserved for graphics use
fln	Field length (divided by 100) reserved for control point n
m	Second control point reserved for graphics use
flm	Field length (divided by 100) reserved for m

(fln and flm cannot be zero.)

This entry can also be used to change control points and/or field lengths.

To assign control points that are currently running batch jobs, type in:

n. CLEAR. (CR)

for each control point (n is the control point number); this type-in command prevents NEXT and/or another batch job from being brought to the control point when the current job terminates. When the control point becomes vacant, assign it to graphics using either of the type-ins above.

If only one control point is assigned to graphics, it can be released by typing:

GRAPH, 0, 1. (CR)

If two control points are assigned and one is to be released, the type-in:

GRAPH, n, fln, 0,1. (CR)

will retain control point n with field length fln (times 100), and release the other assigned control point.

To release both graphics control points to the system, type in:

GRAPH, 0,1,0,1. (CR)
n. DROP

The NEXT package is brought to a released control point after normal termination of job processing.

BATCHIO, B AND K DISPLAYS

CALLING THE K DISPLAY

The K display (see Figure 9-1) shows device assignments and problem messages for each of the 16 buffer areas used by the BATCHIO drivers. The word IDLE appears in the job name area of each buffer not currently assigned to a device, and the word EXPORT appears in the job name area of a buffer currently being used to service a 1700. All other job (or file) names appearing on the K display are truncated and suffixed — as they are for the dayfile (A) display. Nothing appears in the message area of a buffer when its device is operating normally.

The K display may be called to the left console screen with the type-in:

K d. (CR)

where d is the letter identifying the display that is to replace the one currently appearing on the right screen. Similarly, the K display may be placed on the right screen by:

d K. (CR)

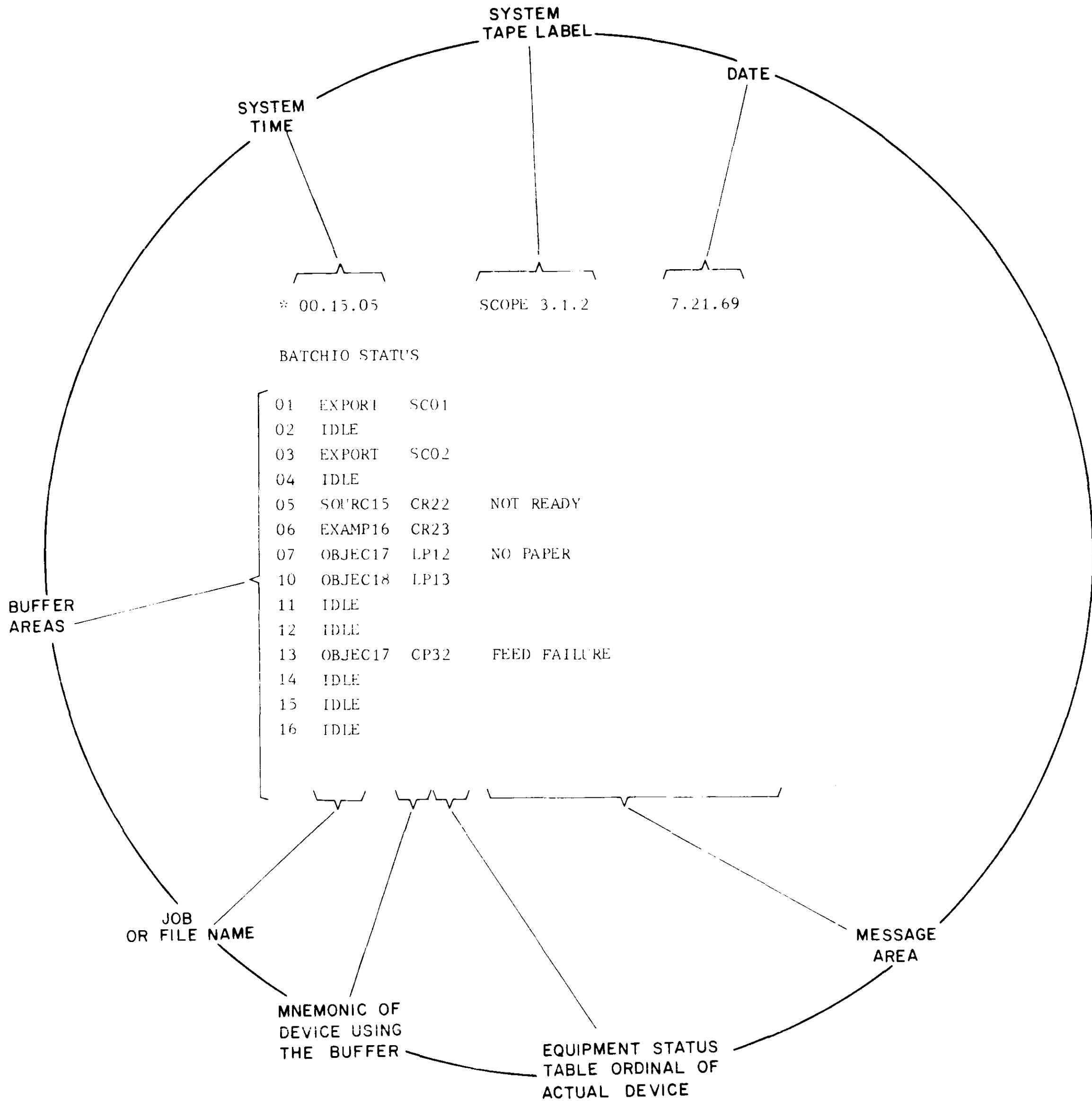


Figure 9-1. DSD 6612 K Display

K DISPLAY EQUIPMENT MNEMONICS

Table 9-1 lists the mnemonics of all the devices that can be serviced by BATCHIO.

TABLE 9-1. EQUIPMENT MNEMONICS

Mnemonic	Equipment
CP	Card punch
CR	Card reader
LP	Line printer
SC	Data Set Controller for 1700's Data Channel

K DISPLAY BUFFER MESSAGES

Table 9-2 contains all of the equipment status messages which may appear in a buffer's message area. Only messages that concern conditions requiring operator action appear on the K display.

TABLE 9-2. BUFFER MESSAGES

Message	Meaning and Action
FEED FAILURE	Card punch is either jammed or out of cards.
NO PAPER	Line printer is out of paper.
NOT READY	Something has occurred that produces a "not ready" condition in this device; check all controller and device switches.
RE-RD 1 CD. COMPARE ERROR.	The last card read must be put through again to compensate for a compare error.

OUTPUT CONTROL COMMANDS

BATCHIO output processing can be controlled by the END, REPEAT, and SUPPRESS type-in commands.

The END xx. (CR) type-in stops the printing or punching of the file at K display buffer area xx, and starts outputting the dayfile entries for that file. Using the same type-in a second time will stop dayfile output and drop the job completely.

The REP xx. (CR) type-in ends the printing or punching of the file at K display buffer area xx, and places the file back in the SCOPE OUTPUT queue so that it can be rescheduled on another device.

The SUP xx. (CR) type-in supresses the processing of all printer format control characters for the file at K display buffer area xx; the file is printed in 136-character lines, and the lines are single-spaced.

BATCHIO B DISPLAY AREA

The structure of the B display area used for the BATCHIO control point is as follows:

```
n. BATCHIO
   m BUFFERS ACTIVE
   messages
   RA aaaaaa FLwwwwww
   EXPORT A. status B. status C. status D. status
   communication messages
```

n	Control point number.
m	Number of buffers currently being used (the whole message is replaced by the word IDLE when there are no buffers in use).
messages	Last K display message, or one of the connect/reject dayfile messages mentioned in Section 2.
aaaaaa	Current octal relocation address of the control point field.
wwwwwww	Current octal length of the field.
EXPORT A.....	Display messages generated by EXPORT. The letters A, B, C, and D refer to the 1700's numbered 1, 2, 3, and 4, respectively. (See the subsection on EXPORT below.)
communication messages	Message from a remote operator identifying the sending terminal. (See the subsection on EXPORT below.)

EXPORT

INSTALLATION PARAMETERS

When the Interactive Graphics System is first configured, several installation parameters must be set to prepare EXPORT for execution; one of which is the following:

- Equipment Status Table entries for each 6673 or 6674 Data Set Controller attached to the system configuration; the correct equipment and channel numbers for each controller must be known to correctly interpret the EXPORT entries on the K display.

INITALIZATION

Once the system is configured, no operator action is necessary to initialize EXPORT. The EXPORT control point area is automatically structured by BATCHIO when that routine is loaded, and all EXPORT routines are kept in Central Memory Resident. Communication with the 1700 Computer and IMPORT is initiated by the 1700 operator.

B DISPLAY STATUS MESSAGES

The fifth line of the BATCHIO B display area (EXPORT A. status B. status C. status D. status) is used to display messages generated by EXPORT. The letters A, B, C, and D refer to the 1700's numbered 1, 2, 3, and 4, respectively. Table 9-3 contains all EXPORT messages, in alphabetical order; if a message does not pertain to a specific 1700 terminal, it is displayed on the sixth line of the BATCHIO area (communication messages).

TABLE 9-3. EXPORT MESSAGES

Message	Meaning	Operator Action
BAD CALL PARAMETER	A graphics job has issued an EXPORT service request that EXPORT cannot interpret; the job is aborted.	None
--DEAD--	The 1700 has sent a directive word which is not valid on the basis of the current EXPORT status word. In order to protect the 6000 and the other 1700's, EXPORT stops communicating with this terminal. In a debugged system, this should not occur.	To recover, idle all 1700 terminals, turn off all of EXPORT's 6673 or 6674 Controllers, wait until EXPORT drops, then turn the equipment back on.
DOWN	This 1700 was active but has ended operations.	None
ERROR	Either a loss of communication or Data Set Controller hardware error has occurred. This message is displayed if EXPORT receives no response from IMPORT after 64 consecutive retransmissions; it usually occurs if the 1700 has stopped communications without sending EXPORT a "shut down remote" directive.	Dependent on cause
IDLE	Communication exists with this 1700, but no data is currently being transmitted.	If a message for a 1700 has been entered and IDLE appears, re-enter message with the correct terminal ID (see below).
I/O	Sending or receiving data.	None
MESSAGE	A message from the operator of 1700 currently appears on the sixth line of the BATCHIO B display area; this EXPORT message identifies the sending terminal.	Acknowledge message by entering: n. GO. (CR) where n is the BATCHIO/ EXPORT control point number.

TABLE 9-3. (Cont'd)

Message	Meaning	Operator Action
NCON ERROR	A graphics program has issued an EXPORT service request containing an invalid NCON parameter; the job is aborted.	None
-PAR-x	A parity error has occurred on output data stream x; output stops for operator action.	To continue output, type in: n. GO. (CR) where n is the BATCHIO EXPORT control point number.
STORAGE	A 1700 is sending batch job data and needs central memory storage.	Free central memory storage or wait until storage is available.

NOTE

Whenever IMPORT ends operations or is declared DOWN by EXPORT, all output files in transmission for that terminal are rewound and returned to the SCOPE output queue. If EXPORT is dropped, all attached output files are rewound and returned to the output queue. (The files are still considered remote files, and are not output locally.)

INTER-COMPUTER OPERATOR COMMUNICATION

A message from a remote operator is displayed on the sixth line of the BATCHIO B display area (communication messages), with the word MESSAGE entered on the fifth line after the appropriate terminal designator. Before another remote operator message can be displayed, the 6000 operator must acknowledge the displayed message by typing in:

n. GO. (CR)

n BATCHIO/EXPORT control point number

To transmit a message to a particular 1700, the 6000 operator types:

n. * x message (CR)

n BATCHIO/EXPORT control point number

x Terminal designator (A, B, C, or D - see above)

message Any alphanumeric or special characters - up to a maximum of 27_{10}

DAYFILE/B DISPLAY MESSAGES

For DAYFILE/B DISPLAY messages concerning graphics jobs see Appendix B.

1700 COMPUTER CONSOLE

INITIALIZATION AND RESTART PROCEDURE

Because the minimum 1700 hardware configuration precludes the use of most of the 1700 operating system, this manual has made no mention of it; the sections of that system which are used will be treated in the following pages as a part of IMPORT.

TYPED-IN BOOTSTRAP LOADER

The 1700 operator must perform the following activities to prepare IMPORT for remote processing. This procedure assumes that all parameters are preset within the operating system at loading time, and that the system is to be loaded from the 853 Disk Drive (it can also be loaded from the card reader or paper tape reader).

1. Set all console switches to neutral.
2. Verify that:
   a. Previously prepared disk pack on the disk drive contains the operating system and IMPORT;
   b. Disk and controller are On and Ready;
   c. DSC is on, all test switches are Off, and the data set is plugged in;
   d. Card reader, printer(s), and teletypewriter power is On;
   e. 1713 Teletypewriter's right-hand selector switch is set in the ON LINE position and that it is in K mode.
3. Depress the Clear switch on the computer console.
4. Momentarily depress the Auto Load button on the 1738 Disk Pack Controller.
5. Momentarily set the Run-Step switch to Run. At the teletypewriter, the typeout PP appears.
6. Set the Protect switch to the Protect position.
7. Depress the Break Release button and type an asterisk (*), followed by a carriage return.
8. Depress the Manual Interrupt button on the teletypewriter; the system responds by typing MI.
9. Depress the Break Release button and type an asterisk followed by IGS and a carriage return.
   The IMPORT program then loads, clears buffer areas, sets flags, adjusts for system environment, and outputs:

   IMPORT READY. . . S, R, OR U**

   on the teletypewriter.

10. Depress the Break Release button (do not depress the Manual Interrupt) and type in one of the above options:

    S Clear 1700 job table and start IMPORT/EXPORT communications;

    R Restart IMPORT/EXPORT communication and do not clear 1700 job table;

    U IMPORT

    If the operator selects the S or R type-in, IMPORT attempts to begin communication with the 6000 Series computer.

    To initiate communications, the 1747 DSC sends an interrupt status code word to the 6000 DSC. The 1700 Computer then delays for a short time on a two-word receive with an end-of-operation interrupt selected. If the central site computer does not respond, the process is repeated. (This repetitive process can be observed by noting the on/off pattern of the overflow indicator on the 1700 console.) The central computer acknowledges the interrupt code word by transmitting two words.

11. IMPORT types IMPORT 1700 when communications are established.

12. Depress the Manual Interrupt button on the teletypewriter. The system responds by typing MI.

13. Depress the Break Release button and type GO followed by a carriage return.

The EXPORT/IMPORT system is now ready for remote processing. The teletypewriter output produced by the preceding activities should appear as follows:

```
PP
*
MI
*IGS
IMPORT READY. . . S, R, OR U ** S
IMPORT 1700
MI
GO, 11
MI
ONGR
```

COMMUNICATIONS FAILURE

If a fatal transmission error occurs and EXPORT declares the 1700 inoperative, IMPORT will output the message:

```
DSC REJECT
   or
1706 REJECT
```

Because EXPORT aborts all jobs associated with an inoperative 1700, there is no reason to attempt recovery of the 1700's contents. Therefore, the only way to re-establish communication is to reload the system through the procedure above. When the system is reloaded, all graphics console controllers are automatically initialized, and their contents lost.

CONTROL TYPE-INS

The 1700 operator can request the execution of a variety of functions through the use of the teletypewriter. Table 9-4 is a list of operator type-ins and their corresponding functions.

TABLE 9-4. IMPORT CONTROL TYPE-INS

Statement	Function
GABT, NCON = n_1, n_2, (etc.)	Abort the graphics job using console n. More than one console can be specified as shown.
STAT, job name	Obtain information on status of job.
CPR, job name, priority	Change priority of job.
CPT, job name, seconds	Change CP run-time limit
RPNT, n, lu	Rewind entire file currently being output on lu, if n is zero. If n is given, the output file is backed up n sectors and then started from that point. (n is 63_{10} sectors maximum; lu is logical unit number.)
DVT, job name (or) DVT, job name, dt	Divert all remote output for job named to central facility. If dt is given, divert only appropriate output if job is in the output stack.
TERM, lu	Terminate output on data stream specified by lu for job currently being output.
ABT, job name	Abort job named, if the job is at a control point. (This has the same action as central operator DROP command.)
DISP, message	Transmit display message for central operator.
LIST	List current contents of IMPORT job table and current status of each job.
END	Shut down remote communication.
GO (or) GO, lu (or) GO, lu, x	Initiate data transfers on all data streams or the data stream associated with lu.
STOP, lu	Stop data transfer on data stream associated with lu.
SEOJ, lu	Stop data transfer at end of the job on data stream associated with lu.

TABLE 9-4. (Cont'd)

Statement	Functions
WAIT (or) WAIT, lu	Suspend data transfer temporarily on all data streams or the data stream associated with lu. (Continue operation with the GO statement.)
RLSE	Release all remaining jobs to central site for output. The response to this statement is: xx PRINT FILES DIVERTED, and/or yy PUNCH FILES DIVERTED, and/or zz OTHER FILES DIVERTED. If there are no files to release, the response is: ------NO REMOTE FILES----. These messages are entered into the central site dayfile. (xx, yy, and zz are decimal.)

Definition of terms in Table 9-4:

job name — Name on job card with SCOPE 3.1 appended sequence number and terminal ID. (i.e., name supplied with a job acknowledgement message.)

priority — Octal number with a maximum of 4 digits.

seconds — Octal number with a maximum of 5 digits.

dt — LP or CP (line printer or card punch).

message — One line of 27_{10} characters maximum.

lu — Logical unit numbers. Decimal numbers assigned to the computer peripheral devices. The number assignments are present within the operating system, and are determined during MSOS initialization.

n — Number of graphics console (1 to 6). Also assigned during system assembly.

x — Parameter used with 430/1728 reader/punch; x = R, read; x = P, punch.

TELETYPEWRITER INPUT PROCEDURE

To enter a type-in command, the following steps must be followed:

1. Press the Manual Interrupt button (the resulting type-out is MI; a line feed carriage return is activated and the Break indicator is lit).
2. Press the Break Release button.
3. Enter the statement.
4. End the statement with a carriage return.

JOB LOCATION ON MESSAGE ACKNOWLEDGEMENT

Certain input statements are only valid when the specified job is in the input stack, at a control point, in the output stack, in the process of being output, or when the output stream is active. Depending on the input statements entered, acknowledgement is made in one of three ways:

- Acknowledgement with the job name specified
- Acknowledgement with no job name specified
- INVALID REQUEST message

Table 9-5 lists these job locations.

TABLE 9-5. JOB LOCATION

Message	Input Stack	At CP	Output Stack	Being Output	Acknowledgement
CPR	X	X	X	X	Without job name
CPT		X			Without job name
ABT		X			Without job name
TERM				X	Without job name
DVT					
blank	X	X	X	X	With job name
LP			X	X	With job name
CP			X	X	With job name
RPNT				X	Without job name

OUTPUT MESSAGES

Certain phases of remote operator command and job processing cause IMPORT or the Buffer Translator to output teletypewriter messages informing the 1700 site of job advancement and of particular error conditions. The time read from the system clock on the 6000 Series computer precedes all teletypewriter messages received at the remote site in the form: xxyy:zz (where xx=hour, yy=minutes, and zz=seconds). Table 9-6 lists all possible messages and provides a brief description of each.

TABLE 9-6. OUTPUT MESSAGES

Source	Message	Definition
EXPORT	job name	Acknowledgement from EXPORT that the indicated job is ready to be released to the system. IMPORT places this job name in its internal job table.
EXPORT	(*) job name IN STACK	Job named has been released to SCOPE by EXPORT and is waiting for a control point.
IMPORT	JOB TABLE FULL	IMPORT job table is full. Reading continues automatically as soon as a job table entry is cleared (e.g., printing complete or a job is diverted). The job table holds 25 jobs.
IMPORT	NO JOBS	Response to list command when IMPORT job table is empty.
EXPORT	(*) job name IOSxxxx	Job named has completed execution and has left the control point. It is in Output Stack. The priority of this output file is xxxx.
EXPORT	(**) job name lu C	Output has completed for job named,for logical unit numbers lu.
EXPORT	(col. 1-7) JOB CARD ERROR	Columns 1-7 contain the first seven characters of what was sent as a job card. IMPORT passes the cards being read until an EOF is reached, after which IMPORT resumes sending jobs to EXPORT.
IMPORT	CL	IMPORT has lost communication with EXPORT. IMPORT automatically attempts to re-establish communication.
IMPORT	EXIT IMPORT	IMPORT operation terminated.
EXPORT	INVALID REQUEST	EXPORT did not accept the last teletypewriter request sent. (See Table 9-3).
EXPORT	(**) job name DONE (or) (OPER....) DONE	EXPORT did process the last teletypewriter request sent by IMPORT. (See Table 9-3.)
EXPORT	(**)(*) job name NOT IN SYSTEM	Job named has completed output or never existed.
IMPORT	DSC (or) 1706 REJECT	An abnormal condition was detected in the communications hardware. IMPORT is aborted. (Check Power On and DSC switches for the proper settings.)

TABLE 9-6. (Cont'd)

Source	Message	Definition
IMPORT	IMPORT READY. . . S. R. OR A**	IMPORT initialization message. Operator input required to acknowledge message. (Depress Break Release button and type either S, R, or A)
IMPORT	IMPORT 1700	Indicates that communications have been established with EXPORT.
BUFFER TRANSLATOR	DIP EXTERNAL REJECT	A graphics console controller has rejected all attempts by the Digigraphic Interrupt Processor to communicate with it; either the data channel cannot be cleared or a hardware failure has occurred. The job associated with that console is automatically aborted.
BUFFER TRANSLATOR	DIP INTERNAL REJECT	The Digigraphic Interrupt Processor routine has received no response when attempting to communicate with a console controller (check Power On and controller switches for proper settings). The job associated with that console is automatically aborted.
BUFFER TRANSLATOR	GICOPY ADDR ERR, NCON y	The Buffer Translator has detected an invalid IDDAD, IDDADI, or MAD programming parameter while processing a buffer from EXPORT that contained a call to the named routine and to console y. This is a non-fatal condition.
BUFFER TRANSLATOR	GIERAS ADDR ERR, NCON y	See above.
BUFFER TRANSLATOR	GIMACE ADDR ERR, NCON y	See above.
BUFFER TRANSLATOR	GIMOVE ADDR ERR, NCON y	See above.
BUFFER TRANSLATOR	GITIMV ADDR ERR, NCON y	See above.
BUFFER TRANSLATOR	GITMMV ADDR ERR, NCON y	See above.
BUFFER TRANSLATOR	GUMACG ADDR ERR, NCON y	See above.

TABLE 9-6. (Cont'd)

Source	Message	Definition
BUFFER TRANS-LATOR	GIxxxx BUFFER OVERFLOW, NCON y	The driver routine for graphics console y has detected a controller memory over-flow condition while processing a request from the Graphics Interface routine with the mnemonic xxxx; this is a non-fatal condition.
BUFFER TRANS-LATOR	GIxxxx EXT REJ, NCON y	The driver routine for graphics console y has detected a controller communica-tion reject while processing a request from the Graphics Interface routine with the mnemonic xxxx; the job containing xxxx is automatically aborted.
BUFFER TRANS-LATOR	GIxxxx INT REJ, NCON y	The driver routine for graphics console y has attempted to process a request from Graphics Interface routine xxxx for a controller that either doesn't exist, isn't turned on, or has suffered a com-munications failure. The job containing xxxx is aborted.
BUFFER TRANS-LATOR	GIxxxx ILLEGAL REQUEST	A graphics console driver has been asked to perform an I/O function it can-not handle; the job containing the Graphics Interface routine xxxx is aborted.
BUFFER TRANS-LATOR	GIxxxx NOT READY, NCON 0	A graphics console driver has encountered a Not Ready hardware condition (check console power switch) while attempting to process an I/O request from Graphics Interface routine xxxx; the job containing GIxxxx is aborted.
BUFFER TRANS-LATOR	GIxxxx SHORT TRANSFER	A graphics console driver has detected the premature termination of a data transfer to or from a controller; be-cause the transfer action requested by Graphics Interface routine xxxx was incomplete, the job containing the routine is aborted.

NOTE

Messages preceded by an asterisk (*) are generated in response to a STAT request or a LIST command.

Messages preceded by two asterisks (**) cause the job name to be cleared from the IMPORT internal job table.

ADDITIONAL STATEMENTS

The PM (SCOPE Print mode) control statements are printed on the teletypewriter with the first two characters set equal to the logical unit of the printer affected. Printing stops to allow operator intervention. The operator can resume printing with the GO command.

In addition to PM statements, all of the Class 2 error messages (Appendix B) produced at the 6000 are printed on the teletypewriter.

ERROR CODES

IMPORT outputs informative codes to the 1700 operator when error conditions occur that are external to the system. Some of these error codes require operator action. Table 9-7 gives an explanation of each code and briefly describes possible operator action.

TABLE 9-7. ERROR CODES

Code	Explanation	Action
**C	Card reader checksum error on a binary card	Card reading stops. The erroneous card is the last card read into the computer. (Non-buffered controllers: Last card in stacker; buffer controllers: Second to last card in stacker.) The operator may reload card reader with erroneous card for re-reading (1 or 2 cards) or ignore the erroneous card. Use of the input statement GO or GO, lu will resume card reading.
**D	Device type error	Entire input statement is ignored by IMPORT.
**E	Job name not found in IMPORT job table	Entire input statement is ignored by IMPORT.
**F	Free form input initiation card error	Card is ignored and operation continues.
**J	Improper job card detected by IMPORT	IMPORT continues to read cards to end of job. Card data transmission resumes with the next job.
**L	Display message length error	Entire input statement is ignored by IMPORT.
**Lu	Unidentified logical unit number	Entire input statement is ignored by IMPORT.

TABLE 9-7 (Cont'd)

Code	Explanation	Action
**M	Message buffers full	Three messages (see DISP 9-10) may be queued; one displayed at the BATCHIO control point, one displayed by IMPORT and one displayed by EXPORT. The fourth cannot be displayed until one or more are acknowledged by the 6000 operator. The 6000 operator may acknowledge a message by typing N.GO. (CR) where N is BATCHIO's control point number.
**P	Improper priority number on a change priority command	Entire input statement is ignored by IMPORT.
**R	Illegal EOR level on an EOR card	EOR level is set to zero and normal operation continues.
**U	Unidentified operator input command	Entire input statement is ignored by IMPORT.
**SQ	6000 format binary card sequence error	Card reading stops. The input statements GO or GO, lu will resume card reading with no further sequence checking prior to reading an End-of-Record or End-of-File. All data is transmitted to the central site.

ERROR REPORTING FORMAT

The 1700 operating system reports errors in the following format:

L, nn FAILED ee

ACTION

nn Logical unit number of the failed device

ee Error code

The 1700 operator may dispose of the above error report with either of two responses:

RP (CR) Directs that the request be repeated.

or

CU (CR) Reports the error to the requesting program. The device is allowed to continue processing requests.

Error codes are defined for each hardware driver and the appropriate manual should be consulted.

GLOSSARY

APPLICATION PROGRAMMER — The programmer who writes graphics programs through the FORTRAN interface called the Basic Graphic Package. The programmer is usually also the graphics console user.

ARGUMENT — Parameters entered by the graphics program in a call to the Basic Graphics Package.

ASSOCIATIVE ADDRESS — Bit pattern that forms the parameter(s) for calls to the Basic Graphics package, i.e., contents of NCON, IDDAD, MAD, IBEAD, NAME, and IFILE.

BASIC GRAPHICS PACKAGE — Collection of FORTRAN callable subroutines that allow access to all the graphics hardware and the data handler.

BATCH JOBS — Programs that are non-real-time and run in the background of graphics.

BEAD — Group of contiguous computer words that may be related to other beads to make up a data structure. Beads contain components and reside in blocks.

BLOCK — Mass storage logical blocks contain beads and are addressed by count. Reside on mass storage and in core.

BUFFER, MEMORY — A storage device attached to the 1744 Controller and used by the Interactive Graphics System for storage of byte-streams during off-line display.

BUTTON — Used to initiate an action from the 274 Console. There are three kinds of buttons:

- Keyboard key
- Light button
- Prime button

BUTTON, PRIME — Allows a display item that is not defined as a button to activate a task when picked, or, is used to temporarily allow a display item to have input significance other than that written into the ID block of the item.

BYTE — A sequence of 12 adjacent binary digits (bits) operated upon as a unit.

COMMON FILE — A file of information that remains in the system, regardless of whether or not it is attached to a program.

COMPONENT — A specific bit, character, or word space within a bead. Each component has a unique address code.

DATA HANDLER — Package which optimizes the use of mass storage and of in-core data file manipulation.

DATA STRUCTURE — A logical relation used in graphics to store relationships for data retrieval.

DIRECTIVE — An IMPORT word code which informs EXPORT of the type of data that is being sent and/or what type of return data is required.

DISPLAY BUFFER — A core memory buffer in the 1744, used for refreshing displays on the 274 Console in an off-line manner.

DISPLAY BYTE-STREAM — Display controller description of the item to be displayed. A serial train of control bytes (see Section 4).

DISPLAY, CORE — A method of graphic display using information stored in computer core memory. Core display is synonymous with on-line display.

DISPLAY ITEM — Any item displayed on 274 Console. Display items are byte-streams placed in the floating address area of the buffer memory, and usually start with a reset sequence and end with an ID block.

DISPLAY, OFF-LINE — A method of graphic display using information stored in 1744 buffer memory which does not require direct computer intervention except to process display change information. Off-line display is synonymous with buffer memory display.

DISPLAY, ON-LINE — See DISPLAY, CORE.

ERASE — An erase function not only removes a display but also removes the pointer from the associative address table block label. The actual bytes of the item are removed from the display controller.

EXPORT/IMPORT — Communications system which permits batch or graphics job submission to a 6000 Series computer from a remote computer.

FILE — 1. A collection of related records treated as a unit.
2. A peripheral device used by a computing system for storing data.

FRAME — A programmer-defined rectangular display on the CRT display surface which encloses the working surface. More than one frame can be specified and displayed at one time.

FRAME-SCISSORING — The process of removing the portion of a display item that exceeds the frame limits. A form of micro-scissoring is done when an item is rescaled such that the item is just a point.

FRAME TIME — Allowed time for any graphics program to remain at a graphics control point. Calculated by the Scheduler routine.

GRAPHICS PROGRAMS — Programs, consisting of many graphic tasks, utilizing the Basic Graphics Package subroutines.

GRAPHICS TASK — Overlay performing one operation, called by a light button or another graphics task.

GRID, DISPLAY — An area consisting of 4096 addressable points on the H and V axes. The display grid circumscribes the display surface such that any combination of points on the H and V axis can be addressed.

HOOK — A 9-bit pointer inserted into a bead address when stringing beads.

ID BLOCK — An identification block of coded information associated with a display item. (See Section 6.)

INPUT, ALPHANUMERIC — Picking characters from a displayed font or inputting from A/N keyboard.

KEYBOARD — Optional input device. There are two types:

- Function
- Alphanumeric

LIGHT BUTTON — Software defined functions displayed on the control surface. They are picked with the light pen and usually call a task to be executed. Light Buttons are items directly related to graphics program options.

LIGHT PEN — A pencil-like bundle of optical fibers which senses the current vertical and horizontal coordinates of the beam and makes them available to the program in order to identify the item that the operator picked.

LIGHT REGISTER — Specific area on the control surface provided for operator input of alphanumeric data. These registers may appear anywhere on the screen, according to the application programmer's wish.

MACRO — The display byte stream for an item which can be displayed in a number of locations on the screen without duplication of the byte-stream.

PICK — The selection of an item with the light pen or function keyboard.

RESET SEQUENCE — Consists of a reset byte to control beam intensity, light pen sense, and blink capabilities; followed by two bytes to establish horizontal and vertical display coordinates to which beam will be set with beam off. In conjunction with last two bytes, a system imposed 25 μsec delay permits beam driving circuits to stabilize.

RESIDENT TIME — Actual time a program has run at a control point.

RESPONSE TIME — The time period between a graphics operator command and the answer he receives.

RESULT — The output of parameters by the Basic Graphics Package.

ROLLIN — The function of transferring a graphics program from mass storage to a control point for execution.

ROLLOUT — The function of transferring a graphics program from a control point to mass storage.

SCHEDULER — A PPU program called by EXPORT to rollout or rollin a graphics program.

SCISSOR — The act of dropping an entity from the display when its coordinate parameters exceed the range of the display grid. This is a software function.

SCISSORING, FRAME — Truncating display items to fit a user defined frame.

SCISSORING, MICRO — The act of non-displaying display items too small to be seen. The cutoff point is 0.025 inch.

SINGLE PICK — A classification given to a display item to cause only the last one of this type picked to remain on the queue.

STATUS CODE — An EXPORT data word which informs the IMPORT program what buffers are available for data I/O.

STRING — A serial linking of display items, buttons, or beads.

STRING PICK — A classification given to a display item to cause each item of this type picked to be put on the end of a string of picked items.

SURFACE, CONTROL — The area reserved for light buttons and light registers. The area on the cathode ray tube display surface exclusive of the working surface. Programmer-defined.

SURFACE, DISPLAY — A 20-inch diameter area on the cathode ray tube screen utilized for man-machine communications. A light, blue, flicker-free display is presented to the operator due to components of the P7 phosphor coating deposited on the inside surface of the cathode ray tube screen.

SURFACE, WORKING — One of two divisions made on the cathode ray tube display surface. The working surface can be enclosed by a frame (viewing window) which is a displayed graphic.

TASK — A program and its subprograms that perform a series of calculations or logical operations. Graphic tasks are, of necessity, as short as possible to define one phase of a multiphase job.

TRACKING — The 1700 Basic Graphic Package function which maintains cognizance of the position of the light pen as it moves across the display surface. A core-displayed tracking cross is used as the light source for the light pen.

TRACKING CROSS — A software displayed item which allows the graphics operator to use the light pen where otherwise no light exists.

UNIT, DISPLAY GRID — The spacing between the 4096 points on the H and V axis of the display grid. A display grid unit is fixed at 0.005 inches and there are 200 display grid units per inch.

USER, CONSOLE — Person who operates a graphics console and uses an application program.

UTILITY PROGRAMS — Programs which support the graphics system, but are not directly involved in graphics program execution.

6000 BASIC GRAPHICS PACKAGE ROUTINE INDEX A

Routine	Page	Routine	Page
AELBUT	7-21	GIFSID	7-23
AERTRN	L-1	GIKYBD	7-14
AETSKC	7-13	GILPKY	7-15
AETSKR	7-13	GIMAC	7-38
DMDMP	7-52	GIMACE	7-39
DMFLSH	7-51	GIMASK	7-19
DMGET	7-53	GIMOVE	7-42
DMGTBD	7-52	GIPBUT	7-16
DMINIT	7-50	GIPLOT	7-56
DMRLBD	7-52	GITCOF	7-45
DMSET	7-53	GITCON	7-45
GFONTA	7-57	GITIMV	7-46
GFONTN	7-58	GITMMV	7-46
GIABRT	7-55	GUAN	7-31
GIANE	7-25	GUARC	7-27
GIANS	7-25	GUARCG	7-36
GIBUT	7-22	GUBYTE	7-37
GICLR	7-20	GULINE	7-26
GICNJB	7-12	GUMACG	7-37
GICNRL	7-12	GURSET	7-30
GICOPY	7-42	GUSEG	7-34
GIDISP	7-40	GUSEGA	7-35
GIEOM	7-15	GUSEGI	7-33
GIERAS	7-41	GUSEGS	7-32
GIFID	7-23	SCHEDR	7-11

GRAPHICS SYSTEM ERROR MESSAGES B

6000 PROGRAMMING DIAGNOSTICS

In addition to the standard FORTRAN compiler, SCOPE loader, and SCOPE execution error diagnostics, the Interactive Graphics System produces several additional diagnostic messages. These diagnostics appear on one or all of the system consoles, and are all entered into the SCOPE dayfile. Dayfile messages pertaining to a specific program are automatically printed with the program's listing.

The special Interactive Graphics error messages are listed below in alphabetic order. Class 1 messages appear only in the SCOPE dayfile (A) and/or job status (B) displays on the 6612 Console screen. Class 2 messages also appear at the 1713 Teletypewriter and on the 274 Console's display screen; they occur only during program execution runs. All messages issued by 6000 Basic Graphics Package routines contain the name of the Package call in which the error was made, and are prefaced by a message which states the name of the task overlay in which the erroneous call occurred.

6000 INPUT/OUTPUT ERRORS

BATCHIO also produces some error messages; these appear only in a program's output file.

1700 ABORT ERRORS

Class 4 error messages are produced by the Buffer Translator, and appear on the console screen and the 1713 Teletypewriter, but are not sent to the 6000 Series machine. If a Class 4 message is associated with a fatal error, it sends an abort flag from the 1700 to the 6000. The Scheduler detects the abort flag and issues the Class 1 error message 1700 ABORT.

Message	Error Type	Class	Meaning	Page References
AEFILE READ ERROR	Fatal	1	Parameters in the File Environment Table indicate a disk read error when control is returned to AEFILE from SCOPE.	2-21
BAD CALL CODE RETURNED-GIANE	Fatal	2	Buffer returned by IMPORT at end of console alphanumeric input does not contain expected valid identification code.	
BAD CALL PARAMETER	Fatal	1	EXPORT has encountered a service request (at RA+76_8 of a graphics job's control point area) with meaningless contents.	3-8, 3-9
BAD NAME CHECK rcrdnam	Fatal	1	Issued by AELOAD utility routine; the name of record rcrdnam in source file does not correspond to any entry in the file index.	2-22, 2-25
BYTE ARRAY EXCEEDS 255-GUBYTE	Non-Fatal	1	There are only 8 bits in the 1700 Package version of the N parameter, so N in this call cannot exceed 255_{10}.	7-36
BYTE ARRAY INDEX ZERO-GUBYTE	Non-Fatal	1	The N parameter in this GUBYTE call is zero, so the call is ignored.	7-29, 7-37
DISPLAY ITEM BUFFER EXCEEDED -GIDISP	Fatal	2	The total number of bytes in the user's IBUF (excluding GIDISP header bytes and trailing ID bytes) exceeds the 310 decimal maximum.	7-29, 7-40 M1
DISPLAY ITEM NBYTE EQUALS ZERO-GIDISP	Non-Fatal	1	Since the description buffer is empty, the call is ignored.	7-29, 7-40
EMPTY FILE filenam	Fatal	1	Issued by AEDUMP; the first record in source file filenam indicates that the file was created from an empty random file.	2-22
EOR NOT READ ON TASK LOAD	Fatal	2	Parameters in the File Environment Table indicate a disk read error during a task call when control is returned to MAIN from SCOPE.	
EXPORT IS NOT UP	Fatal	1	The Scheduler has been called to begin graphics job execution, but EXPORT has not been loaded to handle the job's communications; this message is produced by a communication failure, not a programming error.	3-2, 9-1, 9-6

Message	Error Type	Class	Meaning	Page References
FORMAT ERROR FIRST DATA RECORD	Fatal	1	The first data record of the input file for the job contains no cards or an illegal file name; names must be seven or fewer characters, and must contain no special characters. Produced by MAIN.	2-13
GICOPY ADDR ERR, NCON y	Non-Fatal	4	The 6000 Package routine named has sent the Buffer Translator an invalid IDDAD, IDDADI, or MAD parameter for use on console y.	7-42, 9-14
GICOPY BUFFER OVERFLOW, NCON y	Non-Fatal	4	Console y controller memory has overflowed because of the named call.	9-15
GIDISP BUFFER OVERFLOW, NCON y	Non-Fatal	4	Console y controller memory has overflowed because of the named call.	9-15
GIERAS ADDR ERR, NCON y	Non-Fatal	4	The 6000 Package routine named has sent the Buffer Translator an invalid IDDAD, IDDADI, or MAD parameter for use on console y.	7-41, 9-14
GIMAC BUFFER OVERFLOW, NCON y	Non-Fatal	4	Console y controller memory has overflowed because of the named call.	9-15
GIMAC CALL IGNORED– NBYTE = 0	Non-Fatal	1	Since the programmer has specified that his description buffer is empty, this call is ignored.	
GIMAC ADDR ERR, NCON y	Non-Fatal	4	The 6000 Package routine named has sent the Buffer Translator an invalid IDDAD, IDDADI, or MAD parameter for use on console y.	7-38,
GIMOVE ADDR ERR, NCON y	Non-Fatal	4	See above	7-43, 9-14
GITIMV ADDR ERR, NCON y	Non-Fatal	4	See above	7-46, 9-14
GITMMV ADDR ERR, NCON y	Non-Fatal	4	See above	7-46, 9-14
GUAN CALL IGNORED-NC is ZERO OR NEGATIVE	Non-Fatal	1	Self-explanatory.	7-29, 7-31

Message	Error Type	Class	Meaning	Page References
GUARCG CALL IGNORED-KSHOW ILLEGAL	Non-Fatal	1	The KSHOW parameter is negative or greater than 5	7-29, 7-36
GUARCG CALL IGNORED-ZERO RADIUS ARC	Non-Fatal	1	If IH1, IV1 or IH2, IV2 equals IHC, IVC no arc can be generated.	7-29, 7-36
GUMACG ADDR ERR, NCON y	Non-Fatal	4	The 6000 Package routine named has sent the Buffer Translator an invalid IDDAD, IDDADI, or MAD parameter for use on console y.	7-37, 9-14
GUSEGA CALL IGNORED-N ZERO OR NEGATIVE	Non-Fatal	1	Self-explanatory	7-29, 7-35
ILLEGAL COORDINATE-GITCON	Fatal	2	One of the programmer's tracking cross coordinates is beyond the extent of the display grid (not between -2048 and +2048).	4-5, 7-45
ILLEGAL COORDINATE RETURNED-GITCOF	Fatal	2	One of the tracking cross coordinates from the last button pick is not within the display grid (is less than -2048 or greater than +2048).	4-5, 7-45
ILLEGAL IBEAD-DMGET	Fatal	2	Programmer's bead address either: ● Has an index = 0 ● Has a block number = 0 ● Has a block number greater than the number of existing blocks	7-5, 7-50 7-53
ILLEGAL IBEAD-DMRLBD	Non-Fatal	1	Same as above.	7-5, 7-52
ILLEGAL IBEAD-DMSET	Fatal	2	Same as above.	7-5, 7-53
ILLEGAL IBEAM-GUSEG	Non-Fatal	1	The programmer's beam control parameter is not either 0 or 1.	7-34
ILLEGAL IBEAM-GUSEGS	Non-Fatal	1	Same as above.	7-32
ILLEGAL ICOMP-DMGET	Fatal	2	The programmer's component code contained either: ● Type code = 0 or 9, or greater than 10 ● Word or character number greater than the size of the bead specified by the accompanying IBEAD value	7-47, 7-53
ILLEGAL ICOMP-DMSET	Fatal	2	Same as above.	7-47, 7-53

Message	Error Type	Class	Meaning	Page References
ILLEGAL NBLK-DMINIT	Fatal	2	The number of Data Handler data blocks that the programmer wishes kept in core as copies is either: • Less than the minimum of 2 the Handler needs to function properly • Larger than the number that will fit in core	7-50
ILLEGAL NBSIZE-DMINIT	Fatal	2	The block size specified in this call is larger than the maximum permissible size of an in-core data base.	7-9, 7-50
ILLEGAL NUMBER OF WORDS REQUESTED-DMGTBD	Fatal	2	The programmer is trying to define a bead with a length (N parameter) less than or equal to zero, or $\geq 2^{18}$	7-52
INCORRECT ICODE-GICOPY	Non-Fatal	1	The programmer's reset control code is not a form or value significant to the 1700 version of this routine; a significant ICODE value will be substituted for the one supplied.	7-42
INCORRECT ICODE-GIMOVE	Non-Fatal	1	Same as above.	7-43
INCORRECT ICODE-GURSET	Non-Fatal	1	Same as above.	7-30
INCORRECT NCON-GIFID	Fatal	2	The programmer is trying to fetch a single pick ID block from a console other than the one from which the last button pick ID was fetched. The GIFID NCON must always agree with the NCON of the last GIBUT call.	7-23
INCORRECT NCON-GIFSID	Fatal	2	Same as above.	7-23
ITEM NOT CREATED FOR THIS NCON-GITIMV	Fatal	2	The IDDAD value given does not exist for this console; either the NCON or the IDDAD parameter supplied in this call is wrong.	7-46
JOB NOT ATTACHED TO RANDOM TASK FILE	Fatal	1	Either: • AEDUMP couldn't find the file named in its first parameter field • MAIN couldn't find the file named on its graphics COMMON file name parameter card If the file name is both legal and correct as given, then the file has not been attached to the job by a control card.	2-10, 2-11, 2-13, 2-24, 7-2

Message	Error Type	Class	Meaning	Page References
MACRO BUFFER LENGTH EXCEED-ED-GIMAC	Fatal	2	The total number of bytes in the user's IBUF (excluding GIMAC header and trailer bytes) exceeds the maximum of 310 decimal.	7-29, 7-38, 7-41, M-1
MACRO NOT CREATED FOR THIS NCON-GITMMV	Fatal	2	The MAD value given does not exist for this console; either the NCON or the MAD parameter supplied in this call is wrong.	7-46
MAD ARRAY INDEX ZERO-GUMACG	Non-Fatal	1	The N parameter given in this call indicates that no macro should be created, so this call is ignored.	7-29, 7-37
NBSIZE DIFFERS FROM PREVIOUS DEFINITION-DMINIT	Fatal	2	DMINIT has been called with a block size different from the block size declared by a previous job using IFILE.	7-50
NBYTE EXCEEDS MBYTE-GUAN	Non-Fatal	1	This Graphics Utilities call has produced more bytes in IBUF than the programmer wants.	7-29, 7-31
NBYTE EXCEEDS MBYTE-GUARCG	Non-Fatal	1	Same as above.	7-29, 7-36
NBYTE EXCEEDS MBYTE – GUBYTE	Non-Fatal	1	Same as above.	7-29, 7-37
NBYTE EXCEEDS MBYTE – GUMACG	Non-Fatal	1	Same as above.	7-29, 7-37
NBYTE EXCEEDS MBYTE – GURSET	Non-Fatal	1	Same as above.	7-29, 7-30
NBYTE EXCEEDS MBYTE – GUSEG	Non-Fatal	1	Same as above.	7-29, 7-34
NBYTE EXCEEDS MBYTE – GUSEGA	Non-Fatal	1	Same as above.	7-29, 7-35
NBYTE EXCEEDS MBYTE – GUSEGI	Non-Fatal	1	Same as above.	7-29, 7-33
NBYTE EXCEEDS MBYTE – GUSEGS	Non-Fatal	1	Same as above.	7-29, 7-32
NCON ERROR	Fatal	1	EXPORT has detected a service request ($RA+76_8$ of a graphics job's control point area) with an invalid NCON parameter; either the console being addressed does not exist, or it is not attached to this job.	3-8, 3-9, 7-12
NC RETURNED GREATER THAN MAXIMUM-GIANE	Non-Fatal	1	The number of characters picked exceeds the maximum the programmer wants sent to the calling task; the excess trailing characters are dropped automatically.	7-25

Message	Error Type	Class	Meaning	Page References
NC TOO LARGE -GIANE	Fatal	2	More characters have been picked than can be passed to the task; EXPORT can handle a maximum of 620_{10} characters.	7-25
NC TOO LARGE -GIANS	Fatal	2	The programmer is willing to accept more input characters than EXPORT can handle; maximum is 620_{10}.	7-25
NO ACTIVE GRAPHICS CP.	Fatal	1	The Scheduler cannot find a graphics control point to which the program can be assigned. The system operator has not assigned a control point for graphics use, so the job cannot be executed; no programming error has occurred.	9-1
NO DATA HANDLER FILE OPEN-DMDMP	Fatal	2	The programmer is trying to dump a non-existent IFILE; either: ● DMINIT has not yet been called ● DMINIT has not been called since the last DMFLSH call	7-50, 7-51, 7-52
NO DATA HANDLER FILE OPEN-DMGET	Fatal	2	The programmer is trying to obtain data in a non-existant IFILE; see above.	7-50, 7-51, 7-52
NO DATA HANDLER FILE OPEN-DMGTBD	Fatal	2	The programmer is asking for space in a non-existant IFILE; see above.	7-50, 7-51, 7-52
NO DATA HANDLER FILE OPEN-DMRLBD	Fatal	2	The programmer is trying to clear space in a non-existant IFILE; see above.	7-50, 7-51, 7-52
NO DATA HANDLER FILE OPEN-DMSET	Fatal	2	The programmer is attempting to place data in a non-existant IFILE; see above.	7-50, 7-51, 7-53
NO INITIAL POINT GENERATED-GUSEG	Fatal	2	GUSEG has been called without a previous GUSEGI or GUSEGS call to initialize the figure that the programmer wants to generate.	7-33, 7-34
NO TASK NAME IN BUTTON ID-AETSKR	Non-Fatal	1	Produced by MAIN when AETSKR has been called; IDWA and IDWB of the button in the FETCH queue do not contain information that can be used to load a task; either: ● IDWA = 0 ● The ID block ends short of IDWA ● The bit pattern in IDWA and IDWB does not match a task name in the index	7-13, 7-39

Message	Error Type	Class	Meaning	Page References
NO TC COORDI-NATES THIS NCON-GITCOF	Non-Fatal	1	No tracking cross coordinates can be returned because the NCON of the last button picked doesn't match the NCON supplied in this call.	7-44
PREFIX TABLE FORMAT ERROR-AEFILE	Fatal	1	AEFILE has detected an illegal Prefix table while creating the task file from the overlay scratch file.	2-21
PROGRAM NAME NOT IN FILE CATALOG	Fatal	2	AETSKC cannot find the required task in the directory of the job's graphics COMMON file.	2-22, 7-13
QUEUE TABLE FULL	Fatal	1	The Scheduler has no room in its graphics input queue for this job, so the job cannot be assigned to a control point. The job should be run again when there are fewer graphics jobs in the system.	
RETURN ADDRESS OVERLAYED OR MISSING - AERTRN	Non-Fatal	1	Either AETSKC has never been called or the return address of AETSKC has been over-written by a task load since the last call. After issuing the message, AERTRN exits to AETSKR.	
TASK tasknam		2	Issued by the MAIN error processor before all fatal and nonfatal error mess-ages; tasknam contains the name of the task overlay in which the error occurred.	
TASK ********		2	MAIN issues this when it appears that the contents of the reservation word in MAIN for the current task have been destroyed or when jobs are being run outside applications executive interface.	N-2, N-3
TOO MANY DATA HANDLER FILES CREATED - DMINIT	Fatal	2	DMINIT has been called to create more than the installation-specified number of Data Handler files.	7-50
TOO MANY FILES	Fatal	1	More than 8 files are attached to the job which the scheduler is attempting to roll out.	
1700 ABORT	Fatal	1	Either a Class 4 error or one of the system problems mentioned in Section 9 has caused the 1700 to abort the job. Error correction may have to be done through the 1700.	

CHARACTER CODE EQUIVALENTS

STANDARD FORTRAN CHARACTERS (01–37)

6000 Internal Display Code	Printed Character (standard 6000 set)	1713 Teletypewriter	274 Display Character†††	Alphanumeric or Numeric Keyboard†††	ASCII OR 1700 Hexadecimal Internal Code†	6000 Hollerith (punched card rows)††	Internal BCD	External BCD	EBCDIC Hollerith (punched card rows)	8-bit EBCDIC Hexadecimal Code	EBCDIC Character	ICT 1900 Hollerith (punched card rows)	ICT 1900 Character
01	A	A	A	A	41	12, 1	21	61	12, 1	C1	A	12, 1	A
02	B	B	B	B	42	12, 2	22	62	12, 2	C2	B	12, 2	B
03	C	C	C	C	43	12, 3	23	63	12, 3	C3	C	12, 3	C
04	D	D	D	D	44	12, 4	24	64	12, 4	C4	D	12, 4	D
05	E	E	E	E	45	12, 5	25	65	12, 5	C5	E	12, 5	E
06	F	F	F	F	46	12, 6	26	66	12, 6	C6	F	12, 6	F
07	G	G	G	G	47	12, 7	27	67	12, 7	C7	G	12, 7	G
10	H	H	H	H	48	12, 8	30	70	12, 8	C8	H	12, 8	H
11	I	I	I	I	49	12, 9	31	71	12, 9	C9	I	12, 9	I
12	J	J	J	J	4A	11, 1	41	41	11, 1	D1	J	11, 1	J
13	K	K	K	K	4B	11, 2	42	42	11, 2	D2	K	11, 2	K
14	L	L	L	L	4C	11, 3	43	43	11, 3	D3	L	11, 3	L
15	M	M	M	M	4D	11, 4	44	44	11, 4	D4	M	11, 4	M
16	N	N	N	N	4E	11, 5	45	45	11, 5	D5	N	11, 5	N
17	O	O	O	O	4F	11, 6	46	46	11, 6	D6	O	11, 6	O
20	P	P	P	P	50	11, 7	47	47	11, 7	D7	P	11, 7	P
21	Q	Q	Q	Q	51	11, 8	50	50	11, 8	D8	Q	11, 8	Q
22	R	R	R	R	52	11, 9	51	51	11, 9	D9	R	11, 9	R
23	S	S	S	S	53	0, 2	62	22	0, 2	E2	S	0, 2	S
24	T	T	T	T	54	0, 3	63	23	0, 3	E3	T	0, 3	T
25	U	U	U	U	55	0, 4	64	24	0, 4	E4	U	0, 4	U
26	V	V	V	V	56	0, 5	65	25	0, 5	E5	V	0, 5	V
27	W	W	W	W	57	0, 6	66	26	0, 6	E6	W	0, 6	W
30	X	X	X	X	58	0, 7	67	27	0, 7	E7	X	0, 7	X
31	Y	Y	Y	Y	59	0, 8	70	30	0, 8	E8	Y	0, 8	Y
32	Z	Z	Z	Z	5A	0, 9	71	31	0, 9	E9	Z	0, 9	Z
33	0	0	0	0	30	0	00	12	0	F0	0	0	0
34	1	1	1	1	31	1	01	01	1	F1	1	1	1
35	2	2	2	2	32	2	02	02	2	F2	2	2	2
36	3	3	3	3	33	3	03	03	3	F3	3	3	3
37	4	4	4	4	34	4	04	04	4	F4	4	4	4

† 8-bit ASCII, used for communication with 1713 Teletypewriter

†† 0, 11 is equivalent to 11, 8, 2 and 0, 12 is equivalent to 12, 8, 2

††† CLEAR, TAB, BACKSPACE, and EOM have input control significance only

STANDARD FORTRAN CHARACTERS (codes 40–57)

6000 Internal Display Code	Printed Character (standard 6000 set)	1713 Teletypewriter	274 Display Character†††	Alphanumeric or Numeric Keyboard†††	ASCII OR 1700 Hexadecimal Internal Code†	6000 Hollerith (punched card rows)††	Internal BCD	External BCD	EBCDIC Hollerith (punched card rows)	8-bit EBCDIC Hexadecimal Code	EBCDIC Character	ICT 1900 Hollerith (punched card rows)	ICT 1900 Character
40	5	5	5	5	35	5	05	05	5	F5	5	5	5
41	6	6	6	6	36	6	06	06	6	F6	6	6	6
42	7	7	7	7	37	7	07	07	7	F7	7	7	7
43	8	8	8	8	38	8	10	10	8	F8	8	8	8
44	9	9	9	9	39	9	11	11	9	F9	9	9	9
45	+	+	+	+	2B	12	20	60	13, 8, 6	4E	+	12, 8, 2	+
46	-	-	-	-	2D	11	40	40	11	60	-	11	-
47	*	*	*	*	2A	11, 4, 8	54	54	11, 4, 8	5C	*	11, 8, 4	*
50	/	/	/	/	2F	0, 1	61	21	0, 1	61	/	0, 1	/
51	(	(	(	(	28	0, 4, 8	74	34	12, 8, 5	4D	(	8, 5	(
52	)	)	)	)	29	12, 4, 8	34	74	11, 8, 5	5D	)	8, 6	)
53	$	$	$	$	23	11, 3, 8	53	53	11, 3, 8	5B	$	11, 8, 3	$
54				=	3D	3, 8	13	13	8, 6	7E	=	0, 8, 6	=
55	blank	space		space	20	space	60	20	space	40	blank	space	' blank
56	,	,	,	,	2C	0, 3, 8	73	33	0, 3, 8	6B	,	0, 8, 3	,
57	.	.	.	.	2E	12, 3, 8	33	73	12, 3, 8	4B	.	12, 8, 3	.
60	≡	#	backspace	←	5F	0, 6, 8	76	36	3, 8		#	8, 3	#
61	[	[	?	?	5B	7, 8	17	17	7, 8		¢	11, 8, 2	[
62	]	]			5D	0, 2, 8	72	32	0, 5, 8		underline	8, 7	]
63	:	:	:	:	3A	2, 8	12	00	2, 8	7A	:	12, 8, 5	:
64	≠				27	4, 8	14	14	5, 8		'	12, 8, 6	'
65	→	@	tab	TAB	40	0, 5, 8	75	35	4, 8		@	8, 4	@
66	∨ (OR)	\	clear	RUBOUT	21	0, 11	52	52	7, 8		"	11, 0	"
67	∧ (AND)	%	EOM	RETURN	3F	0, 7, 8	77	37	12, 11		∧	0, 8, 2	£
70	↑	↑		↑	23	11, 5, 8	55	55	12, 8, 7		|	11, 8, 7	↑
71	↓	!		!	5C	11, 6, 8	56	56	11, 8, 2		!	12, 8, 7	!
72	<	<		<	3C	0, 12	32	72	12, 8, 4	4C	<	11, 8, 6	<
73	>	>		>	3E	11, 7, 8	57	57	0, 8, 6	6E	>	11, 8, 5	>
74	≤	&		&	26	5, 8	15	15	12		&	12	&
75	≥	?			5E	12, 5, 8	35	75	0, 8, 7		?	0, 8, 5	?
76	¬ (NOT)	←		←	7C	12, 6, 8	36	76	11, 8, 7	5F	¬	0, 8, 7	←
77	;	;		;	3B	12, 7, 8	37	77	11, 8, 6	5E	;	12, 8, 4	;

† 8-bit ASCII, used for communication with 1713 Teletypewriter

†† 0, 11 is equivalent to 11, 8, 2 and 0, 12 is equivalent to 12, 8, 2

††† CLEAR, TAB, BACKSPACE, and EOM have input control significance only

SAMPLE DATA HANDLER FILE DUMP D

The task overlay shown below creates a Data Handler file called DMFILE containing five blocks of information. It then prints the edited and labelled file dump reproduced on the following pages. Although the call to DMDMP prints out all five file blocks, only the first two are shown here because of space limitations.

```
            OVERLAY (1,0)
            PROGRAM DMPTASK
            DIMENSION IB(8), IVAL (512)
            CALL DMINIT (6LDMFILE,5)        Initialize 5 duplicate blocks in
                                            core for file DMFILE
            IC = 070000000001B  <---------  Set basic component code value
            DO 4 I = 1, 8                 }
4           CALL DMGTBD (64*I,IB(I))      } Obtains all needed beads from file
            DO 45 I = 1, 8                }
            M = I * 64                    }
            IF (I.NE.1) GO TO 2           }
            IVAL (1) = 0                  }
            IVAL (2) = 1                  }
            IVAL (3) = 1                  }
            IVAL (4) = 1                  }
            IVAL (5) = 2                  }
            IVAL (6) = 2                  }
            IVAL (7) = 1                  } Arbitrary establishment of data
            IVAL (8) = 1                  } to be placed in file
            IVAL (9) = 1                  }
            IVAL (10) = 0                 }
            IVAL (11) = 0                 }
            IVAL (12) = 1                 }
            IVAL (13) = 2                 }
            DO 3 IN = 14, 64              }
3           IVAL (IN) = 0                 }
            GO TO 43                      }
2           MB = I * 8 ** 5               }
            DO 43 N = 1, M                }
42          IVAL (N) = MB + N             }
```

```
43          ICOMP = M * 8 ** 6 + IC                      Increment component code for each pass
45          CALL DMSET (ICOMP, IB(I),IVAL)               Store value
            DO 50 I = 1, 512          }
50          IVAL (I) = 0              }                  Zero out value buffer for next use
            CALL DMDMP                                   Print dump of DMFILE
            END
```

As the following printout shows, each dump produced by DMDMP is preceded by a line stating the name of the dumped file; each block in the file is preceded by a line stating the relative number of the dumped block and the amount of empty space within it; each bead in a block is preceded by a line stating its relative index number within the block. Continuation beads are marked and pointers to them are given after the printout of each file block.

```
DUMP OF DATA HANDLER FILE -  DMFILE

BLOCK 0001 EMPTY SPACE 0000

Bead  001

000001   00000 00000 00000 [illegible]          00000 00000 00000 00001
000005   00000 00000 00000 00002   000007*00000 00000 00000 00001   000012*00000 00000 00000 00000   000014*00000 00000 00000 00001
000015   00000 00000 00000 00002          00000 00000 00000 00000   000100*00000 00000 00000 00000

Bead  002

000001   00000 00000 00002 00001     00000 00000 00002 00002     00000 00000 00002 00003     00000 00000 00002 00004
000005   00000 00000 00002 00005     00000 00000 00002 00006     00000 00000 00002 00007     00000 00000 00002 00010
000011   00000 00000 00002 00011     00000 00000 00002 00012     00000 00000 00002 00013     00000 00000 00002 00014
000015   00000 00000 00002 00015     00000 00000 00002 00016     00000 00000 00002 00017     00000 00000 00002 00020
000021   00000 00000 00002 00021     00000 00000 00002 00022     00000 00000 00002 00023     00000 00000 00002 00024
000025   00000 00000 00002 00025     00000 00000 00002 00026     00000 00000 00002 00027     00000 00000 00002 00030
000031   00000 00000 00002 00031     00000 00000 00002 00032     00000 00000 00002 00033     00000 00000 00002 00034
000035   00000 00000 00002 00035     00000 00000 00002 00036     00000 00000 00002 00037     00000 00000 00002 00040
000041   00000 00000 00002 00041     00000 00000 00002 00042     00000 00000 00002 00043     00000 00000 00002 00044
000045   00000 00000 00002 00045     00000 00000 00002 00046     00000 00000 00002 00047     00000 00000 00002 00050
000051   00000 00000 00002 00051     00000 00000 00002 00052     00000 00000 00002 00053     00000 00000 00002 00054
000055   00000 00000 00002 00055     00000 00000 00002 00056     00000 00000 00002 00057     00000 00000 00002 00060
000061   00000 00000 00002 00061     00000 00000 00002 00062     00000 00000 00002 00063     00000 00000 00002 00064
000065   00000 00000 00002 00065     00000 00000 00002 00066     00000 00000 00002 00067     00000 00000 00002 00070
000071   00000 00000 00002 00071     00000 00000 00002 00072     00000 00000 00002 00073     00000 00000 00002 00074
000075   00000 00000 00002 00075     00000 00000 00002 00076     00000 00000 00002 00077     00000 00000 00002 00100
000101   00000 00000 00002 00101     00000 00000 00002 00102     00000 00000 00002 00103     00000 00000 00002 00104
000105   00000 00000 00002 00105     00000 00000 00002 00106     00000 00000 00002 00107     00000 00000 00002 00110
000111   00000 00000 00002 00111     00000 00000 00002 00112     00000 00000 00002 00113     00000 00000 00002 00114
000115   00000 00000 00002 00115     00000 00000 00002 00116     00000 00000 00002 00117     00000 00000 00002 00120
000121   00000 00000 00002 00121     00000 00000 00002 00122     00000 00000 00002 00123     00000 00000 00002 00124
000125   00000 00000 00002 00125     00000 00000 00002 00126     00000 00000 00002 00127     00000 00000 00002 00130
000131   00000 00000 00002 00131     00000 00000 00002 00132     00000 00000 00002 00133     00000 00000 00002 00134
000135   00000 00000 00002 00135     00000 00000 00002 00136     00000 00000 00002 00137     00000 00000 00002 00140
000141   00000 00000 00002 00141     00000 00000 00002 00142     00000 00000 00002 00143     00000 00000 00002 00144
000145   00000 00000 00002 00145     00000 00000 00002 00146     00000 00000 00002 00147     00000 00000 00002 00150
000151   00000 00000 00002 00151     00000 00000 00002 00152     00000 00000 00002 00153     00000 00000 00002 00154
000155   00000 00000 00002 00155     00000 00000 00002 00156     00000 00000 00002 00157     00000 00000 00002 00160
000161   00000 00000 00002 00161     00000 00000 00002 00162     00000 00000 00002 00163     00000 00000 00002 00164
000165   00000 00000 00002 00165     00000 00000 00002 00166     00000 00000 00002 00167     00000 00000 00002 00170
000171   00000 00000 00002 00171     00000 00000 00002 00172     00000 00000 00002 00173     00000 00000 00002 00174
000175   00000 00000 00002 00175     00000 00000 00002 00176     00000 00000 00002 00177     00000 00000 00002 00200

Bead  003

000001   00000 00000 00003 00001     00000 00000 00003 00002     00000 00000 00003 00003     00000 00000 00003 00004
000005   00000 00000 00003 00005     00000 00000 00003 00006     00000 00000 00003 00007     00000 00000 00003 00010
000011   00000 00000 00003 00011     00000 00000 00003 00012     00000 00000 00003 00013     00000 00000 00003 00014
000015   00000 00000 00003 00015     00000 00000 00003 00016     00000 00000 00003 00017     00000 00000 00003 00020
000021   00000 00000 00003 00021     00000 00000 00003 00022     00000 00000 00003 00023     00000 00000 00003 00024
000025   00000 00000 00003 00025     00000 00000 00003 00026     00000 00000 00003 00027     00000 00000 00003 00030
000031   00000 00000 00003 00031     00000 00000 00003 00032     00000 00000 00003 00033     00000 00000 00003 00034
000035   00000 00000 00003 00035     00000 00000 00003 00036     00000 00000 00003 00037     00000 00000 00003 00040
000041   00000 00000 00003 00041     00000 00000 00003 00042     00000 00000 00003 00043     00000 00000 00003 00044
000045   00000 00000 00003 00045     00000 00000 00003 00046     00000 00000 00003 00047     00000 00000 00003 00050
000051   00000 00000 00003 00051     00000 00000 00003 00052     00000 00000 00003 00053     00000 00000 00003 00054
000055   00000 00000 00003 00055     00000 00000 00003 00056     00000 00000 00003 00057     00000 00000 00003 00060
000061   00000 00000 00003 00061     00000 00000 00003 00062     00000 00000 00003 00063     00000 00000 00003 00064
000065   00000 00000 00003 00065     00000 00000 00003 00066     00000 00000 00003 00067     00000 00000 00003 00070
```

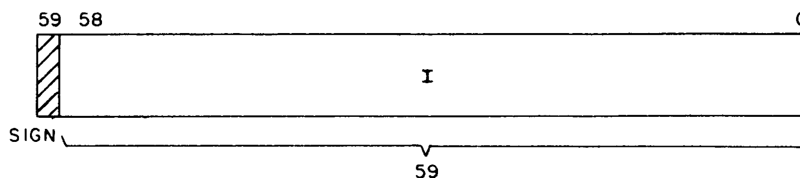
```
000071   00000 00000 00003 00071   00000 00000 00003 00072   00000 00000 00003 00073   00000 00000 00003 00074
000075   00000 00000 00003 00075   00000 00000 00003 00076   00000 00000 00003 00077   00000 00000 00003 00100
000101   00000 00000 00003 00101   00000 00000 00003 00102   00000 00000 00003 00103   00000 00000 00003 00104
000105   00000 00000 00003 00105   00000 00000 00003 00106   00000 00000 00003 00107   00000 00000 00003 00110
000111   00000 00000 00003 00111   00000 00000 00003 00112   00000 00000 00003 00113   00000 00000 00003 00114
000115   00000 00000 00003 00115   00000 00000 00003 00116   00000 00000 00003 00117   00000 00000 00003 00120
000121   00000 00000 00003 00121   00000 00000 00003 00122   00000 00000 00003 00123   00000 00000 00003 00124
000125   00000 00000 00003 00125   00000 00000 00003 00126   00000 00000 00003 00127   00000 00000 00003 00130
000131   00000 00000 00003 00131   00000 00000 00003 00132   00000 00000 00003 00133   00000 00000 00003 00134
000135   00000 00000 00003 00135   00000 00000 00003 00136   00000 00000 00003 00137   00000 00000 00003 00140
000141   00000 00000 00003 00141   00000 00000 00003 00142   00000 00000 00003 00143   00000 00000 00003 00144
000145   00000 00000 00003 00145   00000 00000 00003 00146   00000 00000 00003 00147   00000 00000 00003 00150
000151   00000 00000 00003 00151   00000 00000 00003 00152   00000 00000 00003 00153   00000 00000 00003 00154
000155   00000 00000 00003 00155   00000 00000 00003 00156   00000 00000 00003 00157   00000 00000 00003 00160
000161   00000 00000 00003 00161   00000 00000 00003 00162   00000 00000 00003 00163   00000 00000 00003 00164
000165   00000 00000 00003 00165   00000 00000 00003 00166   00000 00000 00003 00167   00000 00000 00003 00170
000171   00000 00000 00003 00171   00000 00000 00003 00172   00000 00000 00003 00173   00000 00000 00003 00174
000175   00000 00000 00003 00175   00000 00000 00003 00176   00000 00000 00003 00177   00000 00000 00003 00200
000201   00000 00000 00003 00201   00000 00000 00003 00202   00000 00000 00003 00203   00000 00000 00003 00204
000205   00000 00000 00003 00205   00000 00000 00003 00206   00000 00000 00003 00207   00000 00000 00003 00210
000211   00000 00000 00003 00211   00000 00000 00003 00212   00000 00000 00003 00213   00000 00000 00003 00214
000215   00000 00000 00003 00215   00000 00000 00003 00216   00000 00000 00003 00217   00000 00000 00003 00220
000221   00000 00000 00003 00221   00000 00000 00003 00222   00000 00000 00003 00223   00000 00000 00003 00224
000225   00000 00000 00003 00225   00000 00000 00003 00226   00000 00000 00003 00227   00000 00000 00003 00230
000231   00000 00000 00003 00231   00000 00000 00003 00232   00000 00000 00003 00233   00000 00000 00003 00234
000235   00000 00000 00003 00235   00000 00000 00003 00236   00000 00000 00003 00237   00000 00000 00003 00240
000241   00000 00000 00003 00241   00000 00000 00003 00242   00000 00000 00003 00243   00000 00000 00003 00244
000245   00000 00000 00003 00245   00000 00000 00003 00246   00000 00000 00003 00247   00000 00000 00003 00250
000251   00000 00000 00003 00251   00000 00000 00003 00252   00000 00000 00003 00253   00000 00000 00003 00254
000255   00000 00000 00003 00255   00000 00000 00003 00256   00000 00000 00003 00257   00000 00000 00003 00260
000261   00000 00000 00003 00261   00000 00000 00003 00262   00000 00000 00003 00263   00000 00000 00003 00264
000265   00000 00000 00003 00265   00000 00000 00003 00266   00000 00000 00003 00267   00000 00000 00003 00270
000271   00000 00000 00003 00271   00000 00000 00003 00272   00000 00000 00003 00273   00000 00000 00003 00274
000275   00000 00000 00003 00275   00000 00000 00003 00276   00000 00000 00003 00277   00000 00000 00003 00300

Bead 004

000001   00000 00000 00005 00001   00000 00000 00005 00002   00000 00000 00005 00003   00000 00000 00005 00004
000005   00000 00000 00005 00005   00000 00000 00005 00006   00000 00000 00005 00007   00000 00000 00005 00010
000011   00000 00000 00005 00011   00000 00000 00005 00012   00000 00000 00005 00013   00000 00000 00005 00014
000015   00000 00000 00005 00015   00000 00000 00005 00016   00000 00000 00005 00017   00000 00000 00005 00020
000021   00000 00000 00005 00021   00000 00000 00005 00022   00000 00000 00005 00023   00000 00000 00005 00024
000025   00000 00000 00005 00025   00000 00000 00005 00026   00000 00000 00005 00027   00000 00000 00005 00030
000031   00000 00000 00005 00031   00000 00000 00005 00032   00000 00000 00005 00033   00000 00000 00005 00034
000035   00000 00000 00005 00035   00000 00000 00005 00036   00000 00000 00005 00037   00000 00000 00005 00040
000041   00000 00000 00005 00041   00000 00000 00005 00042   00000 00000 00005 00043   00000 00000 00005 00044
000045   00000 00000 00005 00045   00000 00000 00005 00046   00000 00000 00005 00047   00000 00000 00005 00050
000051   00000 00000 00005 00051   00000 00000 00005 00052   00000 00000 00005 00053   00000 00000 00005 00054
000055   00000 00000 00005 00055   00000 00000 00005 00056   00000 00000 00005 00057   00000 00000 00005 00060
000061   00000 00000 00005 00061   00000 00000 00005 00062   00000 00000 00005 00063   00000 00000 00005 00064
000065   00000 00000 00005 00065   00000 00000 00005 00066   00000 00000 00005 00067   00000 00000 00005 00070
000071   00000 00000 00005 00071   00000 00000 00005 00072   00000 00000 00005 00073   00000 00000 00005 00074
000075   00000 00000 00005 00075   00000 00000 00005 00076   00000 00000 00005 00077   00000 00000 00005 00100
000101   00000 00000 00005 00101   00000 00000 00005 00102   00000 00000 00005 00103   00000 00000 00005 00104
000105   00000 00000 00005 00105   00000 00000 00005 00106   00000 00000 00005 00107   00000 00000 00005 00110
000111   00000 00000 00005 00111   00000 00000 00005 00112   00000 00000 00005 00113   00000 00000 00005 00114
000115   00000 00000 00005 00115   00000 00000 00005 00116   00000 00000 00005 00117   00000 00000 00005 00120
000121   00000 00000 00005 00121   00000 00000 00005 00122   00000 00000 00005 00123   00000 00000 00005 00124
000125   00000 00000 00005 00125   00000 00000 00005 00126   00000 00000 00005 00127   00000 00000 00005 00130
000131   00000 00000 00005 00131   00000 00000 00005 00132   00000 00000 00005 00133   00000 00000 00005 00134
000135   00000 00000 00005 00135   00000 00000 00005 00136   00000 00000 00005 00137   00000 00000 00005 00140
000141   00000 00000 00005 00141   00000 00000 00005 00142   00000 00000 00005 00143   00000 00000 00005 00144
000145   00000 00000 00005 00145   00000 00000 00005 00146   00000 00000 00005 00147   00000 00000 00005 00150
000151   00000 00000 00005 00151   00000 00000 00005 00152   00000 00000 00005 00153   00000 00000 00005 00154
000155   00000 00000 00005 00155   00000 00000 00005 00156   00000 00000 00005 00157   00000 00000 00005 00160
000161   00000 00000 00005 00161   00000 00000 00005 00162   00000 00000 00005 00163   00000 00000 00005 00164
000165   00000 00000 00005 00165   00000 00000 00005 00166   00000 00000 00005 00167   00000 00000 00005 00170
000171   00000 00000 00005 00171

                    CONTINUE AS BLOCK 0002 INDEX 002

BLOCK 0002 EMPTY SPACE 0000

Bead 001

000001   00000 00000 00004 00001   00000 00000 00004 00002   00000 00000 00004 00003   00000 00000 00004 00004
000005   00000 00000 00004 00005   00000 00000 00004 00006   00000 00000 00004 00007   00000 00000 00004 00010
000011   00000 00000 00004 00011   00000 00000 00004 00012   00000 00000 00004 00013   00000 00000 00004 00014
000015   00000 00000 00004 00015   00000 00000 00004 00016   00000 00000 00004 00017   00000 00000 00004 00020
000021   00000 00000 00004 00021   00000 00000 00004 00022   00000 00000 00004 00023   00000 00000 00004 00024
000025   00000 00000 00004 00025   00000 00000 00004 00026   00000 00000 00004 00027   00000 00000 00004 00030
000031   00000 00000 00004 00031   00000 00000 00004 00032   00000 00000 00004 00033   00000 00000 00004 00034
000035   00000 00000 00004 00035   00000 00000 00004 00036   00000 00000 00004 00037   00000 00000 00004 00040
000041   00000 00000 00004 00041   00000 00000 00004 00042   00000 00000 00004 00043   00000 00000 00004 00044
000045   00000 00000 00004 00045   00000 00000 00004 00046   00000 00000 00004 00047   00000 00000 00004 00050
000051   00000 00000 00004 00051   00000 00000 00004 00052   00000 00000 00004 00053   00000 00000 00004 00054
000055   00000 00000 00004 00055   00000 00000 00004 00056   00000 00000 00004 00057   00000 00000 00004 00060
000061   00000 00000 00004 00061   00000 00000 00004 00062   00000 00000 00004 00063   00000 00000 00004 00064
000065   00000 00000 00004 00065   00000 00000 00004 00066   00000 00000 00004 00067   00000 00000 00004 00070
000071   00000 00000 00004 00071   00000 00000 00004 00072   00000 00000 00004 00073   00000 00000 00004 00074
000075   00000 00000 00004 00075   00000 00000 00004 00076   00000 00000 00004 00077   00000 00000 00004 00100
000101   00000 00000 00004 00101   00000 00000 00004 00102   00000 00000 00004 00103   00000 00000 00004 00104
000105   00000 00000 00004 00105   00000 00000 00004 00106   00000 00000 00004 00107   00000 00000 00004 00110
000111   00000 00000 00004 00111   00000 00000 00004 00112   00000 00000 00004 00113   00000 00000 00004 00114
000115   00000 00000 00004 00115   00000 00000 00004 00116   00000 00000 00004 00117   00000 00000 00004 00120
000121   00000 00000 00004 00121   00000 00000 00004 00122   00000 00000 00004 00123   00000 00000 00004 00124
000125   00000 00000 00004 00125   00000 00000 00004 00126   00000 00000 00004 00127   00000 00000 00004 00130
000131   00000 00000 00004 00131   00000 00000 00004 00132   00000 00000 00004 00133   00000 00000 00004 00134
000135   00000 00000 00004 00135   00000 00000 00004 00136   00000 00000 00004 00137   00000 00000 00004 00140
000141   00000 00000 00004 00141   00000 00000 00004 00142   00000 00000 00004 00143   00000 00000 00004 00144
000145   00000 00000 00004 00145   00000 00000 00004 00146   00000 00000 00004 00147   00000 00000 00004 00150
000151   00000 00000 00004 00151   00000 00000 00004 00152   00000 00000 00004 00153   00000 00000 00004 00154
000155   00000 00000 00004 00155   00000 00000 00004 00156   00000 00000 00004 00157   00000 00000 00004 00160
000161   00000 00000 00004 00161   00000 00000 00004 00162   00000 00000 00004 00163   00000 00000 00004 00164
000165   00000 00000 00004 00165   00000 00000 00004 00166   00000 00000 00004 00167   00000 00000 00004 00170
000171   00000 00000 00004 00171   00000 00000 00004 00172   00000 00000 00004 00173   00000 00000 00004 00174
```

```
000175   00000 00000 00004 00175   00000 00000 00004 00176   00000 00000 00004 00177   00000 00000 00004 00200
000201   00000 00000 00004 00201   00000 00000 00004 00202   00000 00000 00004 00203   00000 00000 00004 00204
000205   00000 00000 00004 00205   00000 00000 00004 00206   00000 00000 00004 00207   00000 00000 00004 00210
000211   00000 00000 00004 00211   00000 00000 00004 00212   00000 00000 00004 00213   00000 00000 00004 00214
000215   00000 00000 00004 00215   00000 00000 00004 00216   00000 00000 00004 00217   00000 00000 00004 00220
000221   00000 00000 00004 00221   00000 00000 00004 00222   00000 00000 00004 00223   00000 00000 00004 00224
000225   00000 00000 00004 00225   00000 00000 00004 00226   00000 00000 00004 00227   00000 00000 00004 00230
000231   00000 00000 00004 00231   00000 00000 00004 00232   00000 00000 00004 00233   00000 00000 00004 00234
000235   00000 00000 00004 00235   00000 00000 00004 00236   00000 00000 00004 00237   00000 00000 00004 00240
000241   00000 00000 00004 00241   00000 00000 00004 00242   00000 00000 00004 00243   00000 00000 00004 00244
000245   00000 00000 00004 00245   00000 00000 00004 00246   00000 00000 00004 00247   00000 00000 00004 00250
000251   00000 00000 00004 00251   00000 00000 00004 00252   00000 00000 00004 00253   00000 00000 00004 00254
000255   00000 00000 00004 00255   00000 00000 00004 00256   00000 00000 00004 00257   00000 00000 00004 00260
000261   00000 00000 00004 00261   00000 00000 00004 00262   00000 00000 00004 00263   00000 00000 00004 00264
000265   00000 00000 00004 00265   00000 00000 00004 00266   00000 00000 00004 00267   00000 00000 00004 00270
000271   00000 00000 00004 00271   00000 00000 00004 00272   00000 00000 00004 00273   00000 00000 00004 00274
000275   00000 00000 00004 00275   00000 00000 00004 00276   00000 00000 00004 00277   00000 00000 00004 00300
000301   00000 00000 00004 00301   00000 00000 00004 00302   00000 00000 00004 00303   00000 00000 00004 00304
000305   00000 00000 00004 00305   00000 00000 00004 00306   00000 00000 00004 00307   00000 00000 00004 00310
000311   00000 00000 00004 00311   00000 00000 00004 00312   00000 00000 00004 00313   00000 00000 00004 00314
000315   00000 00000 00004 00315   00000 00000 00004 00316   00000 00000 00004 00317   00000 00000 00004 00320
000321   00000 00000 00004 00321   00000 00000 00004 00322   00000 00000 00004 00323   00000 00000 00004 00324
000325   00000 00000 00004 00325   00000 00000 00004 00326   00000 00000 00004 00327   00000 00000 00004 00330
000331   00000 00000 00004 00331   00000 00000 00004 00332   00000 00000 00004 00333   00000 00000 00004 00334
000335   00000 00000 00004 00335   00000 00000 00004 00336   00000 00000 00004 00337   00000 00000 00004 00340
000341   00000 00000 00004 00341   00000 00000 00004 00342   00000 00000 00004 00343   00000 00000 00004 00344
000345   00000 00000 00004 00345   00000 00000 00004 00346   00000 00000 00004 00347   00000 00000 00004 00350
000351   00000 00000 00004 00351   00000 00000 00004 00352   00000 00000 00004 00353   00000 00000 00004 00354
000355   00000 00000 00004 00355   00000 00000 00004 00356   00000 00000 00004 00357   00000 00000 00004 00360
000361   00000 00000 00004 00361   00000 00000 00004 00362   00000 00000 00004 00363   00000 00000 00004 00364
000365   00000 00000 00004 00365   00000 00000 00004 00366   00000 00000 00004 00367   00000 00000 00004 00370
000371   00000 00000 00004 00371   00000 00000 00004 00372   00000 00000 00004 00373   00000 00000 00004 00374
000375   00000 00000 00004 00375   00000 00000 00004 00376   00000 00000 00004 00377   00000 00000 00004 00400

Bead 002
CONTINUATION BEAD

000001   00000 00000 00005 00172   00000 00000 00005 00173   00000 00000 00005 00174   00000 00000 00005 00175
000005   00000 00000 00005 00176   00000 00000 00005 00177   00000 00000 00005 00200   00000 00000 00005 00201
000011   00000 00000 00005 00202   00000 00000 00005 00203   00000 00000 00005 00204   00000 00000 00005 00205
000015   00000 00000 00005 00206   00000 00000 00005 00207   00000 00000 00005 00210   00000 00000 00005 00211
000021   00000 00000 00005 00212   00000 00000 00005 00213   00000 00000 00005 00214   00000 00000 00005 00215
000025   00000 00000 00005 00216   00000 00000 00005 00217   00000 00000 00005 00220   00000 00000 00005 00221
000031   00000 00000 00005 00222   00000 00000 00005 00223   00000 00000 00005 00224   00000 00000 00005 00225
000035   00000 00000 00005 00226   00000 00000 00005 00227   00000 00000 00005 00230   00000 00000 00005 00231
000041   00000 00000 00005 00232   00000 00000 00005 00233   00000 00000 00005 00234   00000 00000 00005 00235
000045   00000 00000 00005 00236   00000 00000 00005 00237   00000 00000 00005 00240   00000 00000 00005 00241
000051   00000 00000 00005 00242   00000 00000 00005 00243   00000 00000 00005 00244   00000 00000 00005 00245
000055   00000 00000 00005 00246   00000 00000 00005 00247   00000 00000 00005 00250   00000 00000 00005 00251
000061   00000 00000 00005 00252   00000 00000 00005 00253   00000 00000 00005 00254   00000 00000 00005 00255
000065   00000 00000 00005 00256   00000 00000 00005 00257   00000 00000 00005 00260   00000 00000 00005 00261
000071   00000 00000 00005 00262   00000 00000 00005 00263   00000 00000 00005 00264   00000 00000 00005 00265
000075   00000 00000 00005 00266   00000 00000 00005 00267   00000 00000 00005 00270   00000 00000 00005 00271
000101   00000 00000 00005 00272   00000 00000 00005 00273   00000 00000 00005 00274   00000 00000 00005 00275
000105   00000 00000 00005 00276   00000 00000 00005 00277   00000 00000 00005 00300   00000 00000 00005 00301
000111   00000 00000 00005 00302   00000 00000 00005 00303   00000 00000 00005 00304   00000 00000 00005 00305
000115   00000 00000 00005 00306   00000 00000 00005 00307   00000 00000 00005 00310   00000 00000 00005 00311
000121   00000 00000 00005 00312   00000 00000 00005 00313   00000 00000 00005 00314   00000 00000 00005 00315
000125   00000 00000 00005 00316   00000 00000 00005 00317   00000 00000 00005 00320   00000 00000 00005 00321
000131   00000 00000 00005 00322   00000 00000 00005 00323   00000 00000 00005 00324   00000 00000 00005 00325
000135   00000 00000 00005 00326   00000 00000 00005 00327   00000 00000 00005 00330   00000 00000 00005 00331
000141   00000 00000 00005 00332   00000 00000 00005 00333   00000 00000 00005 00334   00000 00000 00005 00335
000145   00000 00000 00005 00336   00000 00000 00005 00337   00000 00000 00005 00340   00000 00000 00005 00341
000151   00000 00000 00005 00342   00000 00000 00005 00343   00000 00000 00005 00344   00000 00000 00005 00345
000155   00000 00000 00005 00346   00000 00000 00005 00347   00000 00000 00005 00350   00000 00000 00005 00351
000161   00000 00000 00005 00352   00000 00000 00005 00353   00000 00000 00005 00354   00000 00000 00005 00355
000165   00000 00000 00005 00356   00000 00000 00005 00357   00000 00000 00005 00360   00000 00000 00005 00361
000171   00000 00000 00005 00362   00000 00000 00005 00363   00000 00000 00005 00364   00000 00000 00005 00365
000175   00000 00000 00005 00366   00000 00000 00005 00367   00000 00000 00005 00370   00000 00000 00005 00371
000201   00000 00000 00005 00372   00000 00000 00005 00373   00000 00000 00005 00374   00000 00000 00005 00375
000205   00000 00000 00005 00376   00000 00000 00005 00377   00000 00000 00005 00400   00000 00000 00005 00401
000211   00000 00000 00005 00402   00000 00000 00005 00403   00000 00000 00005 00404   00000 00000 00005 00405
000215   00000 00000 00005 00406   00000 00000 00005 00407   00000 00000 00005 00410   00000 00000 00005 00411
000221   00000 00000 00005 00412   00000 00000 00005 00413   00000 00000 00005 00414   00000 00000 00005 00415
000225   00000 00000 00005 00416   00000 00000 00005 00417   00000 00000 00005 00420   00000 00000 00005 00421
000231   00000 00000 00005 00422   00000 00000 00005 00423   00000 00000 00005 00424   00000 00000 00005 00425
000235   00000 00000 00005 00426   00000 00000 00005 00427   00000 00000 00005 00430   00000 00000 00005 00431
000241   00000 00000 00005 00432   00000 00000 00005 00433   00000 00000 00005 00434   00000 00000 00005 00435
000245   00000 00000 00005 00436   00000 00000 00005 00437   00000 00000 00005 00440   00000 00000 00005 00441
000251   00000 00000 00005 00442   00000 00000 00005 00443   00000 00000 00005 00444   00000 00000 00005 00445
000255   00000 00000 00005 00446   00000 00000 00005 00447   00000 00000 00005 00450   00000 00000 00005 00451
000261   00000 00000 00005 00452   00000 00000 00005 00453   00000 00000 00005 00454   00000 00000 00005 00455
000265   00000 00000 00005 00456   00000 00000 00005 00457   00000 00000 00005 00460   00000 00000 00005 00461
000271   00000 00000 00005 00462   00000 00000 00005 00463   00000 00000 00005 00464   00000 00000 00005 00465
000275   00000 00000 00005 00466   00000 00000 00005 00467   00000 00000 00005 00470   00000 00000 00005 00471
000301   00000 00000 00005 00472   00000 00000 00005 00473   00000 00000 00005 00474   00000 00000 00005 00475
000305   00000 00000 00005 00476   00000 00000 00005 00477   00000 00000 00005 00500

Bead 003

000001   00000 00000 00010 00001   00000 00000 00010 00002   00000 00000 00010 00003   00000 00000 00010 00004
000005   00000 00000 00010 00005   00000 00000 00010 00006   00000 00000 00010 00007   00000 00000 00010 00010
000011   00000 00000 00010 00011   00000 00000 00010 00012   00000 00000 00010 00013   00000 00000 00010 00014
000015   00000 00000 00010 00015   00000 00000 00010 00016   00000 00000 00010 00017   00000 00000 00010 00020
000021   00000 00000 00010 00021   00000 00000 00010 00022   00000 00000 00010 00023   00000 00000 00010 00024
000025   00000 00000 00010 00025   00000 00000 00010 00026   00000 00000 00010 00027   00000 00000 00010 00030
000031   00000 00000 00010 00031   00000 00000 00010 00032   00000 00000 00010 00033   00000 00000 00010 00034
000035   00000 00000 00010 00035   00000 00000 00010 00036   00000 00000 00010 00037   00000 00000 00010 00040
000041   00000 00000 00010 00041   00000 00000 00010 00042   00000 00000 00010 00043   00000 00000 00010 00044
000045   00000 00000 00010 00045   00000 00000 00010 00046   00000 00000 00010 00047   00000 00000 00010 00050
000051   00000 00000 00010 00051   00000 00000 00010 00052   00000 00000 00010 00053   00000 00000 00010 00054
000055   00000 00000 00010 00055   00000 00000 00010 00056   00000 00000 00010 00057   00000 00000 00010 00060
000061   00000 00000 00010 00061   00000 00000 00010 00062   00000 00000 00010 00063

          CONTINUE AS BLOCK 0005 INDEX 001
```

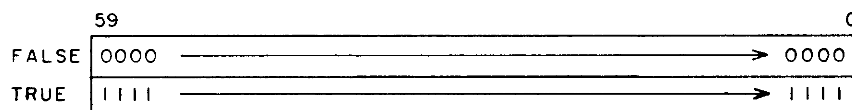
6000 SERIES CENTRAL MEMORY WORD ORGANIZATION

E

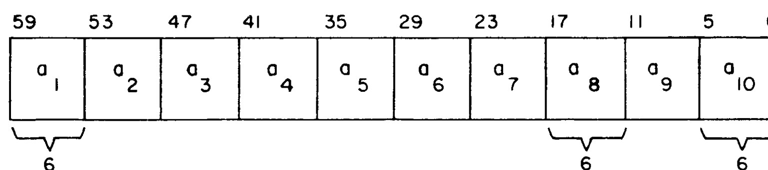
INTEGER



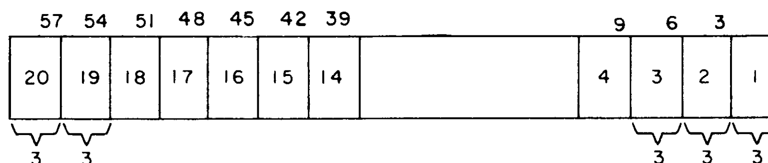
LOGICAL



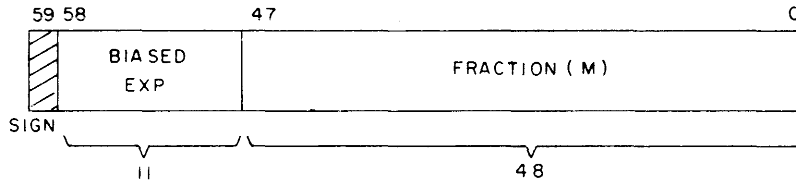
HOLLERITH BCD AND DISPLAY CODE



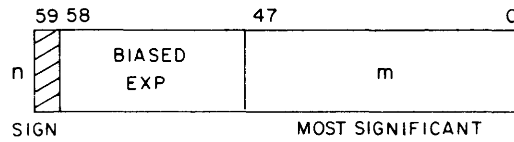
OCTAL

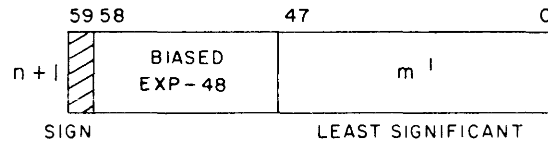


REAL

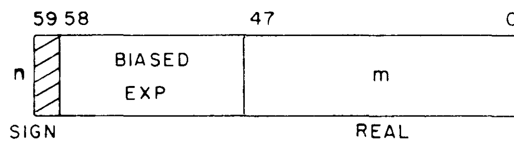


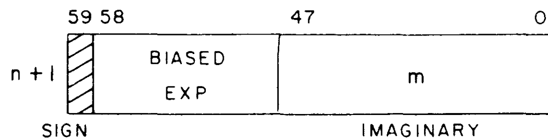
DOUBLE PRECISION



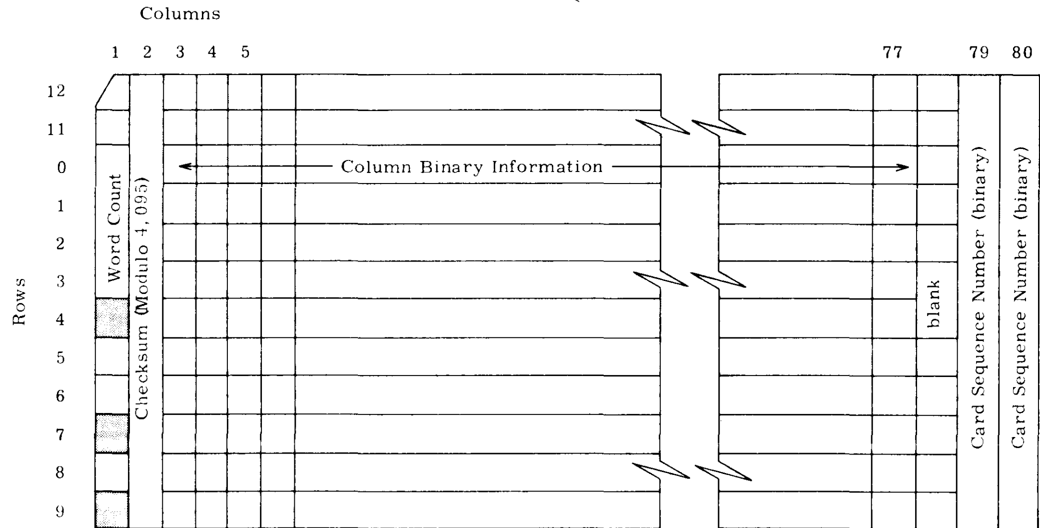


COMPLEX





CARD FORMATS F



NORMAL MODE

Card Type	Column 1 Punch	Other Columns
Enter Free-Form Mode	All rows	All of one other column punched, other columns blank.
*		
End-of-File	6, 7, 8, 9	All other columns ignored.
End-of-Record	7, 8, 9	All other columns ignored.
Standard 6000 Binary Card	7, 9	See below
Coded Card	Not 7, 9	See below

*References to ICT, EBCDIC and Hollerith switch cards apply only to the BATCHIO 6000 card reader driver and not to IMPORT.

BINARY CARDS

A Normal mode binary card can contain 15 central memory words of data, starting in column 3. Rows 0, 1, 2 and 3 of column 1 contain the number of central memory words on the card; if no punch occurs in row 4 of column 1, then column 2 contains a binary checksum (modulo 4,095) for the card.

Columns 3 through 77 contain the central memory data words.

Column 78 is blank; columns 79 and 80 contain a 24-bit binary card sequence number, which starts at 1 on the first card of each record.

CODED CARDS

Coded cards contain up to 80 characters per card. When the system reads coded cards, it converts the data to display code from the Hollerith type specified on the last Hollerith Switch card; if no Hollerith Switch card has been read, the system assumes that conversion is to be made from standard 6000 Hollerith code.

When the system converts 6000 Hollerith code, it packs the data 10 card columns per central memory word. Trailing blanks are suppressed during all types of code conversions, and an end-of-line terminator is forced for each card.

FREE-FORM MODE

Card Type	Column 1 Punch	Other Columns
Exit Free-Form Mode	All rows	Must be identical to last card used to enter free-form mode
Absolute End-of-File	6, 7, 8, 9	Columns 2 through 80 all blank.

All other cards are read as 80-column binary, with 16 central memory words of data per card. An Absolute End-of-File card writes an End-of-File mark and causes processing to return to Normal mode.

CYCLIC ERROR DETECTION G

To ensure reliable transmission of binary data via a Telpak-A line, or a coaxial cable, it is necessary to transmit redundant information which enables the receiving Data Set Controller (DSC) to determine if any errors have occurred. Noise generated at switching centers, lightning, electrical disturbances, or random line noise cause errors.

The DSC utilizes a cyclic code as a method for detecting transmission errors. This type of code is defined in terms of a generating polynomial, $G(x)$, of degree n-k, where n = total bits transmitted and k = the data bits transmitted. Using this polynomial, k information bits can be augmented with n-k redundant bits in such a way that n bits can be described by a code polynomial of a degree $\leq n-1$, that makes possible the detection of most error patterns.

Binary data can readily be associated with an algebraic polynomial. For instance, the binary information 1, 0, 1, 1, 0, 0, 1, is used to describe the generating polynomial (reading from left to right) $G(x) = x^6 + 0x^5 + x^4 + x^3 + 0x^2 + 0x + 1$, or omitting the terms with zero coefficients, $G(x) = x^6 + x^4 + x^3 + 1$. The rules of ordinary algebra permit the addition, subtraction, multiplication, or division of this polynomial. The arithmetic is based on the modulus 2; i.e., it obeys the following laws:

$$0 + 0 = 1 + 1 - 0, \; -1 = +1, \; 1 + 0 = 0 + 1 = 1$$

The code polynomial is generated in the following manner: first, multiply the information polynomial, $I(x)$, representing the information to be transmitted by x^{n-k}. The product, $x^{n-k} I(x)$ is then divided by the generating polynomial $G(x)$. The operation can be denoted as follows:

$$\frac{x^{n-k}I(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

or $x^{n-k}I(x) = Q(X)\, G(x) + R(x)$ where

$Q(x)$ is the quotient and $R(x)$ is the remainder of the code polynomial. Using the modulus 2 arithmetic operations, (1) can be written in the following manner:

$$x^{n-k}I(x) - R(x) = Q(x)\, A(x)$$

But since -1 = +1, the left side can also be written as:

$$x^{n-k}I(x) + R(x) = Q(x)\, A(x)$$

This equation states that if the remainder is added to $x^{n-k}I(x)$, a polynomial divisible by $G(x)$ results since the right side is obviously divisible by $G(x)$. For example, suppose the bit

sequence 1, 0, 1, 1, 0, 1 is to be sent as a data message with three redundant bits (the DSC uses 12 bits) added for error-checking purposes. The information can be represented by $I(x) = x^5 + x^3 + x^2 + 1$. Since three redundant bits are going to be used, $I(x)$ must be multiplied by x^3. This yields $x^8 + x^6 + x^5 + x^3$. Suppose $G(x) = x^3 + x + 1$ is chosen as a third degree generating polynomial, then:

$$
\frac{x^3 I(x)}{G(x)} = \qquad
\begin{array}{r|l}
 & x^5 + 1 = Q(x) \\
\hline
x^3 + x + 1 & x^8 + x^6 + x^5 + x^3 \\
 & \underline{x^8 + x^5 + x^5} \\
 & \qquad\qquad x^3 \\
 & \qquad\qquad \underline{x^3 + x + 1} \\
 & \qquad\qquad\quad x + 1 = R(x)
\end{array}
$$

Thus, $x^3 I(x) + R(x) = x^8 + x^6 + x^5 + x^3 + x + 1$ is the coded polynomial divisible by $G(x)$. The information actually transmitted would be 1, 0, 1, 1, 0, 1, 0, 1, 1.

The data is checked for divisibility by $G(x)$ at the receiving station as follows:

$$
\begin{array}{r|l}
 & x^5 + 1 \\
\hline
x^3 + x + 1 & x^8 + x^6 + x^5 + x^3 + x + 1 \\
 & \underline{x^8 + x^6 + x^5 \qquad\qquad\qquad} \\
 & \qquad\qquad x^3 + x + 1 \\
 & \qquad\qquad \underline{x^3 + x + 1} \\
 & \qquad\qquad\qquad\quad 0
\end{array}
$$

If instead of 1, 0, 1, 1, 0, 1, 0, 1, 1 being received, suppose the sequence 1, 0, 0, 1, 0, 1, 0, 0, 1 is received. Checking this yields:

$$
\begin{array}{r|l}
 & x^5 + x^3 + x \\
\hline
x^3 + x + 1 & x^8 + x^5 + x^3 + 1 \\
 & \underline{x^8 + x^6 + x^5 \quad} \\
 & \qquad x^6 + x^3 + 1 \\
 & \qquad \underline{x^6 + x^4 + x^3} \\
 & \qquad\qquad x^4 + 1 \\
 & \qquad\qquad \underline{x^4 + x^2 + x} \\
 & \qquad\qquad\quad x^2 + x + 1 = R(x)
\end{array}
$$

The remainder is $R(x) = x^2 + x + 1$. Since the remainder is not zero, an error occurred during transmission and was detected. Undetected errors occur when the received polynomial is divisible by $G(x)$, the generating polynomial, even though it is not necessarily the intended code. The rarity of this occurrence gives the cyclic code detection high reliability.

The multiplication of the information polynomial by x^{n-k} and then division by the generating polynomial G(x) is implemented by using shift registers with suitable linear feedback connections. Detection of errors is accomplished by sensing the shift register for "1's" after the data message and information has been received at the receiving end.

A block diagram of a typical encoder-decoder is shown in Figure G-1 below. The generating polynomial actually utilized is $G(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$ and the feedback connections shown perform the multiplication of information polynomial, I(x), by x^{12}, and then add the remainder of I(x)/G(x) to I(x). Control gate 2 is activated all during the time I(x) is being shifted through the encoder. This gate is deactivated after the last bit of I(x) is in the first stage of the encoder. At this time, control gate 1 is activated and the contents of the shift register are shifted out, augmenting the data. At the receiving end, control gate 2 is activated all during the time the information and redundant bits are being transmitted. When the last transmitted bit has been shifted into the first stage of the register, the entire register is sampled and if a "1" does not occur in any stage, it is assumed that the information received is identical to that which was transmitted. If any stage contains a "1", at least one bit was aborted during transmission.

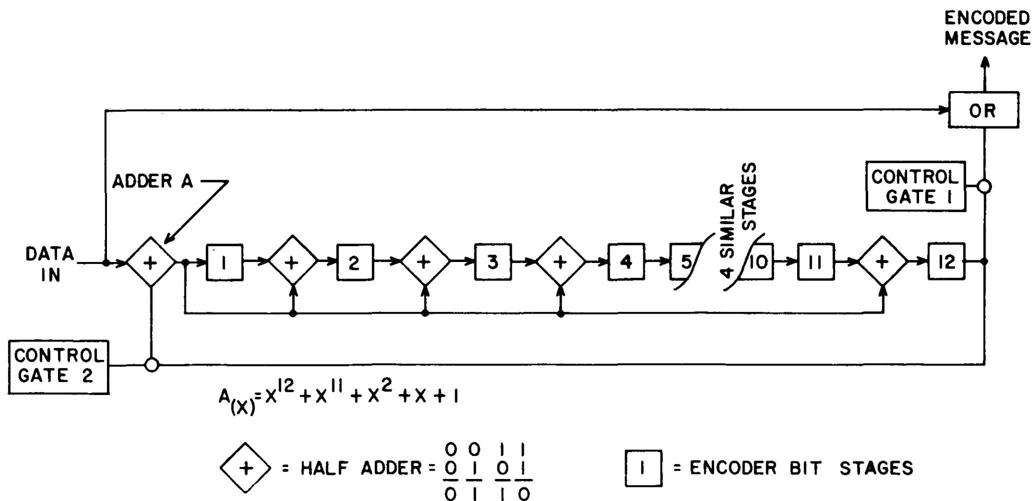


Figure G-1. Typical Encoder/Decoder

SAMPLE GRAPHICS PROGRAMS H

The following printout lists all of the cards needed for the file creation and execution runs of a simple graphics job. This job consists of one short primary overlay that draws a star and a square at one console, then creates two light buttons. The console user is informed that the square is supposed to be within the star; he then picks the proper button to center the square within the star, and the figure is moved. If the button he picks is invalid, he receives a message and the job aborts.

CARD SEQUENCE FOR GRAPHICS FILE CREATION

```
XMPL,P37,T1000,CM60000.
RUN(S)
LGO.
AEFILE.
7
8
9

      OVERLAY(SCR,0,0)
      PROGRAM CREATE
      CALL MAIN
      STOP
      END

      OVERLAY (1,0)
      PROGRAM LITTLE
      DIMENSION IBUF (64),IBCD(30)
      NCON=1

C     CONNECT CONSOLE
      CALL GICNJB(NCON)

C     SET ITEM MASKS
      DO 10 I=1,5
      IDDTS=2** (I-1)
      IMASK=IDDTS
      CALL GIMASK(NCON, -0,IDDTS,IMASK)
   10 CONTINUE

C     SET DISPLAY CONSTANTS
      ICODE=103B
      ISTYLE=7777B
      MBYTE=310
      NBYTE=0

C     START DRAWING SQUARE
      CALL GURSET (200,200,ICODE,IBUF,NBYTE,MBYTE)
      CALL GUSEGI (200,200,ISTYLE,IBUF,NBYTE,MBYTE)
      CALL GUSEG(200,-200,1)
```

```
      CALL GUSEG(-200, -200, 1)
      CALL GUSEG(-200, 200, 1)
      CALL GUSEG(200, 200, 1)
      IDDT=2
C     DISPLAY SQUARE AS SINGLE PICK ITEM
      CALL GIDISP(NCON, IBUF, NBYTE, IDDAD, IDDT, 1, -0)
      IDRSV=IDDAD
C     DRAW STAR OF DAVID (TWO TRIANGLES)
      NBYTE=0
      CALL GURSET (-500, 300, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUSEGS (-500, 300, 500, 300, 1, ISTYLE, IBUF, NBYTE, MBYTE)
      CALL GUSEGS (500, 300, 0, -600, 1, ISTYLE, IBUF, NBYTE, MBYTE)
      CALL GUSEGS (0, -600, -500, 300, 1, ISTYLE, IBUF, NBYTE, MBYTE)
C     DRAW SECOND TRIANGLE
      CALL GUSEGS (-500, -300, 500, -300, 1, ISTYLE, IBUF, NBYTE, MBYTE)
      CALL GUSEGS (500, -300, 0, 600, 1, ISTYLE, IBUF, NBYTE, MBYTE)
      CALL GUSEGS (0, 600, -500, -300, 1, ISTYLE, IBUF, NBYTE, MBYTE)
      IDDT=24B
C     DISPLAY STAR AS STRING PICK WITH MARKER MASK SET
      CALL GIDISP(NCON, IBUF, NBYTE, IDDAD , IDDT, 2, -0)
C     LABEL THE FIGURES
      NBYTE=0
      ENCODE (37, 20, IBCD)
   20 FORMAT(37HSUPPOSED TO BE A SQUARE INSIDE A STAR)
      NC=37
      CALL GURSET(-300, -700, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUAN(IBCD, NC, IBUF, NBYTE, MBYTE)
C     DISPLAY ITEM AS AN IGNORE ITEM
      CALL GIDISP(NCON, IBUF, NBYTE, IDDAD, 1, 3, -0)
C     MAKE TWO BUTTON CHOICES
      NBYTE=0
      CALL GURSET(-400, -900, ICODE, IBUF, NBYTE, MBYTE)
      ENCODE (16, 30, IBCD)
   30 FORMAT(16HMOVE SQUARE LEFT)
      NC=16
      CALL GUAN(IBCD, NC, IBUF, NBYTE, MBYTE)
      IDDT=8
      CALL GIDISP(NCON, IBUF, NBYTE, IDDAD, IDDT, 4, -0)
      NBYTE=0
      CALL GURSET(300, -900, ICODE, IBUF, NBYTE, MBYTE)
      ENCODE (17, 35, IBCD)
   35 FORMAT(17HMOVE SQUARE RIGHT)
      NC=17
      CALL GUAN(IBCD, NC, IBUF, NBYTE, MBYTE)
      CALL GIDISP(NCON, IBUF, NBYTE, IDDAD, IDDT, 5, -0)
      CALL GIBUT(0, NCON, IDDT, IDDC)
      IF (IDDC.EQ.4) GO TO 41
      IF (IDDC.EQ.5) GO TO 42
      ENCODE (10, 40, IBCD)
   40 FORMAT(10HWRONG IDDT)
      NC=10
      CALL GIABRT(NCON, IBCD, NC)
      STOP
```

```
   41 IH=-600
      GO TO 43
   42 IH=1000
C     MOVE ITEM AS SPECIFIED BY OPERATOR
   43 CALL GIMOVE(IH,200,107B,IDRSV,2,6,-0)
C     MOVED ITEM WILL BLINK
C     RELEASE CONSOLE
      CALL GICNRL(NCON)
      STOP
      END
7
8
9
```

Two file names cause MAIN to recognize the file creation job:

```
XMPL
SCR
6
7
8
9
```

CARD SEQUENCE FOR GRAPHICS EXECUTION JOB

```
LPMX,P37,T200,CM60000.
COMMON,XMPL.
RUN(S)
LGO.
RELEASE,XMPL.
EXIT.
RELEASE,XMPL.
7
8
9
      OVERLAY(SCR,0,0)
      PROGRAM EXECUTE
      CALL MAIN
      STOP
      END
7
8
9
```

One file name causes MAIN to recognize the file execution job:

```
XMPL
6
7
8
9
```

The following example consists of a program which sets the ID processor mask, generates a line and a circle with subroutines, and generates four light buttons; one to display a line, one to display a circle, one to erase a picked line or circle, and one to terminate the application. Coordinate information for display item position will be furnished by calls to the tracking cross position fetch routine GITCOF.

The console number, NCON, is input from a card and stored in COMMON location NCON. The program is designed to terminate automatically when 10 components have been created.

PROGRAM CLASDEM

The program starts by reading a card for the console number and storing it in COMMON location NCON. Console NCON is assigned to the job with a call to GICNJB.

The byte-stream for the line is generated and stored in IBUF with a call to GUSEGS; it is made a macro and stored in the macro area of the display buffer with a call to GIMAC. The macro address is returned in MAD (1).

The byte-stream for the circle is generated and stored in IBUF with a call to GUARCG; it is made a macro and stored in the macro area of the display buffer with a call to GIMAC. The macro address is returned in MAD (2). The macro addresses are needed when the line or circle is to be displayed. The MAD parameters are used in calls to GUMACG which generates the macro call. The macro call, provides access to the line and circle byte-streams for display.

Once the byte-streams are taken care of, the next step is to set up the ID processor mask for the light buttons and components. This is done with calls to GIMASK. Light buttons are designated type 1 and are set to blink when picked as operator feedback. Lines and circles designated type 2, are set to blink when picked, and are further designated as single pick items. This last means that if more than one line or circle is picked for erasure, only the last one picked has its ID block retained; the preceding ID blocks are deleted.

The next step is to create and display four light buttons:

- LINE
- CRCL
- ERAS
- OVER

The first three light buttons call subroutines LINE, CIRCLE, and ERASE, respectively. OVER calls GICNRL to terminate the application.

The operator at this time sees the light buttons displayed along with the tracking cross. He responds by picking up the cross with the light pen and moves the cross to where he wants a line or circle to be displayed. His next step is to select either the LINE or CRCL light button to indicate whether he wants a line or circle to be displayed at the tracking cross position.

GIBUT has been called and is waiting for a light button pick. Once the operator has responded, the ID block of the selected light button is returned; a computed GO TO is executed, based on the IDDC parameter which indicates which particular light button was selected. It is safe to assume that either LINE or CRCL was selected since there is nothing to erase and its not likely that the operator would terminate at this time. The following paragraphs analyze the functions of subroutines LINE, CIRCLE, and ERASE which are called as a result of the execution of the computed GO TO.

SUBROUTINE LINE

LINE retrieves the horizontal and vertical coordinates of the tracking cross with a call to GITCOF. The coordinates are returned in ITH and ITV. A reset sequence is created using ITH and ITV for the position at which the line will start. This is followed with a call to GUMACG to generate the macro call which provides access to the line byte-stream in the macro area of buffer memory. Note that the macro address given in the call is the one for the line (MAD [1]).

The bytes for the reset sequence and the subroutine call are temporarily stored in IBUF. Now GIDISP is called to transfer NBYTE bytes of IBUF to the display item area of buffer memory. The display address is stored in IDDAD (K + 1) to enable the operator to erase the line if he desires. Note that the ID block contains the line type (2), the line code (1), and K + 1. K + 1 will be used in subroutine ERASE to determine which particular display item is to be erased. K is then incremented to be ready for the next item to be displayed. K is also tested for equality to 11 to see if all ten locations of IDDAD have been used. If not, the program continues. If all the locations have been used, GICNRL is called to terminate the application.

SUBROUTINE CIRCLE

CIRCLE is identical to LINE except that MAD (2) is used instead of MAD (1) and the circle code (2) is used in the call to GIDISP instead of code (1) (as was the case with LINE). Both LINE and CIRCLE could easily be combined into one routine; however, the redundancy reinforces the learning process.

SUBROUTINE ERASE

ERASE fetches the ID block of the line or circle picked by the operator for erasure. IDWA contains the K parameter set into the ID block by GIDISP when the subroutine call sequence was generated. GIERAS is then called to erase the line or circle whose display address is found in IDDAD (IDWA), where IDWA once again is the K parameter. This does not remove the byte-stream from the display item area; however, it merely removes a particular reset and macro call sequence from the macro area. A printout of these routines is shown below.

```
              OVERLAY (SCR,0,0)
              PROGRAM M(INPUT,OUTPUT)
000003        COMMON IBUF (100), MAD (2), IDDAD (10), NBYTE, MBYTE, NCON, K
000003        CALL MAIN
000004        END

              OVERLAY (1,0)



              PROGRAM CLASDEM
000003        COMMON IBUF (100), MAD (2), IDDAD (10), NBYTE, MBYTE, NCON, K
          C
          C      MAD(1)    LINE MACRO ADDRESS
          C      MAD(2)    CIRCLE MACRO ADDRESS
          C
          C      IDDAD(1) TO IDDAD(10)  DISPLAY ITEM ADDRESS
          C
          C      DISPLAY ITEM BLOCK
          C           WORD1  DISPLAY TYPE
          C                  1= BUTTON
          C                  2= SINGLE PICK
          C           WORD2  DISPLAY ITEM
          C                  1= LINE
          C                  2= CIRCLE
          C           WORD3  DISPLAY ITEM MATRIX ADDRESS
          C
          C   SIGN ON CONSOLE
000003        READ 1, NCON
000011      1 FORMAT (I2)
000011        CALL GICNJB (NCON)
000013        MBYTE = 320
000014        NBYTE = 0
000015        K = 0
          C   GENERATE LINE MACRO
000016        CALL GUSEGS (0, 0, 600, 0, 1, -0, IBUF, NBYTE, MBYTE)
000026        CALL GIMAC(NCON,IBUF,NBYTE,MAD(1))
000031        NBYTE = 0
          C   GENERATE CIRCLE MACRO
000032        CALL GUARCG (1,0,0,300, 0, 300, 0, -0, IBUF, NBYTE, MBYTE)
000045        CALL GIMAC(NCON,IBUF,NBYTE,MAD(2))
000050        NBYTE = 0
          C   SET BUTTON MASK
000051        CALL GIMASK (NCON,-0,1,16+8)
          C   SET SINGLE PICK MASK
000056        CALL GIMASK (NCON,-0,2,16+2)
          C   DISPLAY LINE BUTTON
000063        CALL GURSET (0, -1500, 102B, IBUF, NBYTE, MBYTE)
000067        CALL GUAN (4HLINE, 4, IBUF, NBYTE, MBYTE)
000073        CALL GIDISP (NCON, IBUF, NBYTE, IDA, 1, 1)
000077        NBYTE = 0
          C   DISPLAY CIRCLE BUTTON
000100        CALL GURSET (0, -1600, 102B, IBUF, NBYTE, MBYTE)
000104        CALL GUAN (4HCRCL, 4, IBUF, NBYTE, MBYTE)
000110        CALL GIDISP (NCON, IBUF, NBYTE, IDA, 1, 2)
```

```
000114           NBYTE = 0
           C     DISPLAY ERASE BUTTON
000115           CALL GURSET (0, -1700,  102B, IBUF, NBYTE, MBYTE)
000121           CALL GUAN (4HERAS, 4, IBUF, NBYTE, MBYTE)
000125           CALL GIDISP (NCON, IBUF, NBYTE, IDA, 1, 3)
000131           NBYTE = 0
           C     DISPLAY OVER BUTTON
000132           CALL GURSET (0, -1800, 102B, IBUF, NBYTE, MBYTE)
000136           CALL GUAN (4HOVER, 4, IBUF, NBYTE, MBYTE)
```

```
000142           CALL GIDISP (NCON, IBUF, NBYTE, IDA, 1, 4)
000146           NBYTE = 0
           C     TURN ON TRACKING CROSS
000147         2 CALL GITCON (NCON,0,0)
           C     WAIT TO PICK BUTTON
000152           CALL GIBUT (0,NCON,IDDT,IDDC)
000155           GO TO (3, 4, 5, 6), IDDC
000165         3 CALL LINE
000166           GO TO 2
000167         4 CALL CIRCLE
000170           GO TO 2
000171         5 CALL ERASE
000172           GO TO 2
           C     JOB DONE. RELEASE CONSOLE
000173         6 CALL GICNRL (NCON)
000175           END
```

```
                 SUBROUTINE CIRCLE
           C     DISPLAY CIRCLE
000002           DIMENSION MESS(4)
000002           DATA MESS/40H TOO MANY FIGURES. CONSOLE RELEASED      /
000002           COMMON IBUF (100), MAD (2), IDDAD (10), NBYTE, MBYTE, NCON, K
000002           NBYTE = 0
000003           CALL GITCOF (NCON, ITH, ITV)
000006           CALL GURSET (ITH, ITV, 102B, IBUF, NBYTE, MBYTE)
000012           CALL GUMACG(MAD(2),1,IBUF,NBYTE,MBYTE)
000016           CALL GIDISP (NCON, IBUF, NBYTE, IDDAD (K+1), 2, 2, K+1)
000027           K = K+1
000031           IF (K.EQ.11) 1, 2
           C     TOO MANY FIGURES. RELEASE CONSOLE
000035         1 NBYTE=0
000036           CALL GURSET (0,-1400, 2,IBUF,NBYTE,MBYTE)
000042           CALL GUAN (MESS, 35,IBUF,NBYTE, MBYTE)
000046           CALL GIDISP(NCON,IBUF,NBYTE,IDDAD)
000051           CALL GICNRL (NCON)
000053           STOP
000055         2 RETURN
000056           END
```

```
                 SUBROUTINE LINE
         C       DISPLAY LINE
000002           COMMON IBUF (100), MAD (2), IDDAD (10), NBYTE, MBYTE, NCON, K
000002           DIMENSION MESS(4)
000002           DATA MESS/40H TOO MANY FIGURES. CONSOLE RELEASED     /
000002           NBYTE = 0
000003           CALL GITCOF (NCON, ITH, ITV)
000006           CALL GURSET (ITH, ITV, 102B, IBUF, NBYTE, MBYTE)
000012           CALL GUMACG(MAD(1),1,IBUF,NBYTE,MBYTE)
000016           CALL GIDISP (NCON, IBUF, NBYTE, IDDAD (K+1), 2, 1, K+1)
000027           K = K+1
000031           IF (K.EQ.11) 1, 2
         C       TOO MANY FIGURES. RELEASE CONSOLE
000035         1 NBYTE =0
000036           CALL GURSET (0,-1400, 2,IBUF,NBYTE,MBYTE)
000042           CALL GUAN (MESS,35, IBUF, NBYTE,MBYTE)
000046           CALL GIDISP(NCON,IBUF,NBYTE,IDDAD)
000051           CALL GICNRL (NCON)
000053           STOP
000055         2 RETURN
000056           END


                 SUBROUTINE ERASE
         C       CLEAR DISPLAY ITEM
000002           COMMON IBUF (100), MAD (2), IDDAD (10), NBYTE, MBYTE, NCON, K
         C       READ DISPLAY ITEM ID BLOCK PICKED
000002           CALL GIFID (NCON, IDDT, IDDC, IDWA)
000005           CALL GIERAS (IDDAD(IDWA)   )
000010           RETURN
000011           END

         7
         8
         9
         XMPL
         SCR
         6
         7
         8
         9
```

DIFFERENCES BETWEEN 6000 BASIC GRAPHICS PACKAGE AND 3000 DIGIGRAPHICS CONTROL PACKAGE MARK 4.0

I

The differences between the graphics routines of the 6000 Series Interactive Graphics System and the 3000 Series Master Graphics System can be grouped in three categories:

- Graphics processing differences
- Hardware-produced variations
- Operating system differences

Because of these differences, a program written for one system may not run in the other unless changes are made in the coding. Although complete compatibility between the systems is a desirable feature, such compatibility cannot be achieved without serious restrictions to both systems. Consequently, a programmer converting jobs from one system to the other must bear in mind the differences listed below.

PROCESSING VARIATIONS

The subroutines discussed here exist in both systems, have identical calling sequences, and perform similar functions. The majority of incompatibilities exist in interpretation of the parameters and/or the processing of the functions.

ALPHANUMERIC FONT SIZE

The 3000 graphics system uses a standard font size requiring each character to occupy a square of 30_8 display grid units per side. The 6000 system can use either this size font or a larger one of 40_8 units per side; the choice of font size is a system assembly option, and affects the number of characters that can be displayed on one line by a GUAN call.

USE OF MINUS ZERO

The 6000 system allows various routine calling sequences to be truncated with a minus zero parameter or a right parenthesis; the 3000 system permits truncation only by a right parenthesis.

The 6000 system allows a -0 to indicate a no change option for GIMOVE and GICOPY subroutines; the 3000 system allows a -0 to indicate a no change option only for the GIMOVE subroutine.

IBUF SIZE

There are also differences between the two systems in the maximum number of bytes allowed on calls to GIDISP and GIMAC. The 3000 system returns an error indication if the number of bytes is greater than 4095, while the 6000 system has a limit of 310 bytes on calls to GIDISP and 316 bytes on calls to GIMAC.

This difference arises because the graphics consoles communicate directly with the computer in the 3000 system, but must use EXPORT/IMPORT in the 6000 system. EXPORT/IMPORT has a limit on the number of 12-bit words which can be communicated in one transmission between computers.

ATTRACT MASK

The 3000 system has an attract tracking cross mask (IMASK = 32) which causes the tracking cross to be positioned under the light-pen when an entity is picked which has the attract mask set. The 6000 system does not have this feature because the tracking cross is a software (rather than a hardware) controlled entity in systems that use the 1700 graphics hardware.

TASK PROCESSING

The 3000 system allows 4-character task names and uses only the IDWA word in the graphics buffer memory for such names; whereas, the 6000 system allows 7-character task names and can use both the IDWA and IDWB words in the graphics buffer memory for such names.

The Application Executives of the two Packages also differ; the 6000 routines perform slightly different functions than the 3000 routines, and are part of a SCOPE library subroutine called MAIN. While the Executive routines need not be used in either system, it is very difficult to write a program for the 6000 without using MAIN.

The 6000 system MAIN program loads the first task on the multi-task file as the initial application task, while the 3000 system allows the user to specify the application task to be loaded first by a control card parameter.

The 3000 system allows one task to call another task and then automatically have the called task return control to the original task following that task's CALL AETSKC statement. The 6000 implements this feature through a separate call AERTRN.

In addition, the 3000 system permits a subroutine to be called as a separate task, while the 6000 system requires a subroutine to be part of a task overlay.

These differences result from the separate multiprogramming characteristics of the 6000 Series and 3000 Series computer operating systems.

DISPLAY SUBROUTINES

Because of differences between the graphics console controllers used in the two systems, the 6000 Basic Graphics Package does not permit the use of display subroutines. The GUSUBG, GISUB, and GITSMV subroutines which are available in the 3000 system to allow use of the subroutine feature are not available in the 6000 system.

HARDWARE-PRODUCED VARIATIONS

DATA HANDLER ROUTINES

The 6000 Series computer's large word size causes differences between the systems in Data Handler operation. The Data Handlers of both systems have almost identical calling sequences, but the Data Handler routines of the 6000 Basic Graphics Package have different component codes and do not access their files in the same manner as the 3000 routines; the structures of the files are identical, except for word size.

The 6000 Data Handler also has an additional optional parameter in the CALL DMINIT statement to allow more efficient multiprogramming use of the 6000 hardware.

HARDWARE TESTING AND CONTROL

The GIBWRT, GIBRD, GIBERS, and GISTAT subroutines which are available in the 3000 system to allow both on-line hardware testing and low-level user control over the graphic system are not provided in the 6000 system. These functions are not needed or desirable in the 6000 system, because the 6000 Series computer is not directly connected to the console controllers.

CONTROLLER MEMORY SIZE

The 3000 system allows display buffer sizes from 4K to 16K, and possibly 32K. The 6000 system allows display buffer sizes from 4K to 8K because of 1700 Computer addressing limitations. Therefore, a programmer converting from the 3000 system to the 6000 system must be sure that his display generation calls do not cause controller memory overflow.

GIDISP/GIMAC ERRORS

Since the 3000 system has a channel/controller interface, the software returns a zero value in the identification parameter (IDDAD or MAD) if a graphic display buffer memory overflow occurs. The 6000 system cannot efficiently provide this capability because it does not directly interface with the console controllers.

CONTROLLER MEMORY DUMP

The GUDDMP utility subroutine in the 3000 system provides for a dump of the display buffer memory. This is not provided in the 6000 system for reasons similar to those given under GIDISP/GIMAC errors.

OPERATING SYSTEM DIFFERENCES

The 3000 computer system has multiprogramming features which are significantly different from those of the 6000 system. The two areas which produce the majority of additional subroutines in the 3000 graphics system are the 3000 Chapter Two COMMON and task processing concepts.

USE OF COMMON

The AEADDM and AECOPC subroutines for COMMON manipulations on the 3000 system are not available on the 6000 system. The AETSKC and AETSKR routines exist in both systems but are different because of the multiprogramming and Chapter Two characteristics of the 3000 Series computer. An additional difference results from the 4-character limit on task names for the 3000 system and 7-character limit for task names on the 6000 system.

TASK CONTROL

The AETSKT and AETSKW routines perform 3000 system functions which are not available on the 6000 system. The 3000 AEOFF subroutine provides for a voluntary termination of a graphic application on the 3000 system. This can be done on the 6000 system by using a STOP statement in the FORTRAN program.

TASK FILES

The random access task file building and processing is different. The 3000 system uses the AETG subroutine and the 6000 system uses the AEFILE subroutine. These task building differences are considered utility areas.

HARDCOPY FILES

The 6000 system GIPLOT subroutine provides a method for a 6000 application program to produce a file which can be put on microfilm using a 250 microfilm system. The 3000 system does not presently have this capability. The 3000 system could have this feature added in the future if 250 microfilm on the 3000 systems becomes a requirement.

6000 ROUTINES NOT IN 3000 SYSTEM

The following list contains the subroutines which are presently available on the 6000 system but not on the 3000 system:

GIPLOT	Creates hardcopy file
GITIMV	Moves item with tracking cross
GITMMV	Moves macro
SCHEDR	Transfers job from batch control point to graphics control point
MAIN	System supplied overlay processor
AEFILE	System supplied utility
AELOAD	System supplied utility
AEDUMP	System supplied utility
GFONTA	Creates alphanumeric font
GFONTN	Creates numeric font

3000 ROUTINES NOT IN 6000 SYSTEM

The list below contains subroutines which are presently available on the 3000 system but not on the 6000 system. The majority of subroutines which were not discussed previously can be considered utility routines and are not deemed necessary on the 6000 system at this time.

AEADDM	Extend Chapter 2 COMMON
AECOPC	Copy COMMON
AEID	Not called by application, a system program task processor
AEIT	Not called by application
AEOFF	Voluntary graphic termination
AESO	Not called by application
AETG	Not called by application
AETSKT	Indirect task call
AETSKW	Task wait
AEXF	Not called by application
GIBERS	Display buffer erase
GIBRD	Display buffer read
GIBWRT	Display buffer write
GISTAT	Console status
GISUB	Subroutine insert
GITSMV	Tracking subroutine move
GUCONV	BCD to floating-point conversion
GUDDMP	Display buffer dump
GUKM	Keyboard maker
GUSUBG	Subroutine call generator

CREATING ALPHANUMERIC DISPLAY FONTS J

For certain applications, the programmer may wish to provide the console user with a display font other than the two supplied in the 6000 Basic Graphics Package (see Section 7, GFONTA and GFONTN). The following discussion covers some of the more important points that a programmer should consider when creating his own display font.

FONT CHARACTER RECOGNITION

The 1700 Basic Graphics Package recognizes a sequence of display generation bytes followed by a one-word ID as a display font character. When the character is picked with the light-pen and GIANS has been called, the ID word is queued on an alphanumeric string so that it can be sent to the application program when a GIANE call occurs. Each 8-bit ID word in the 1700 is an ASCII character and is converted to 6000 display code before being sent to the 6000 application program.

Because of this processing, the application programmer can create font characters by supplying the one-word ASCII ID through a call to GUBYTE. For example, the three calls:

```
CALL GURSET(IH,IV,ICODE,IBUF,NBYTE,MBYTE)
CALL GUAN(1LA,1,IBUF,NBYTE,MBYTE)
CALL GUBYTE(101B,1,IBUF,NBYTE,MBYTE)
```

create an alphanumeric font of one character, A, at screen coordinates IH and IV. The ASCII code for A is 101_8 or 41_{16}.

The call:

```
CALL GIDISP(NCON,IBUF,NBYTE,IDDAD, -0)
```

then displays this one character font. After the font appears on the screen, the call:

```
CALL GIANS (NCON, 10, IH1, IV1)
```

creates a light register at screen coordinates IH1 and IV1; this register can contain up to 10 of the A's, if the character is picked that many times.

If the character A is picked once and GIANE is called, the parameters returned to the call will be:

```
NC   = 1
IBCD = Abbbbbbbbb
```

where the letter b indicates a blank.

SPECIAL CHARACTERS

Two special characters are defined for the 1700 Package. These two characters, backspace and clear, allow the console operator to remove characters which have been queued since the call to GIANS and before the next call to GIANE occurs.

BACKSPACE

Any display followed by a one-word ID of 137B (or $5F_{16}$) is defined as a backspace character. When such a character is picked with the light-pen, the last picked character in the light register is erased from the display and the underline is restored; the ID of the erased character is also removed from the buffer of queued alphanumeric information.

CLEAR

Any display followed by a one-word ID of 177B (or $7F_{16}$) is defined as a clear character. When such a character is picked with the light-pen, all of the characters currently in the light register are erased and the entire underline is restored; in addition, the ID's for all of the erased characters are removed from the buffer of queued alphanumeric information.

Backspace and clear have no other effect on alphanumeric picking.

RESET SEQUENCES

When a GURSET call is used in the definition of a font character, the ICODE's bit (bit 2^6) must be set. The s bit of the reset sequence is the enable light-pen bit; if it is not set, the character's ID word is not read when a pick is made, and the character consequently cannot be entered into the light register or queued for 6000 processing.

A font character can be generated without a reset sequence by using a GUAN call with NC set equal to one, but a no-operation instruction must precede the GUAN call in the character's IBUF. This no-op may be supplied by a GUBYTE call of one byte, where the byte is a positive zero value.

CONSERVING ID WORD SPACE

The ID words IDDT, IDDC, IDWA, and IDWB of the GIDISP call or calls which display font characters need not be referenced; the 1744 buffer space they normally occupy can be conserved by truncating the parameter list with a closing or right parenthesis after IDDAD.

DYNAMIC ADDITION OF CHARACTERS

Characters may be added to an existing console display font by successive calls to GIDISP at any time; duplicates of the same character, i.e., characters with the same ASCII code ID words, may be present in a font.

SAMPLE FONT CREATION ROUTINES

The following subroutine creates a display font containing

    0 1 2 3 4 5 6 7 8 9 X

```
      SUBROUTINE NFONT (NCON,IBUF,NBYTE,MBYTE,IDDAD)
      DIMENSION IBCD (10)
      DATA (IBCD(I),I=1,10)/1L0,1L1,1L2,1L3,1L4,1L5,1L6,1L7,1L8,1L9/
      CALL GURSET (0,-600,103B,IBUF,NBYTE,310)
      ICON1 = 60B
      ICON2 = 71B
      DO 1 I = ICON1,ICON2
      J = I -57B
      CALL GUBYTE(0,1,IBUF,NBYTE,MBYTE)
C     THE PRECEDING CALL PROVIDES A NO-OP BEFORE EACH GUAN CALL TO
C     GENERATE A CHARACTER AND IS NECESSARY ONLY WHEN EACH CHARACTER
C     IS GENERATED BY A SEPARATE GUAN CALL
      CALL GUAN (IBCD(J), 1, IBUF, NBYTE, MBYTE)
C     THE PRECEDING CALL GENERATES ONE OF THE FONT CHARACTERS
      CALL GUBYTE (I, 1, IBUF, NBYTE, MBYTE)
C     THE FOLLOWING CALL PROVIDES SPACING BETWEEN CHARACTERS AND
C     COULD BE REPLACED BY A GURSET CALL
    1 CALL GUAN (1L , 1, IBUF, NBYTE, MBYTE)
      CALL GUAN (2L   , 2, IBUF, NBYTE, MBYTE)
      CALL GUBYTE (0, 1, IBUF, NBYTE, MBYTE)
      CALL GUAN (1LX, 1, IBUF, NBYTE, MBYTE)
      CALL GUBYTE (130B, 1, IBUF, NBYTE, MBYTE)
C     THE THREE PRECEDING CALLS CREATE AND IDENTIFY THE CHARACTER X
C     AS AN END-OF-MESSAGE CHARACTER FOR USE IN GIEOM ASSIGNMENT
C     THE FOLLOWING CALL DISPLAYS THE FONT
      CALL GIDISP (NCON, IBUF, NBYTE, IDDAD, -0)
      RETURN
      END
```

The programmer can also create display font characters of any size he wishes; he need not use the size characters that are defined by the 1700 Basic Graphics Package alphanumeric macros. For example, the three following calls create a circle with a center at IHC and IVC, and an initial/termination point at IH and IV. This circle is queued as an alphanumeric 0 when picked with the light-pen.

```
      CALL GURSET (IH,IV,ICODE,IBUF,NBYTE,MBYTE)
      CALL GUARCG (1,IHC,IVC,IH,IV,IH,IV,IBUF,NBYTE,MBYTE)
      CALL GUBYTE (117B,1,IBUF,NBYTE,MBYTE)
```

Note that the ASCII code equivalent of 0 is 117B ($4F_{16}$).

The programmer can create a true/false font with coding like the following:

```
      CALL GURSET (IH1, IV1, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUAN (4HTRUE, 4, IBUF, NBYTE, MBYTE)
      CALL GUBYTE (124B, 1, IBUF, NBYTE, MBYTE)

C     THE PRECEDING CALLS CREATE THE WORD TRUE BEGINNING AT IH1/IV1
C     AND QUEUE AN ALPHANUMERIC T (=124B) WHEN IT IS PICKED
      CALL GURSET (IH2, IV2, ICODE, IBUF, NBYTE, MBYTE)
      CALL GUAN (5HFALSE, 5, IBUF, NBYTE, MBYTE)
      CALL GUBYTE (106B, 1, IBUF, NBYTE, MBYTE)

C     THE PRECEDING 3 CALLS CREATE THE WORD FALSE BEGINNING AT IH2/IV2
C     AND QUEUE AN ALPHANUMERIC F (=106B) WHEN IT IS PICKED
```

HEXADECIMAL/OCTAL CONVERSION TABLE K

Hexadecimal	Octal	Hexadecimal	Octal	Hexadecimal	Octal
8	10	5B	133	AE	256
9	11	5C	134	AF	257
A	12	5D	135	B0	260
B	13	5E	136		
C	14	5F	137	B8	270
D	15	60	140	B9	271
E	16			BA	272
F	17	68	150	BB	273
10	20	69	151	BC	274
		6A	152	BD	275
18	30	6B	153	BE	276
19	31	6C	154	BF	277
1A	32	6D	155	C0	300
1B	33	6E	156		
1C	34	6F	157	C8	310
1D	35	70	160	C9	311
1E	36			CA	312
1F	37	78	170	CB	313
20	40	79	171	CC	314
		7A	172	CD	315
28	50	7B	173	CE	316
29	51	7C	174	CF	317
2A	52	7D	175	D0	320
2B	53	7E	176		
2C	54	7F	177	D8	330
2D	55	80	200	D9	331
2E	56			DA	332
2F	57	88	210	DB	333
30	60	89	211	DC	334
		8A	212	DD	335
38	70	8B	213	DE	336
39	71	8C	214	DF	337
3A	72	8D	215	E0	340
3B	73	8E	216		
3C	74	8F	217	E8	350
3D	75	90	220	E9	351
3E	76			EA	352
3F	77	98	230	EB	353
40	100	99	231	EC	354
		9A	232	ED	355
48	110	9B	233	EE	356
49	111	9C	234	EF	357
4A	112	9D	235	F0	360
4B	113	9E	236		
4C	114	9F	237	F8	370
4D	115	A0	240	F9	371
4E	116			FA	372
4F	117	A8	250	FB	373
50	120	A9	251	FC	374
		AA	252	FD	375
58	130	AB	253	FE	376
59	131	AC	254	FF	377
5A	132	AD	255		

RE-ENTERING A GRAPHICS TASK OVERLAY L

A graphics task overlay consists of a FORTRAN program and its associated subroutines in absolute format. Each task is entered by an unconditional jump to the entry address of the overlay, and normally no provision is made to return to the statement following the task call.

AERTRN

Under certain circumstances, the programmer may wish to return from a graphics task to the statement following the CALL AETSKC card which caused entry to the task; he might do this if he wanted to call several tasks in a row. A routine called AERTRN is provided for this purpose.

When the programmer wants to return from one task to another, he places a card with the format:

```
CALL AERTRN
```

in the task he wishes to return from. When this card is encountered, control is passed to the return address of the last executed return jump to AETSKC.

Note that AERTRN does not provide for reloading the task that called AETSKC; it provides only the jump to pass control and the record of the last call to AETSKC. The programmer must insure that the tasks do not overload each other, and that the AERTRN call occurs whenever the return feature is desired.

EXAMPLES

The following examples show how secondary overlays and the C parameter on the overlay card may be used to set up a task file so that AERTRN can be used.

C PARAMETER

The standard FORTRAN overlay card has the format:

```
OVERLAY (lfn, p, s, Cnnnnnn)
```

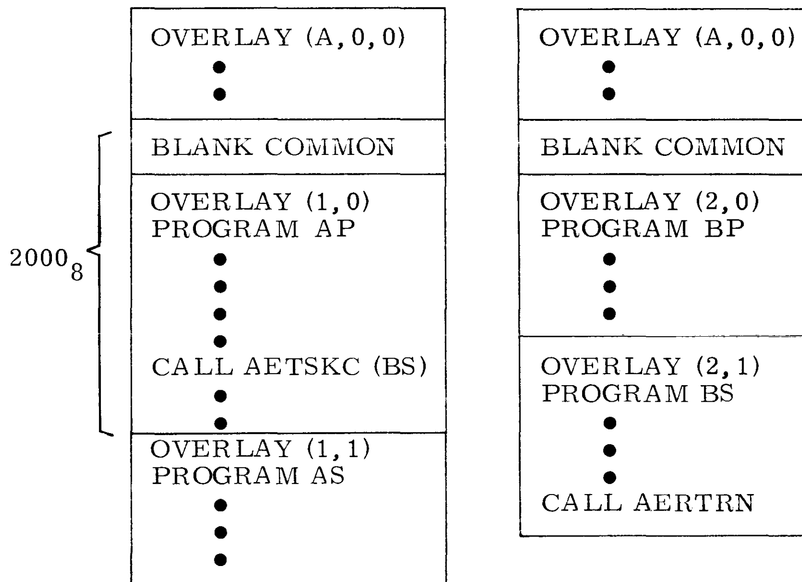
where lfn, p, and s have the meanings given in the overlay card definition of Section 2. The quantity nnnnnn after the letter C in the parameter field is an octal value that specifies the first word address of the overlay with respect to the beginning of blank COMMON; i.e., the

overlay coding is loaded and entered at a location nnnnnn words after the beginning of the program's blank COMMON area. This C parameter cannot be used on the zero-level overlay card, but is optional on all other overlay cards.

Since the first word address of blank COMMON is constant for any given overlay or task file, all overlays with the same C parameter will have the same first word address in core.

For example, assume that AP and BP are primary overlays, and AS and BS are secondary overlays; all four have been written into a task file by AEFILE.

These four overlays would appear in core as:



Program AS has been relocated with respect to the last word address plus one of program AP because they have the same primary level number. Program BS has been relocated with respect to the last word address plus one of program BP because they also have the same primary level number.

OVERLOD, the standard SCOPE overlay loader, will not allow overlays (1,0) or (1,1) to call overlay (2,1). Primary overlays and overlays with the same primary number may call each other; no other calls are allowed. However, a call to AETSKC allows any of the four overlays to call any of the others by using their program name.

If a programmer places a task call to BS from AP, part of the called task will be loaded over the calling task. If a call to AERTRN is then made at the end of BS, AERTRN will return control to the core address following the last AETSKC call, but the return will have chaotic results because the core locations that contained the code which the programmer wished to execute have been overlayed by the beginning of program BS. The following paragraphs describe one method of avoiding this problem.

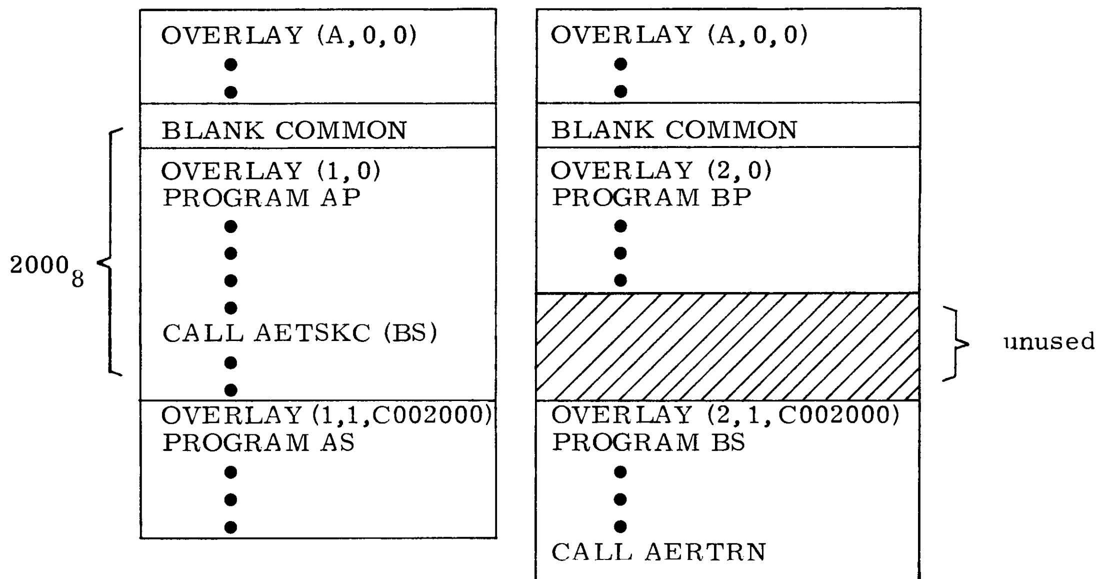
Assume that overlay (1, 1) is loaded 2000_8 words from the first word address of blank COMMON. If the secondary overlay cards are written with the C parameter so that they appear as:

OVERLAY (1, 1, C002000)

and

OVERLAY (2, 1, C002000)

the routines in the overlays will have the same first word address and will appear in core as:



Now each of the programs is free to call the others and to use a CALL AERTRN card to return to the address following the last call to AETSKC.

It is up to the programmer to keep track of the overlay core relationships when using AERTRN. A task which calls another with the expectation of returning should be located so that the two do not overlay each other. AERTRN provides limited capability for constructing tasks which may be called by AETSKR and also entered as subroutines. If logic requires the use of AERTRN in some cases and AETSKC or AETSKR in others, a flag may be set in blank COMMON by the calling task and interrogated before each return is executed.

An error message, RETURN ADDRESS OVERLAYED, will be sent to the dayfile and a task return will be executed only if the return address of AERTRN is within the overlay calling AERTRN.

Note that linkage of external symbols is not provided for by the GPSL Loader between overlays with different primary level overlay numbers. If overlays 2.1 and 2.0 have subroutine linkages in common, the overlay 2.1 will probably not run correctly unless 2.0 is in core at the same time. AERTRN should primarily be used in secondary overlays with the same primary number while the primary is in core.

SYSTEM PACKING OF IBUF DESCRIPTION BUFFERS M

Nine Graphics Utilities routines of the 6000 Basic Graphics Package place item description bytes in IBUF; in addition, both GIDISP and GIMAC place header and trailer bytes into the description buffer before sending it to the 1700 Buffer Translator through EXPORT.

Table M-1 lists all of the routines that place bytes into IBUF and gives the number of bytes packed by each; all 12-bit bytes are packed five to a 60-bit central memory word, starting in byte zero.

TABLE M-1.
IBUF/1744 BYTE COMPARISON, ITEM DESCRIPTION BYTE GENERATORS

Routine	(Octal) Call Code	Number of Bytes Packed	Explanation
GUAN	02	$1 + \frac{NC + 1}{2}$	NC is the character number parameter in the call
GUARCG	06	6 + 4 * KSHOW	KSHOW is the arc segment number parameter in the call
GUBYTE	08	1 + L	L is the byte number parameter in the call
GUMACG	07	1 + 2 * L	L is the macro address number parameter in the call
GURSET	01	3 (4 Bits)	
GUSEG	04	3	
GUSEG	05	2 + 3 * (N + 1) (4 Bits)	N is the line segment number parameter in the call
GUSEGI	11	4	
GUSEGS	03	6	
GIDISP	01	10 (6 Bits)	Eight trailer and two header bytes; explained below
GIMAC	05	4 (6 Bits)	Two header and two trailer bytes; explained below

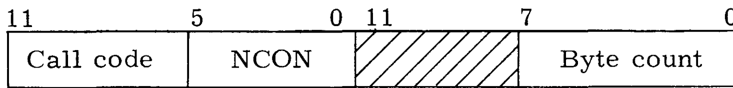
Each Graphics Utilities call packs a call code for the corresponding 1700 Basic Graphics Package routine into the upper four bits of the first 12-bit byte it places in IBUF; if a Graphics Utilities routine is called with NBYTE equal to zero, the routine will leave two

bytes empty at the beginning of the next unfilled central memory word in IBUF. The two empty bytes are usually used by GIDISP or GIMAC for the two header bytes which each packs in IBUF.

NOTE

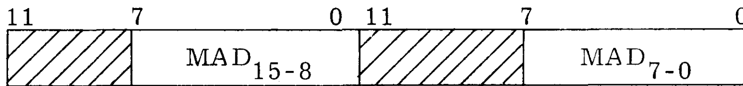
The call codes of all GU routines are in the upper 4 bits of a 12-bit byte and are within a GIDISP, GIMAC, or GIPLOT buffer. The call codes of all GI routines are in the upper 6 bits of a 12-bit byte at the beginning of a 60-bit central memory word.

These bytes have the structure:



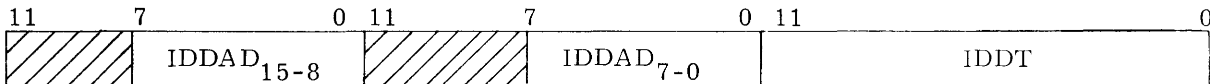
where the byte count excludes the header and trailer bytes.

The two trailer bytes packed by GIMAC are placed in IBUF immediately after the last item description byte. The first of these two bytes contains bits 15 through 8 of the lower 16 bits of MAD, right-justified; the second byte contains bits 7 through 0 of MAD, also right-justified, as shown:

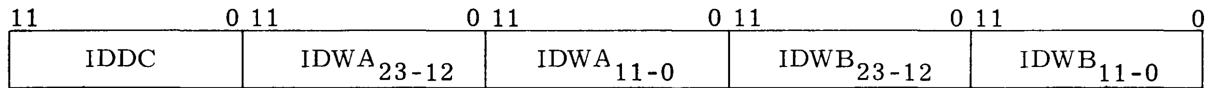


GIDISP places a variable number of trailer bytes (three to eight) in IBUF, immediately following the last packed item description byte. The number packed depends on the number of parameters present in the GIDISP calling sequence before a minus zero parameter or a right parenthesis is encountered.

The three trailer bytes always packed by GIDISP contain bits 15 through 8 of IDDAD right-justified in the first, bits 7 through 0 of IDDAD right-justified in the second, and IDDT in the third, as follows:



In a full calling sequence, five more bytes would be packed. These would contain: IDDC in the first additional byte, bits 23 through 12 of IDWA in the second, bits 11 through 0 of IDWA in the third, bits 23 through 12 of IDWB in the fourth, and bits 11 through 0 of IDWB in the fifth. These bytes appear as follows:



11 — 0	11 — 0	11 — 0	11 — 0	11 — 0
IDDC	$IDWA_{23-12}$	$IDWA_{11-0}$	$IDWB_{23-12}$	$IDWB_{11-0}$

GIDISP terminates packing with the first minus zero parameter. If a right parenthesis terminates the calling sequence before IDWB, the first missing parameter is replaced by a minus zero and packing is terminated.

Both GIMAC and GIDISP issue a fatal error diagnostic if the IBUF description buffer resulting from the call is longer than 64 central memory words (including header and trailer bytes).

Both routines process a non-fatal error and issue an informative dayfile message if the NBYTE parameter is equal to zero when the routine is called.

OMISSION OF MAIN FROM PROGRAM CODING N

Although the procedure is not recommended, the Application Executive MAIN program can be omitted from a graphics program. If the programmer does not use MAIN, he must either provide substitutes for each of its routines or else resign himself to the use of an inordinately large amount of central memory by his job. This appendix provides an outline of the structure and functions of MAIN so that a programmer can write replacement routines if he wishes.

STRUCTURE OF MAIN

The first 74_8 words in MAIN are entry points and buffer areas shared by the 6000 Basic Graphics Package routines (see Figure N-1).

MAIN has seven formal entry point names:

AERTRN
AETSKC
AETSKR
BGP
DATBUF
MAIN
RIDDAD

The entry point MAIN is called only once during a job and is not referenced by any of the routines in the 6000 Basic Graphics Package. MAIN is the entry point used by the CALL MAIN card in the program's zero-level overlay and provides access to coding that provides AEFILE with the file names it uses during a task file creation run; the coding associated with entry point MAIN also initiates loading and execution of the first overlay in the task file during an execution run.

The entry points AETSKR, AERTRN, and AETSKC are used by the respective 6000 Basic Graphics Package routines. Both entry points provide unconditional jumps to appropriate subroutines within the MAIN program.

The entry points BGP, DATBUF, and RIDDAD are used by the 6000 Basic Graphics Package routines to share access to common pointers.

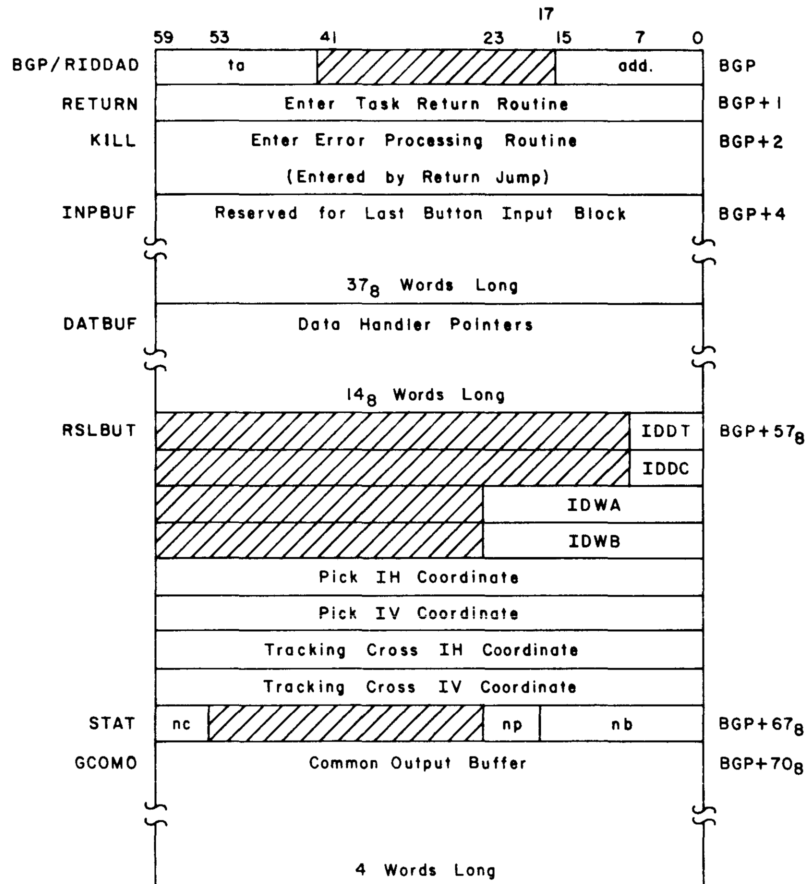


Figure N-1. MAIN Communications Area

The last four entry points described above are all linked to the presence of MAIN in the program's zero-level overlay. If MAIN is not called there, and the application program contains no subroutines with entry points named BGP, RIDDAD and DATBUF, then the entire MAIN program will be loaded from the system library into every overlay that contains Package routines which reference any of these entry points.

If the application program does contain a subroutine (in place of MAIN) with the proper entry points, all 6000 Package routine references to the points are linked to the locations listed in Table N-1.

TABLE N-1. 6000 PACKAGE EXTERNAL LINKAGES

Entry Point Name	Purpose	Type of Reference
FORTRAN Callable:		
AETSKC	Load an overlay by name	Relocatable
AERTRN	Return from task to calling address plus one	Relocatable
MAIN	Initialize Application Executive	Relocatable
Not FORTRAN Callable:		
BGP	Used to reference first 74_8 words of MAIN relatively	Relocatable
RETURN	Perform task return (AETSKR)	Relative to BGP
KILL	Process errors and messages	Relative to BGP
INPBUF	Reserve last button input block	Relative to BGP
RIDDAD	Reserve task name address and IDDAD/MAD counter	Relative to BGP
RSLBUT	Reserve last button ID parameters	Relative to BGP
STAT	Reserve NCON and single/string pick counters	Relative to BGP
GCOMO	Common EXPORT output buffer	Relative to BGP

CODING EXAMPLES O

This appendix contains examples of code for creating such things as light buttons, lines, circles, arcs, figures, etc. For the most part, assume that dimensioning has been performed; special cases will include dimension statements.

CONSOLE TO JOB ASSIGNMENT

```
   READ 1, NCON
 1 FORMAT (O2)
   CALL GICNJB (NCON)
```

A data card is read from the card reader and the octal contents of the card (the console number) is placed in NCON. GICNJB is called to make the console available to the calling job.

CREATE A LIGHT BUTTON

```
   NAME = 8HDISPLAY
   CALL GURSET (-1500,0,102B,IBUF,NBYTE,MBYTE)
   CALL GUAN (NAME,8,IBUF,NBYTE,MBYTE)
   CALL GIDISP (NCON,IBUF,NBYTE,IDDAD,1,1)
   NBYTE = 0
```

The characters to be displayed are DISPLAY. GURSET is called to generate a reset sequence byte-stream which is placed in temporary user buffer IBUF. GUAN is called to generate the byte-stream which will cause the alphanumerics to be displayed. This byte-stream follows the reset sequence byte-stream in IBUF. Parameter NBYTE is automatically updated to reflect the number of bytes in IBUF.

GIDISP is called to send a copy of the contents (NBYTE bytes) of IBUF to the 1700 Buffer Translator. NBYTE is then set back to zero by the application program to initialize IBUF for the next byte-stream. The Buffer Translator calls the 1700 graphics BGP which generates a display byte-stream and places it in the display controller. Once the byte-stream is in the display buffer,it is displayed. Note that parameters IDDT and IDDC in the call to GIDISP are both one. For IDDT = 1 to be meaningful to the system (i.e., when the button is picked with the light pen by the operator, the system interprets IDDT for the action it is to perform) the following call must be executed prior to the pick of the button:

```
   CALL GIMASK (NCON,0,1, 16+8)
```

This call defines items with the IDDT designation of 1 as buttons and the buttons so designated will have the marker mask set such that when the button is picked, it will blink (assuming that the button is not already blinking).

There are several means by which alphanumeric input can be provided. The above example uses a dimensioned array. A data statement could be used instead, as in this example:

```
COMMON/DATA/NAME (2)
DATA NAME (I,I = 1,2)/8HDISPLAY, 6HBUFFER)
```

or an ENCODE and FORMAT statement used as follows:

```
ENCODE (16,1,NAME)
FORMAT (16HDISPLAY    BUFFER)
```

CREATE A LIGHT BUTTON — UNDER APPLICATION EXECUTIVE

```
NAME = 8HDISPLAY
NAME (2) = 4RDISP
CALL GURSET (-1500,0,102B,IBUF,NBYTE,MBYTE)
CALL GUAN (NAME,8,IBUF,NBYTE,MBYTE)
CALL GIDISP (NCON,IBUF,NBYTE,IDDAD,1,1,NAME(2))
NBYTE = 0
```

Note that the calls are essentially the same as those in the next example.

GENERATE AND DISPLAY A LINE — NOT FRAME-SCISSORED

The only difference is that in this call to GIDISP, an additional parameter is used to identify the task to be called by the application executive when the button is picked. To simplify matters, the first four characters of the light button name are used as the task name; thus, if button DISPLAY is picked, the application executive brings task DISP into core from mass storage for execution.

```
CALL GURSET (-1000,0,102B,IBUF,NBYTE,MBYTE)
CALL GUSEGS (-1000,0,1000,0,1,0,IBUF,NBYTE,MBYTE)
CALL GIDSP (NCON,IBUF,NBYTE,IDDAD,2,1,0,0)
NBYTE = 0
```

These calls will display a line from -1000 H, 0 V to 1000 H, 0 V. The line is solid, light pen sensitive, and does not blink.

A previous call to GIMASK was made for type 2 as follows:

```
CALL GIMASK (NCON,0,2, 16+4)
```

This call associates the marker mask with type 2 and makes all type 2 display items string pick items.

Note that all ID block parameters are used.

One might like to store things such as the item display address and/or bead address in the ID block. For instance, the display address for the line is in IDDAD. GIMOVE can be used to insert this display address into the ID block as follows:

```
CALL GIMOVE (-0, -0, -0, IDDAD, 2, 1, 0, IDDAD)
```

The display address is now in the IDWB word of the ID block for that line. The display address can be extracted from the block by GIFSID, any time the line is picked. It is important to have access to the display addresses of the items if they are to be erased, copied, or have their ID blocks modified (as was the case above in the call to GIMOVE).

If the Data Handler is used, and a bead is formed for this line, the bead address could be placed in IDWA; it is usually handier to put the display address in the bead rather than in the ID block, but these decisions are up to the individual.

GENERATE AND DISPLAY A FRAME-SCISSORED LINE

```
CALL GULINE (0, 0, 1000, 1000, 500., 0., -600., -500., KSHOW, IHS, IVS, IHF, IVF)
CALL GURSET (IHS, IVS, 102B, IBUF, NBYTE, MBYTE)
CALL GUSEGS (IHS, IVS, IHF, IVF, 1, 0, IBUF, NBYTE, MBYTE)
CALL GIDISP (NCON, IBUF, NBYTE, IDISPAD, 2, 2, 0, 0)
NBYTE = 0
```

GULINE is called to set up the size and position of the frame for frame scissoring. Floating-point coordinates for the beginning and end of the line are given in the call and converted by the system to fixed-point coordinates. Although not shown in this example, it is a good idea to test KSHOW for equality to zero to see if the described line can be displayed. If KSHOW=0, there is no reason to call GURSET, GUSEGS, and GIDISP.

The following code will perform the check and skip to statement 2 if the line cannot be displayed:

```
    CALL GULINE (0, 0, 1000, 1000, 500., 0., -600., -500., KSHOW, IHS, IVS, IHF, IVF)
    IF (KSHOW .EQ. 0) GO TO 2
    CALL GURSET (IHS, IVS, 102B, IBUF, NBYTE, MBYTE)
    CALL GUSEGS (IHS, IVS, IHF, IVF, 1, 0, IBUF, NBYTE, MBYTE)
    CALL GIDISP (NCON, IBUF, NBYTE, IDISPAD, 2, 2, 0, 0)
    NBYTE = 0
  2 CONTINUE
```

GENERATE AND DISPLAY A CIRCLE — NOT FRAME-SCISSORED

```
      CALL GURSET (0,0,102B,IBUF,NBYTE,MBYTE)
      CALL GUARCG (1,0,0, 300,0,300,0,-0,IBUF,NBYTE,MBYTE)
      CALL GIDISP (NCON,IBUF,NBYTE,ICRCDSPD,2,3,0,0)
      NBYTE = 0
```

This code generates a circle with solid line style; the origin of the circle is displaced from the reset coordinates by 300 display grid units. (There are 200 such units per inch in the scope coordinate system.) This is so because circles and arcs are displayed in a counter-clockwise manner from the initial point to the terminal point. Thus, the center of this circle is at -300,0.

The reader should note that parameter KSHOW is set to one, since all of the circle can be displayed on the display surface; the arc scissoring routine GUARC was not used to make that determination. An example of arc scissoring is given in the next paragraph.

GENERATE AND DISPLAY A FRAME -SCISSORED CIRCLE

```
      DIMENSION IHS(5),IVS(5),IHF(5),IVF(5)
      CALL GUARC (0,0,1000,1000,0.,0.,1125.,0.,1125.,0.,KSHOW,IHC,IVC,IHS,IVS,
      IHF,IVF)
      IF (KSHOW .EQ. 0) GO TO 3
      CALL GURSET (IHS,IVS,102B,IBUF,NBYTE,MBYTE)
      CALL GUARCG (KSHOW,IHC,IVC,IHS,IVS,IHF,IVF,-0,IBUF,NBYTE,MBYTE)
      CALL GIDISP (NCON,IBUF,NBYTE,ICRCAD,2,4,0,0)
      NBYTE = 0
    3 CONTINUE
```

Since it is possible to have a circle scissored into four segments and an arc into five segments, arrays had to be dimensioned to accept the starting and ending points of the arc segments. The coordinates used in the call to GUARC are deliberately chosen to generate four arc segments for the subject circle. As a formality, KSHOW is examined here for equality to zero, since it is known in advance that four arc segments will be displayed; however, this is not very often the case and the test for KSHOW should be made as a matter of good programming practice.

The remainder of the code merely resets the CRT beam to the start of the first arc segment. GUARCG is called to generate the four arc segments, and GIDISP transfers the byte-stream from IBUF to the display buffer.

GENERATE AND DISPLAY A 2-INCH SQUARE — NOT FRAME-SCISSORED

```
CALL GURSET (0,0,102B,IBUF,NBYTE,MBYTE)
CALL GUSEGS (0,0,400,0,1,0,IBUF,NBYTE,MBYTE)
CALL GUSEG (400,-400,1)
CALL GUSEG (0,-400,1)
CALL GUSEG (0,0,1)
CALL GIDISP (NCON,IBUF,NBYTE,ISQDSPAD,2,6,0,0)
NBYTE = 0
```

This square starts at the origin of the scope display grid. The first line segment goes right 2 inches, the second segment goes down 2 inches, the third goes left 2 inches, and the last is drawn up 2 inches to complete the square.

GENERATE AND DISPLAY A COLUMN OF FIVE HORIZONTAL LINES

```
CALL GURSET (0,0,102B,IBUF,NBYTE,MBYTE)
CALL GUSEGS (0,0,400,0,1,-0,IBUF,NBYTE,MBYTE)
CALL GUSEG (0,-100,0)
CALL GUSEG (400,-100,1)
CALL GUSEG (0,-200,0)
CALL GUSEG (400,-200,1)
CALL GUSEG (0,-300,0)
CALL GUSEG (400,-300,1)
CALL GUSEG (0,-400,0)
CALL GUSEG (400,-400,1)
CALL GIDISP (NCON,IBUF,NBYTE,ILNDSPAD,2,7,0,0)
NBYTE = 0
```

The first line is drawn from the scope display grid origin to a point 2 inches to the right. The first call to GUSEG positions the beam for the next line. Note that the GUSEG call turns the beam off and thus nothing is displayed. The second call to GUSEG generates the byte-stream for the second of the five lines to be displayed. From this point it is a repetition of the first two calls to GUSEG until the entire byte-stream is generated. GIDISP is then called to transfer the byte-stream to the display buffer for subsequent display.

An alternate method for generating the same five lines uses a call to GUSEGA, as follows:

```
DIMENSION IH(10),IV(10),IBEAM(10)
IH(1)=IH(3)=(IH(5)=IH(7)=IH(9)=IV(1)=IV(2)=0
IH(2)=IH(4)=IH(6)=IH(8)=IH(10)=400
IV(3)=IV(4)=-100
(V(5)=IV(6)=-200
```

```
      IV(7)=IV(8)=-300
      IV(9)=IV(10)=-400
      IBEAM(1)=IBEAM(3)=IBEAM(5)=IBEAM(7)=IBEAM(9)=1
      IBEAM(2)=IBEAM(4)=IBEAM(6)=IBEAM(8)=IBEAM(10)=0
      CALL GURSET (IH,IV,102B,IBUF,NBYTE,MBYTE)
      CALL GUSEGA (IH,IV,IBEAM,9,-0,IBUF,NBYTE,MBYTE)
      CALL GIDISP (NCON,IBUF,NBYTE,ILNDSPAD,2,7,0,0)
      NBYTE = 0
```

In this example the single call to GUSEGA generates the byte-streams for all the line segments by referring to the IH, IV and IBEAM arrays.

GENERATE AND DISPLAY A LINE AS A MACRO

```
      CALL GUSEGS (0,0,400,500,1,-0,IBUF,NBYTE,MBYTE)
      CALL GIMAC (NCON,IBUF,NBYTE,MAD)
      NBYTE = 0
```

The above code generates the byte-stream for the line, transfers a copy of the byte-stream to the fixed address area of the display buffer, and returns a macro address in MAD. The line is not displayed at this time. To display the line, the following code is required:

```
      CALL GURSET (100,200,102B,IBUF,NBYTE,MBYTE)
      CALL GUMACG (MAD,1,IBUF,NBYTE,MBYTE)
      CALL GIDISP (NCON,IBUF,NBYTE,MACDSPAD,2,8,0,0)
      NBYTE = 0
```

The call to GURSET determines where the macro is displayed.

DISPLAY AN ID BLOCK RETURNED FROM A CALL TO GIFSID

GIFSID is used to return the ID blocks of string pick items in the FETCH queue. This example shows the use of GIFSID in conjunction with calls to display the contents of the ID block. Assume that the proper calls to GIMASK have been made and the console operator has picked at least one string pick item and a button.

```
      DIMENSION IBCD (20)
      N = 1
      CALL GIFSID (NCON,1,IT,IC,IA,IB,IH,IV)
      CALL GURSET (-1200,1300,102B,IBUF,NBYTE,MBYTE)
      ENCODE (49,30,IBCD) NCON,IT,IC,IA,IB,IH,IV
   30 FORMAT (7HGIFSID(3(I5,1H,),2(1X,R4,1H,),I5,1H,I5,1H))
      CALL GUAN (IBCD,49,IBUF,NBYTE,MBYTE)
```

```
CALL GIDISP (NCON,IBUF,NBYTE,IDMESS,2,0,0,0)
NBYTE = 0
```

GIFSID is called to extract one string pick ID block and return it to IT,IC,IA,IH, and IV. GURSET is used to set the point at which the A/N display starts. The ENCODE and FORMAT statements assemble the indicated characters into array IBCD, which GUAN uses as input to generate the byte-stream for the display. GIDISP transfers the byte-stream to the display buffer and appends the indicated ID block to the byte-stream. This display consists of the call to GIFSID with its calling and result parameters.

MOVE A DISPLAY ITEM

Assume that a line has been displayed as a display item and has its display address stored in IDSPAD(12). Further, it is required that the line be displayed at new coordinates IH = 1000, IV = -400. It is coded as follows:

```
CALL GIMOVE (1000,-400,-0,IDSPAD(12) )
```

ICODE as -0 indicates that the ICODE already associated with the line is not to be changed. The call is truncated after IDSPAD(12), since the ID block for the line is not to be changed either.

The line could have been moved and had its ID block changed as well by including those ID parameters to replace the existing ones. Remember that the ID block cannot be expanded beyond the size already in existence for the line. For instance, if only parameters IDDT and IDDC were used in the call to GIDISP for this line, GIMOVE cannot be used to add ID parameters IDWA and/or IDWB since space has not been allocated for these parameters.

COPY A DISPLAY ITEM

This is similar to using GIMOVE except that a copy is moved and the original still exists. To copy the line described in the last example,use the following code:

```
CALL GICOPY (IDSPAD(12),NCON,1000,400,106B,ICPYADD,IT,IC,IA,IB)
```

This will cause a copy of the line (with display address IDSPAD(12)), to be displayed four inches higher on the scope, and the copy will blink. The display address of the copy will be put in ICPYADD.

ERASE A DISPLAY ITEM

To erase the line made as a result of the call to GICOPY that was just described, use the following code:

```
CALL GIERAS(ICPYADD)
```

ERASE A MACRO

As a review, a macro is displayed in two steps:

1. Generate the byte-stream and call GIMAC to put the byte-stream in the fixed-address area of the display buffer. The address of the macro is returned in an output parameter referred to as MAD.

2. Generate a reset sequence with a call to GURSET; call GUMACG giving the MAD parameter. GUMACG will generate a calling sequence for that macro. Call GIDISP to transfer the byte-stream generated in this second step to the display buffer and the item is displayed.

The object now is to erase the macro from both the fixed- and floating-address areas of the display buffer. The procedure is inflexible because the calling sequence in the floating-address area must be erased first, then the display item byte-stream from the fixed-address area. If the calling sequence remains while the macro is erased, the display jumps to a non-existent macro and the result is chaos. The following code will correctly erase a macro:

```
CALL GIERAS (ISBDSPAD)
CALL GIMACE (MADDSPAD)
```

ISBDSPAD	Display address in the floating-address area for the calling sequence.
MADDSPAD	Macro address in the fixed-address area.

To erase a specific display of a macro and still retain its byte-stream in the final address area, call GIERAS and not GIMACE.

COMMENT SHEET

MANUAL TITLE ______________________________

PUBLICATION NO. ______________ **REVISION** ______________

FROM· NAME: ______________________________

BUSINESS
ADDRESS: ______________________________

COMMENTS:

This form is not intended for use as an order blank. Your evaluation of this manual is welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be noted below. Please include page number references and fill in the publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer Engineers are urged to use the TAR.

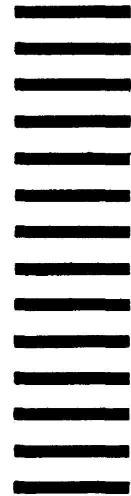
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.



BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

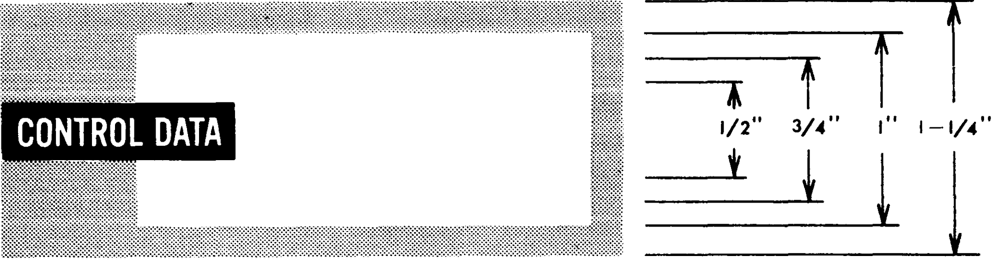
POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Systems Publications
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ON THIS LINE

FOLD FOLD

STAPLE STAPLE



➤➤ CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

SERIES INTERACTIVE GRAPHICS SYSTEM REFERENCE MANUAL

Pub. No. 44616800

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

PRINTED IN U.S.A.