RETURN TO
K. KLIMAS

SANDERS.

GRAPHIC 7 MONITOR

PRELIMINARY USER'S GUIDE

5'th Printing, May 1979 COPY____

GRAPHIC 7 MONITOR
PRELIMINARY USER'S GUIDE

TABLE OF CONTENTS

GRAPHIC 7 MONITOR

1.0 INTRODUCTION

The Graphic 7 Monitor provides support for multi-task application problems within the Graphic 7 Display Processor. The monitor supplies most of those functions commonly associated with a real-time system while emphasizing facilities associated with display creation and management. Figure 1.0-1 presents the overall monitor structure.

Fundamental to any monitor capability is the management of system resources, with memory being perhaps the most important. In the Graphic 7 monitor, memory must be shared among user tasks, refresh code and the monitor itself. In addition, the utilization of the memory management hardware is left to the monitor. This entails initialization of the mapping hardware prior to passing control to a user task.

Within the Graphic 7 monitor, user tasks represent demands for various system resources. As in most real-time systems, each task is assigned a priority which reflects its need for CPU time in relation to other tasks. Once in execution, tasks may issue requests for monitor services in the areas of I/O, display management, timing, etc. These monitor services provide many functions which would normally be left to the host machine or application programmer.

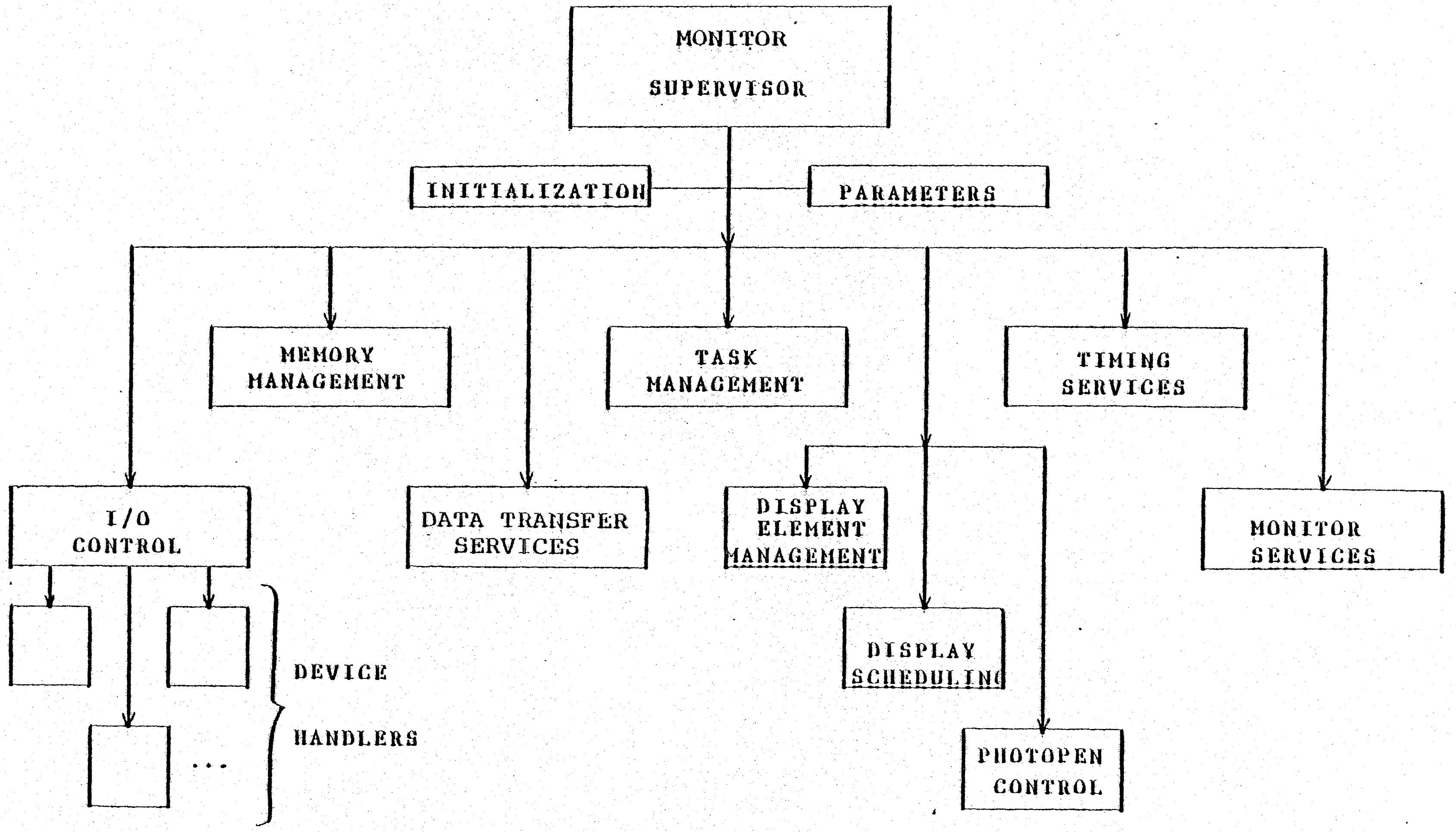


Figure 1.0-1 GRAPHIC 7 MONITOR FUNCTIONAL AREAS

Of these services, the Display Management area represents an important advance over traditional implementations. Within Display Management are the three functions of Display Element Maintenance, Display Scheduling and Photopen servicing. Let us address these individually. A Display Element consists of a set of refresh instructions which define a particular display entity. Within the monitor, these Display Elements are assigned a separate memory area managed by the Monitor. Services within this domain include Element Creation, deletion, enable and disable. In general, Display Elements may be either passed from a user task area or loaded directly from an external medium. Once defined to the system, Display Elements may be brought into the current image via the Display Scheduler.

The Display Scheduler assumes responsibility for managing the Graphic Controller. This consists of displaying all the currently scheduled Elements according to attributes supplied by the user tasks. These attributes consist of, for example, position, color, intensity and line structure. Selective enable/disable of individual schedule table entries is also provided. The Display Scheduler thus provides for considerable flexibility in the final image with minimal refresh code manipulation.

The Photopen capability is included within the Display Management facilities because of the close association with the display schedule table. In particular, users may selectively sensitize individual display elements to be responsive to the photopen. Furthermore, photopen input may be done in either wait or no-wait mode. Thus, user tasks may, at their discretion, provide for the typical photopen applications of object specification, list selection, image editing, etc.

The last major monitor functional area comprises those functions associated with data input-output. These activities include task/display-element loading, data input, operator interaction and any host communications. As in most computer operating systems, these functions consist principally of the various device handlers. With respect to the Graphic 7 monitor, it should be noted that considerable flexibility has been left to the individual device handlers due to minimal centralized I/O activities. In other words, most of the processing associated with a user I/O request is done within the handler rather than having some functions provided within an I/O nucleus. The advantage of this approach is maximum flexibility to the driver and relative ease of system adaption to configuration changes. One possible disadvantage is greater memory usage. With regard to user task I/O, the approach is fairly conventional with a standard control block structure and the usual transfer options.

In summary, the Graphic 7 Monitor provides many of the functions relevant to the real-time graphics problem. These capabilities have the effect of relieving the host machine of burden-some terminal management responsibilities and improving response to operator demands.

GRAPHIC 7 MONITOR

MONITOR SUPERVISOR

2.0 MONITOR SUPERVISOR

The Monitor Supervisor will be principally concerned with coordinating the CPU resource. This includes selecting the appropriate task for execution and fielding user-task-originated requests for monitor functions. The communication medium between the monitor and user tasks is the EMT instruction. This instruction allows for passing an 8-bit variable code in the instruction word to the EMT trap handler. This trap handler represents the heart of the Monitor Supervisor.

The 8-bit variable field is utilized in two sections as follows: The high order four bits indicate a particular monitor functional area. Current assignments for these four bits (i.e. the major monitor functional areas) are listed in Figure 2.0-1. The low order four bits of the EMT instruction word indicate the particular sub-function (i.e. entry point) within the indicated functional area. The Supervisor will simply pass this entry code to the functional area where decisions regarding such are handled on a localized basis. Specific sub-functions for each monitor functional area are outlined within the individual functional area descriptions.

Many monitor functions require the specification of a list of parameters which indicate the location of relevant data or details regarding the service to be performed. This parameter list, when required, will be supplied via a pointer in general register one. Thus, prior to executing the EMT instruction, the user will typically load a parameter block address into register one. Particular parameter block structures are indicated with each sub-function description.

The user may ascertain the success or failure of any monitor request by testing the status return code which resides in the task header (T$MRST). Success is generally indicated by a zero in this byte. Non-zero return codes indicate an error

EMT INSTRUCTION CODE: 104000 $\rightarrow$ 104377 (octal)

CODE (HEX)	MONITOR FUNCTION
0x	Reserved
1x	Monitor Services
2x	Memory Management
3x	Task Management
4x	Display Element Management
5x	Display Management
6x	PHOTOPEN Services
7x	I/O
8x	Timing Services
9x	Data Transfer Services
Ax	Unused*
Bx	Unused*
Cx	Unused*
Dx	Unused*
Ex	Unused*
Fx	Unused*

*Reserved for future expansion

FIGURE 2.0-1 MONITOR CALL CODES

in the parameter specifications or the unavailability of a necessary system resource. Particular codes relevant to each sub-function are listed with the function description.

The Monitor-Task communications mechanism is depicted in Figure 2.0-2.

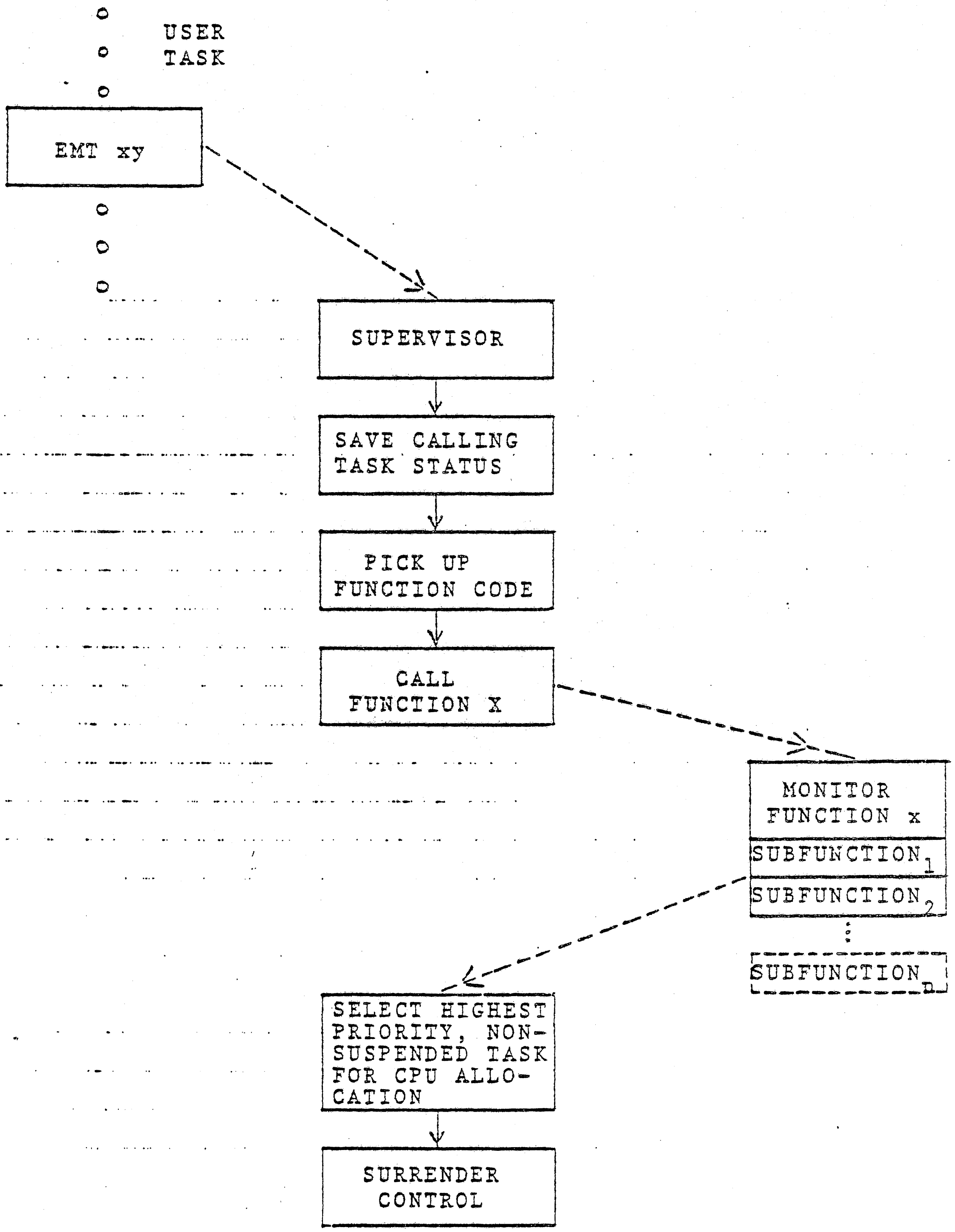


FIGURE 2.0-2 TASK $\longleftrightarrow$ MONITOR COMMUNICATIONS

2.1 SUPERVISOR IMPLEMENTATION

The Supervisor module includes the following components:

a) System constants and parameters
b) System initialization
c) EMT servicing
d) Interrupt save-state/restore-state

The constants and parameters section includes those quantities which describe the current system state. Some of the parameters may require adjustment during system generation. The system initialization section receives control when the monitor image is initially loaded. In addition, this routine receives control when no other task is active. The EMT service routine will intercept monitor requests issued by user tasks and direct them to the appropriate monitor functional area. During I/O operations the respective interrupt handlers will use the save/restore subroutines to record the system state prior to interrupt servicing, and then restore that state after servicing is completed. Inherent in these subroutines is the CPU allocation algorithm which operates on the basis of task priority and other task status conditions.

GRAPHIC 7 MONITOR
MEMORY MANAGEMENT

3.1 MEMORY MANAGEMENT OVERVIEW

The Memory Management section of the Graphic 7 Monitor is responsible for maintaining the current status of the memory resource. For this system, the memory resource is divided into several classifications, with a management scheme tailored to each. An important aspect of the memory structure is the hardware memory management. The allocation of available memory space is heavily dependent on the hardware flexibilities provided. First, the resident monitor code must reside in non-relocated (direct address) space as shown in Figure 3.1-1. This includes addresses 100000 thru 157777 (octal). The remaining memory space is then divided into three distinct regions as follows: Task space will include all 4K blocks in relocated memory space, normally above octal address 160000. Display Element memory space will normally encompass octal addresses 20000 through 77777. This is indirect address space dedicated to Graphic Controller refresh code. The remaining 4K block, addresses 0 through 17777, will be used for the schedule table. This direct address space will be used to control the display operation per directives issued from user tasks. The following sections address each of the memory areas in more detail and discuss the various monitor services available for communicating with the memory management services. Users should note that these services are available only via other monitor functions, i.e., user tasks do not directly issue memory management requests.

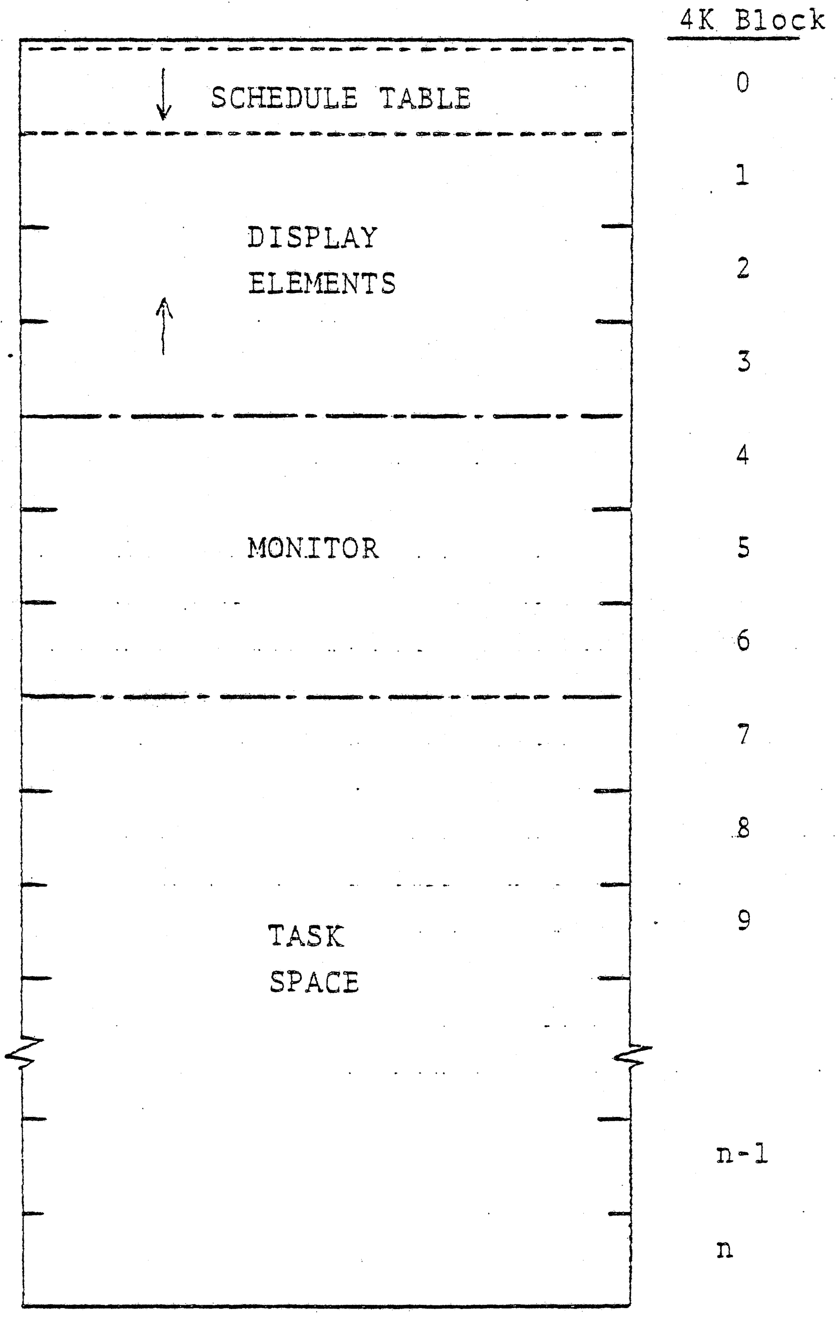


Figure 3.1-1 Memory Allocation

3.2 TASK MEMORY MANAGEMENT

Task memory space will be allocated in 4K (words) sections with up to three blocks per task. This limitation is based on the hardware memory management scheme which provides three relocation registers. For task memory then, the allocation algorithm simply looks for available 4K blocks without regard to continuity or actual location. As with the other memory areas, the monitor routines are principally concerned with available space rather than maintaining a list of allocated space. The implication is that the space requestor is responsible for returning the space when it is no longer required. In the case of task memory space, the status is maintained via a simple bit map. Thus each 4K block has a corresponding bit which is zero if the block is available. The initial bit map value is determined at system generation time by setting bits corresponding to those 4K blocks which are taken up by the other data types, the monitor or global common blocks. The bit map approach is highly efficient and simple to implement but, of course, the 4K block granularity results in some inefficiencies in allocation. Readers expecting to use this system should also consult the Appendix which discusses the hardware memory management scheme.* Following are the monitor subfunctions associated with task memory management.

* In particular, note that user tasks should begin at address 20000 (octal) in order to properly activate the relocation feature.

3.2.1 Allocate Task Memory Space

This service will be used by the Task Management area to fetch memory space prior to task load. The requested area length is rounded up to include whole 4K (word) blocks. Up to three such blocks will be provided. The blocks are not guaranteed to be contiguous.

EMT Code: 21 (Hex)

Parameter Block:

WORD	CONTENTS
0	Start Address (returned)*
1	Required Length (bytes)
2	Relocation Register 1 & 2
3	Relocation Register 3

Status Return Codes:

0 - Request satisfied
2 - Insufficient memory

* The Start Address is basically meaningless - it is always set to the constant 20000 (8). The area start is effectively the beginning of the 4K block pointed to by relocation register one.

3.2.2 Deallocate Task Space

Upon task exit or abort, the corresponding 4K blocks are returned to the system pool. Respective bits in the allocation map are cleared. The parameter block is of the same structure as that used in the Allocate service, however, the first two words are ignored. This service is employed only by the Task Management monitor area.

EMT Code: 22 (Hex)

Parameter Block:

WORD	CONTENTS
0	Ignored
1	Ignored
2	Relocation Register 1 & 2
3	Relocation Register 3

(A zero value for a Relocation Register entry implies no memory was associated with that offset, i.e., the task image did not require all three registers).

Status Return Codes:

0 - Request satisfied

3.3 <u>DISPLAY ELEMENT MEMORY MANAGEMENT</u>

Display Elements represent jobs which are presented to the Graphic Controller. These data blocks are stored together in a common memory region managed by this monitor section. Display element lengths can vary widely, thus these routines must be able to optimize the allocation to achieve reasonable memory usage. On deallocation, these routines will concatenate adjacent areas when possible. Due to addressing complications, allocated areas are prevented from crossing 4K (word) memory boundaries. Available space in the display element area is maintained by a list of free areas and a pointer to the next available open space. On allocation, the list of free areas is first searched to determine if an available space exists. Otherwise, the space is taken in the open area. The list of free areas is maintained via forward pointers and length fields, as depicted in Figure 3.3-1. Note that indirect addressing will generally be necessary throughout the display element area. The following two sections discuss the relevant Memory Management entry points.

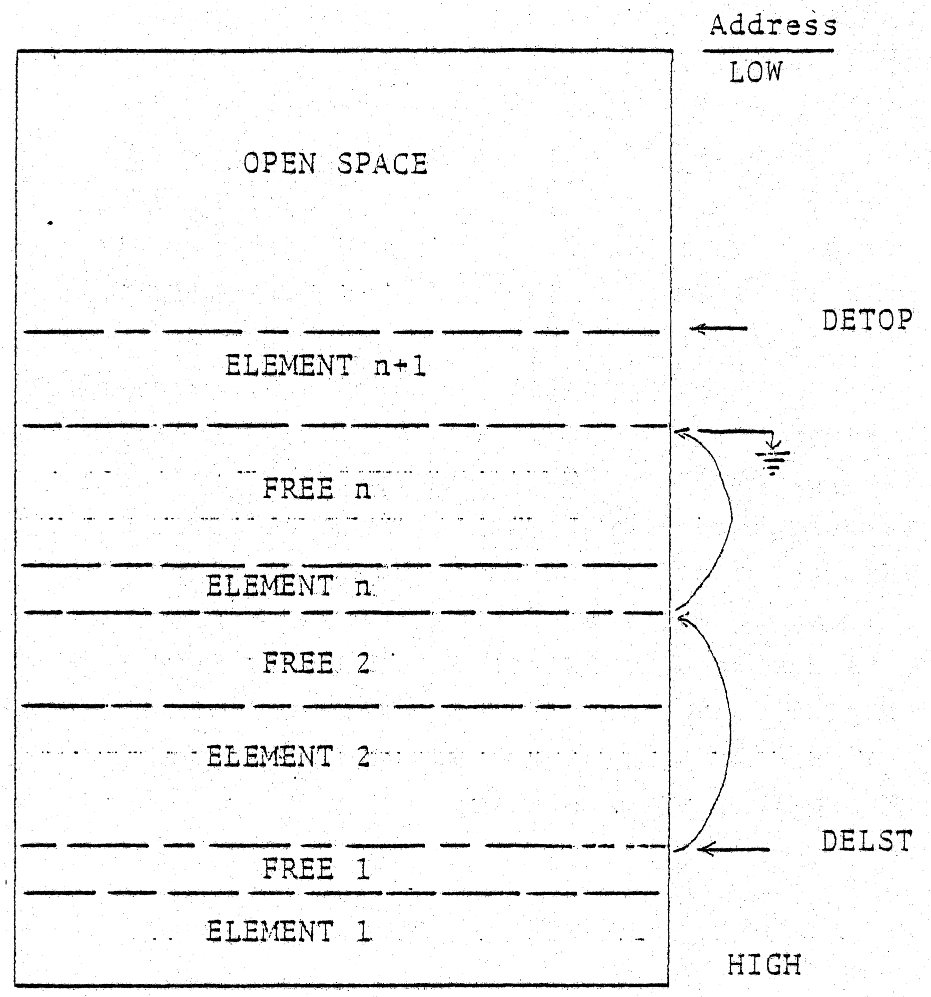


Figure 3.3-1 Display Element Space Inventory

3.3.1 Allocate Display Element Space

This entry will be used by the Display Element Management section to fetch space for element storage. The provided area will be rounded up to include an even number of 4-word allocation units. In addition, the routine will ensure that the area does not cross a 4K memory block boundary. If available "slots" exist in the element region, such will be searched for that which yields minimum residue.

EMT Code: 23 (Hex)

Parameter Block:

WORD	CONTENTS
0	Start Address (returned)*
1	Required length (bytes)

Status Return Codes:

0 - Request satisfied
2 - Insufficient memory

* As per Figure 3.1-1, this is the actual physical address since this area lies within the first 32K of core.

3.2.2 Deallocate Display Element Space

The Display Element Management monitor services will use this entry to release display element space. Released space will be combined with other "holes" to the extent possible while prohibiting 4K boundary spans. Users should delete display elements as soon as they are no longer required so as to free space for other tasks' elements.

EMT Code: 24 (Hex)

Parameter Block:

WORD	CONTENTS
0	Start Address
1	Length (bytes)

Status Return Codes:

0 - Request satisfied

3.4 SCHEDULE TABLE SPACE

The Schedule Table is used to control the operation of the Graphic Controller Processor. This table is managed by the Display Scheduler monitor services with calls to this and the following subfunctions for fetch/release of table space. The inventory of Schedule Table memory space is simplified by the allocation of fixed length (12 word) blocks. As in the other memory areas, the Memory Management function only maintains the inventory of un-allocated space. This free space is recorded by a linked list of free blocks and a pointer to the beginning of the open area. No attempt at concatenating adjacent blocks is necessary since allocations are always of the same size. The system design assumes that the schedule table space is contained within a direct addressing area, in this case, octal addresses 1000 through 17777. Usage of the direct addressing area will allow more efficient schedule table monitor services while also allowing users direct access to their table entries, if desired. As shown in Figure 3.1-1, the schedule table and the display elements build against each other but, of course, the schedule table is not allowed to grow out of the first 4K block. (Recall also that interrupt vectors occupy the beginning of said 4K block.) The following two monitor services provide for allocation/release of the schedule table space.

3.4.1 Allocate Schedule Table Space

The Display Scheduler monitor service will use this entry to fetch a 12-word block for construction of a new schedule table entry. Since the length is fixed, only the start address need be returned in the caller parameter block.

EMT Code: 25 (Hex)

Parameter Block:

WORD	CONTENTS
0	Start Address (returned)

Status Return Codes:

0 - Request satisfied
2 - No space available

3.4.2 Deallocate Schedule Table Space

This service will be utilized when the user deletes a schedule table entry. As with allocation, this entry is generally only called by the Display Scheduler monitor services.

EMT Code: 26 (Hex)

Parameter Block:

WORD	CONTENTS
0	Start Address

(The length is assumed to be a fixed 12 words.)

Status Return Codes:

0 - Request satisfied

3.5 GLOBAL COMMON ALLOCATION

During system generation, up to four, 4K blocks may be set aside as system global common areas. Depending on their length, user tasks may have access to either one or two of the four areas. Generally, the global common blocks will be taken directly from available task space. However, if demand on display element space is lax then it will be possible to use a 4K block from this area.* In any case, user tasks will need to indirectly map into the global common area and thus cannot be using all three relocation registers for task execution. The usage or format of the global common area is entirely left to the particular applicational requirements. Obtaining access to a global common area, from within a user task, is discussed in the Task Management section.

* It will not be possible to specify block 0 (zero).

3.6 MEMORY MANAGEMENT SYSTEM GENERATION CONSIDERATIONS

At system generation time the user may wish to tailor the memory configuration to fit a particular application requirement. Normally, the configuration shown in Figure 3.1-1 will be provided, with users specifying global common blocks as necessary. However, users may wish to restrict the display element space or schedule table space via manipulation of the associated bounds contained within the routine. In any case, the bit map initial value (BMASK) will need to be adjusted to account for which 4K blocks are available for task execution. In particular, bits must be set to correspond to those blocks which have been specified for global common areas (GBLCOM).

GRAPHIC 7 MONITOR
TASK MANAGEMENT

4.0 GRAPHIC 7 TASK DEFINITION

Within the Graphic 7 Operating System, user problems will generally be solved by a group of inter-related "tasks". To the Monitor, a task represents the smallest entity which can be allocated - the CPU resource. Beyond this and certain memory allocation restrictions, the definition of "tasks" is left entirely up to the particular user application. Also, there is no maximum number of tasks other than as restricted by the memory size.

The Task Management monitor section is responsible for maintaining the current system status with respect to the task population. In general, any activity which requires a task to either enter or exit the system must be coordinated by this Monitor section. In addition, should a task wish to suspend it-self (timed or indefinite) the appropriate Task Management entry will be employed.

Current system task status is maintained by means of a linked list with pointers contained in the task headers. (Since all task headers start on a 4K boundary, the pointers are simply relocation register 1 values.) The task list is maintained in priority order so that the highest priority task capable of execution may easily be located. The task list header will be located in monitor global variable TSKLST. Allocation of task memory space with implications for system design is discussed in the following.

4.1 TASK MEMORY ALLOCATION

As discussed under the Memory Management section, allocation of task space is in 4K word blocks. Furthermore, since only three of eight 4K blocks within the 16 bit address space are relocatable, task size is limited to 12K words.* This restriction may make it necessary to divide tasks which would otherwise be a logical, unified application area.

In addition to task execution space, it may also be necessary to access a global common area or the Display Element storage region.** For these actions, it will be necessary to have available one of the three relocation registers to point to the particular 4K block under consideration.*** Given that such additional addressing is necessary, it is clear that user task length will need to be held to 8K words. (Actually somewhat less due to stack space requirements.)

The above discussed limitations will obviously make it necessary to give careful consideration to task definition during system design.

*For additional clarification on this point, please see the Appendix which discusses the Memory Management hardware.

**Access to a global common area may be provided automatically on task load. References into the Display Element region must be arranged by the user.

***Relocation Register 1 may never be used for other than addressing into the first 4K block of task space.

4.2 TASK-MONITOR COMMUNICATION

User tasks may issue a variety of requests to the Graphic 7 Monitor. Such requests are communicated to the Monitor by an EMT instruction which contains a function code in the low order byte. The Monitor interprets this byte as two hexadecimal digits, the first specifying the monitor functional area and the second a particular subfunction. Along with the EMT instruction, most Monitor functions require a list of parameters against which the function is applied. The user supplies this parameter list by placing the start address of such in Register 1 prior to the EMT.

Successful completion of the user request is indicated by returning a status code to a dedicated location in the task header. Generally, user tasks will need to check the status code to verify proper command performance. Particular status codes for each Monitor function are listed with the function description. Normally, a zero status indicates successful completion.

On issuing an EMT to the Monitor, the current task status is saved either in the task header or on the user stack. Except for I/O operations which may be either wait or no-wait, the user task is suspended while the request is satisfied. On return from the subject Monitor functional area, the system supervisor will normally perform a task scan to allocate the CPU resource to the current highest priority task. Thus, as the system environment changes, the various tasks can all have an opportunity for execution.

4.3 TASK HEADER FORMAT

Most of the information concerning a task's current status is contained in the task header. This data block, outlined in Figure 4.3-1, will be pre-allocated by the user at assembly time. Some of the parameters must be initialized at assembly time while others are strictly reserved for Monitor usage. The following sections address each of the individual parameters. The various fields will normally be referred to via their offset names which are of the form T$xxxx. Usage of the offset definitions is recommended since some re-arrangement of the header structure may be necessary as the system definition matures.

4.3.1 Task ID (T$ID)

The Task ID byte will contain the unique identification for the task. Each task to be entered into the system must have a unique ID to be used in conjunction with operator communication or task-task/task-monitor data exchange. Generally, this field will be referred to as two hexadecimal digits. The Task ID should be inserted at assembly time. User task ID's are restricted to the range 10_{16} to FF_{16}.

Word	Offset	Byte 1	Byte 0	
0	0	Priority T$PR	ID	T$ID
1	2	Status		T$ST
2	4		Monitor Req. Status	T$MRST
3	6		Task List Forward Link	T$FLNK
4	10	Relocation Reg. Two Save	Relocation Reg. One Save	T$RR
5	12	Memory Extension T$EXTR	Relocation Reg. Three Save	
6	14	Stack Pointer Save		T$REG
7	16	Allocation Save		T$ALLC
8	20	Start Address		T$STRT
9	22	Global Common Requests		T$COM
10	24	Display Element Control Block List Header		T$ELS
11	26	Display Schedule Control Block List Header		T$SLS
12	30	Photopen Control Block List Header		T$PPEN
13	32	Task Suspension		T$TIM
14	34	Control		
15	36	Private Timer		T$PTI
16	40	Control		
17	42			
18	44			
19	46			
20	50			
21	52			
22	54			
23	56			
24	60	Logical Device 1		T$DEV
25	62	Logical Device 2		
26	64	Logical Device 3		
27	66	Logical Device 4		
28	70	Logical Device 5		
29	72	Logical Device 6		
30	74	Logical Device 7		
31	76	Logical Device 8		

Figure 4.3-1 Task Header Layout

4.3.2 Task Priority (T$PR)

The Task Priority will indicate the particular task's requirements for CPU time with respect to other tasks in the system. The task list will be maintained in decreasing priority order. Priority should be a positive integer between 1 (lowest) and 127. This field should be defined at assembly time.

4.3.3 Task Status (T$ST)

The Task Status bits will indicate the current state of the task with respect to various system-task interfaces. The low order eight bits of the status word will be modifiable by other tasks. The definition of the individual bits is as follows:

Bit	On Implies
15	Task Disabled
14	Task Suspended
13	I/O Wait
12	Timed Suspension
11	Photopen Wait
10	Waiting for Data
9	
8	
7-0	User Defined (Externally Modifiable)

The user will normally set this word to zero at assembly time, however, the Disable or Suspend bits may be used to effect a particular initial task state. With regard to bits 0-7, user tasks will usually establish usage conventions during system design.

4.3.4 Monitor Request Status (T$MRST)

The Monitor Request Status Byte is used to return a status code to the user task following a request for a monitor service. Zero generally indicates successful request satisfaction with other numbers defined for each individual subfunction. Initialization of this parameter is not necessary however, of course, space allocation is necessary at assembly time.

4.3.5 Task List Forward Link (T$FLNK)

The Forward Link simply serves to point to the next lower priority task. If this is the lowest priority task, then this parameter will be zero. This parameter is maintained by the operating system and will generally be of no concern to the user. (The actual contents will simply be relocation register one of the next task.) (Backward links in the task list are not currently being used.)

4.3.6 <u>Relocation Register Save Area (T$RR)</u>

The three bytes in this area will be used to store the task relocation registers whenever control is passed to the operating system. The user will not generally be concerned with these entries. (These values must be saved on interrupt since some tasks may use the relocation registers to access global common areas or display element space.)

4.3.7 <u>Extended Memory Allocation (T$EXTR)</u>

A user task may request an additional memory allocation above that required by the loaded task image. The extension quantity, expressed at offset T$EXTR (byte), is in units of 256 bytes. Such memory space is located at the end of the task image. Remember that task space is normally allocated in 4K word blocks so the memory extension may not actually result in any additional memory being allocated. The purpose of this service is to allow tasks to set aside space for data bases or other usages without having to define this space at assembly time; the result being a smaller load file.

4.3.8 Stack Pointer Save (T$REG)

This word will be used to save the task stack pointer whenever the task surrenders control. Other general registers will be saved on the stack. Note that the task relocation registers will, in general, need to be installed to access stack entries. The stack pointer save word need not be initialized by the user at assembly time since the operating system will automatically point R6 to the last word in the last 4K block allocated. (In allocating space for the user task, the system will request 32 words greater than the task length.)

4.3.9 Allocation Save (T$ALLC)

The Allocation Save word will be used to record the initial values of the task relocation registers. This serves to indicate which 4K memory blocks were allocated so that such can be de-allocated on task exit. This parameter is of no concern to user tasks but, of course, space must be allocated for the word at assembly time.

4.3.10 Start Address (T$STRT)

The Start Address word will contain the desired start address for user task execution. This will be defined at assembly time; typically the first word following the task header.

4.3.11 Global Common Requests (T$CØM)

User tasks which do not require all three relocation registers for execution addressing may access one or two global common areas. The operating system may have defined to it, at system generation time, up to four 4K blocks to be used for global common areas. For user tasks of less than 8K words, the third relocation register (RR3) may be used to reference one of the four global common blocks. (Addresses would be of the form: 6xxxx where 1/0xxxx is the offset into the 4K block.) This may be indicated at assembly time by placing a binary 1-4 at T$CØM. The correct relocation register value, to correspond to the common block selected, will be inserted at task load time. Similarly, for user tasks of less than 4K words, relocation register two may be pointed to a global common area by placing a binary 1-4 at T$CØM+1. (Clearly it would not make much sense to have the same number in both T$COM bytes.) If global common access is not desired, then these bytes should be set to zero at assembly time. Inability to satisfy global common requests will result in task load failure.

4.3.12 Display Element Control Block List Header (T$ELS)

A linked list of Display Element Control Blocks is maintained within each user task. This list is updated whenever the user task references the Display Element Management monitor services. Subject word is simply the list header (i.e., pointer to first entry). Users should set this word to zero at assembly time.

4.3.13 Schedule Control Block List Header (T$SLS)

User Schedule Control Blocks are maintained on a linked list for which T$SLS is the list header. This list is updated by the Display Scheduler Monitor sub-functions. Users should set this word to zero at assembly time.

4.3.14 Photopen Control Block List Header (T$PPEN)

Requests for Photopen input will be linked together within each user task. This word serves as the linked list header. The Photopen monitor services maintain the control block list. Users should set this word to zero at assembly time.

4.3.15 Task Suspension Control (T$TIM)

Words at T$TIM and T$TIM+2 will be used to maintain the task suspension timer on timed suspension. The first word is used for linking with the second word containing the actual timer value. These words are manipulated only by the operating system. They should be set to zero at assembly time.

4.3.16 Private Timer Control (T$PTI)

Users issuing private timer requests will have such maintained on linked lists within their task areas. The two words at T$PTI are used for linking. The first word essentially links the various tasks currently having outstanding private timers. The second word is the list header for the individual task's timers. These words are maintained by the Timing Services monitor section and are not manipulated by the user task. Both words should be set to zero at assembly time.

4.3.17 <u>Logical Device Association (T$DEV)</u>

The eight words at T$DEV will serve to connect the user specified logical devices (1-8) with any eight physical device/unit combinations. The logical device specification (contained in the second word of the I/O control block) will simply serve as an index into this table. The low order byte should contain the physical device number (1-) with the high order byte the unit number. These specifications will normally be made at assembly time. All eight entries must be defined; unused entries set to zero. The correspondence between physical device number and respective handler will be set up during system generation.

4.4 TASK LOADING

Tasks will generally be brought into the system from some external, bulk storage medium. The task load monitor request allows for the specification of a file name to be communicated to the host or external medium controller. The load process involves first issuing a "File Query" to the subject handler. This request specifies the target file name and should return the file length. The length information is then used to allocate task space. After the memory space has been successfully allocated, the task image is input using a standard read request.*

Two points should be emphasized regarding the above described procedure. First, the specified handler must be able to accept a "File Query" request, as described in the I/O section. Secondly, when building user task images on the host storage medium, the length information must be made readily available. For paper tape oriented input, this information is already included in the absolute load format.

Following the task image input, the task is linked into the current task list and made eligible for execution beginning at the user specified start address.

*Note that contiguous 4K memory blocks are <u>not</u> guaranteed by the memory allocation section. This has possible implications for DMA transfers.

4.5 TASK MANAGEMENT MONITOR SERVICES

User tasks will access the various Task Management monitor functions via EMT codes of the form 3x (Hex). The services available are as follows:

4.5.1 Load Task

This service will be used to bring a new task into the system. Specified input device (logical unit number) must be able to accept a "File Query" request.

EMT Code: 31 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)
1	Logical Device Number
2-6	Source File Name (ASCII, terminated by a null)

Status Return Codes:

0	Request Satisfied
2	Insufficient Memory
32	I/O Failure
33	Invalid Device Code
34	Task ID Mis-Match
35	Device Unavailable
36	File Not Found
37	Unable to Satisfy Global Common Requests

4.5.2 Load Overlay

This service is not currently supported.

4.5.3 Enable Task

This entry may be used to reset the task disabled bit and thus make the task eligible for execution. The Enable/Disable services should not normally be used on an inter-task basis.

EMT Code: 33 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)

Status Return Codes:

```
 0 - Request Satisfied
32 - No Such Task  Found
```

4.5.4 Disable Task

This service will set the Disabled bit in the task header status bits (T$ST). Task is then ineligible for execution.

EMT Code: 34 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)

Status Return Codes:

```
 0 - Request Satisfied
32 - No Such Task Found
```

4.5.5 Task Suspend

This service may be used to suspend the calling task for a timed or indefinite interval. If timed, then the Timing Services section will be notified to set up a timer. Tasks can only suspend themselves.

EMT Code: 35 (Hex)

Parameter Block:

Word	Contents
0	Time Out Value in 1/10 Seconds, Zero implies indefinite suspension

Status Return Codes:

0 - Request Satisfied

4.5.6 Task Continue

Continue will remove a task from a timed or indefinite suspension state. This service may be directed from/to any task in the system.

EMT Code: 36 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)

Status Return Codes:

0 - Request Satisfied
32 - No Such Task Found

4.5.7 Task Exit

A task leaves the system and thereby relinquishes system resources by issuing the Exit EMT. All I/O must have been completed prior to issuing this request. It is also recommended, though not necessary, to delete all schedule table entries, display elements and private timers.

EMT Code: 37 (Hex)

Parameter Block: None (Only the calling task may Exit)

Status Return Codes: N/A

4.5.8 Fetch Task Status

The current task status is maintained in word T$ST in the task header. This service will allow any task to check the status of any other task.

EMT Code: 38 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)
1	Returned Task Status

Status Return Codes:

0 - Request Satisfied
32 - No Such Task

4.5.9 Update Task Status

This service allows for the modification of bit 0-7 in the subject task's status word. The supplied bits are logically ored with the current status word contents. Users may wish to restrict, by definition, some of these bits to be read or write only.

EMT Code: 39 (Hex)

Parameter Block:

Word	Contents
0	Task ID (low order byte)
1	New Status Bits (Byte 0)

Status Return Codes:

0 - Request Satisfied
32 - No Such Task

4.5.10 Abort Task

The Abort entry may be used to cause the involuntary termination of a user task. This entry is included primarily for operator communications usage.

EMT Code: 3A (Hex)
Parameter Block:

Word	Contents
0	Task ID (Low Order Byte)

Status Return Codes:

0	- Request satisfied
32	- No such task found

4.5.11 Search for Task

The Search function will allow the caller to locate the beginning of any user task in the system. The returned value is essentially the 4K block number which becomes the relocation register value if access to the subject task is required.

EMT Code: 3B (Hex)
Parameter Block:

Word	Contents
0	Task ID (low order byte)
1	4K block number (returned)

Status Return Codes:

0	- Request satisfied
32	- No such task found

4.5.12 <u>Continue Task & Suspend</u>

Inter-task execution control may be aided by the ability to, in a single monitor request, continue a given task and suspend oneself. The suspension provided by this service is non-timed.

EMT Code: 3C (Hex)

Parameter Block:

Word	Contents
0	Task ID of task to be continued (low order byte)

Status Return Codes:

- 0 - Request satisfied
- 32 - No such task found

4.6 TASK ERROR MONITORING

The monitor generally performs no error processing for user tasks. Thus, mistakes in the preparation of the various control blocks or other programming oversights will generally result in fatal system errors. Little advice can be offered in such circumstances above the usual debug techniques of examining the various task variables, status indicators and stack. This examination may be performed through the ROM resident debug capability. The appropriate 4k block number for the task in question will need to be inserted in the relocation register before attempting to display task-memory contents.

One exception to the above is the case of the bus time-out error. The monitor supervisor will intercept this interrupt and, if currently executing within a user task, will mark the task disabled and notify operator communications (assuming OPCOM is installed).

GRAPHIC 7 MONITOR

DISPLAY ELEMENT MANAGEMENT

5.0 DISPLAY ELEMENT MANAGEMENT OVERVIEW

Construction of the graphic image will entail the definition of various picture elements and the scheduling of these elements to suit the particular application requirement. Display Element Management includes those services which are used to define picture elements to the Graphic 7 monitor. Scheduling of picture elements is covered in a separate monitor functional area. Within the Graphic 7 Operating System structure, display elements are stored in a dedicated memory area, separate from task image areas. The picture elements (refresh code blocks) are processed directly from this dedicated memory area by the Graphic Controller. Three methods exist for presenting display elements to the monitor. First, the user task may "pass" an element directly from his task image. This procedure would generally apply to small, fairly static picture elements. Secondly, the user may request to have a display element loaded from an external medium. For this case, the user must, obviously, specify a file name or other external data source identifier. The external medium approach will normally be used for large, static display elements. Finally, user tasks may allocate display element space and dynamically build the refresh block as, for example, real time data is received. This approach yields maximum flexibility but can be rather expensive to implement.

Once the display element has been defined, the user task may cause it to be displayed via the Display Scheduler. Temporary removal of the element from the current image can be done via the Element Disable Service. Elements which are no longer needed should be discarded via the Element Delete Service.

	WORD	CONTENTS
	0	Reserved (Link Field)
Required	1	Element Start Address*
	2	Flags/Status
	3	Length (Bytes)
	4	Start Address or Input Device**
Optional	5-9	File Name***

STATUS BITS			FLAG BITS		
0			8	1	Position Absolute****
1			9	1	Size Absolute****
2			10		
3			11		
4			12		
5			13		
6	1	Control block in use	14		
7	0	Enabled, 1 Disabled	15		

* Address in monitor storage area (returned).
** Start Address in user task for "PASS" service, Input Device for "LOAD" service.
*** File Name relevant only for "LOAD"; must be terminated by Null character.
**** These bits are not currently used by the monitor.

Figure 5.1-1 Display Element Control Block Format

5.1 STANDARD ELEMENT CONTROL BLOCK

Display Elements always have an associated control block within the task which requested their definition. This control block, outlined in Figure 5.1-1, contains all the parameters necessary to the various Monitor functions. This block is important because the Monitor does not maintain any internal inventory of Display Elements or associated memory space. Thus, whenever a user task issues a monitor request relating to a display element, the associated parameter (control) block address must be specified. Within the user task image, the Display Element Control Blocks are maintained on a linked list. This is a simple, singly linked list with the list header at T$ELS within the task header. Maintenance of this list is done entirely by the Monitor routines and will generally be of no concern to the application task developer. In summary, the Display Element Control Block is the parameter block for user requests to monitor service and also the inventory mechanism for monitoring the current display element configuration.

The individual fields within the element control block are defined as follows: word zero is used for control block list linking and is generally of no concern to the user. Word one contains the element start address within the monitor storage area. This address is the actual physical address since the display elements are stored within the first 32K of memory. Note also that this address actually points to word zero as shown in Figure 5.2-1, thus the user code will normally start at 4 bytes past the given start address. The third control block word contains status and flag bits as outlined in Figure 5.1-1. Next comes the element length in bytes. This length does <u>not</u> include the six control bytes which are added to each element. Words 4 through 9 are defined only for certain element services as described in Section 5.3.

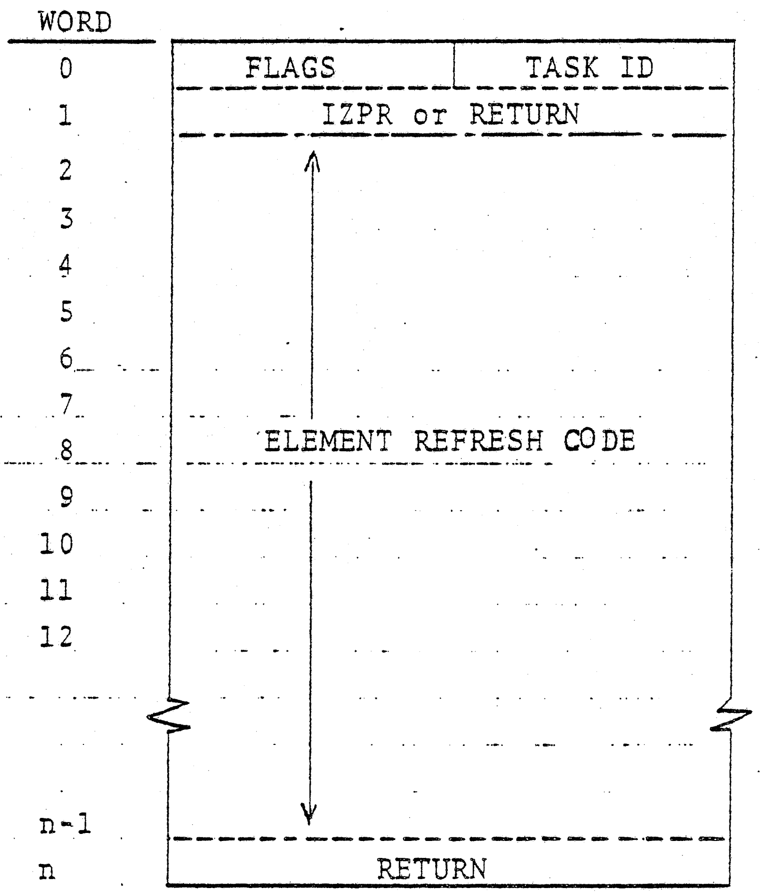


Figure 5.2-1 Display Element Storage Format

5.2 DISPLAY ELEMENT STORAGE FORMAT

Display Elements are of variable length and packed within the reserved system storage area. (Generally this will be some or all of 4K pages 1, 2 and 3.) The length of the allocated space is six bytes greater than that specified by word three of the Element Control block. The three extra words are accounted for by two control words at the beginning of the element and a "RETURN" instruction as the last word.* Of the first two words, the initial contains the Flags byte (direct from word 2 of the respective element control block) and the parent task ID. The second word will contain either an IZPR instruction (if the element is enabled) or a RETURN instruction (for the case of a disabled element). The Element Storage format is portrayed in Figure 5.2-1. The maximum element length, including control words, is 4K words.

* Recall that the elements are executed as subroutines from related Schedule Table entries.

5.3 DISPLAY ELEMENT MONITOR SERVICES

The following sections detail the basic monitor services related to Display Element manipulation within the Monitor. Note that the first four words of all parameter blocks are common, as presented in Figure 5.1-1.

5.3.1 Load Display Element

The Load service will be used to fetch a display element from an external medium. The input device specification should be a logical device number, relative to the calling task. The source file identifier should be an ASCII string terminated by a null.

EMT Code: 41 (Hex)

Parameter Block:

WORD	CONTENTS
0-3	(standard)
4	Logical Input Device
5-9	Source File Identifier

Status Return Codes:

0	Request satisfied
2	Insufficient Memory
32	I/O failure
35	Device unavailable
36	File not found

5.3.2 Pass Display Element

The Pass service may be used to transfer a display element image from user task space to monitor space. Display elements, i.e., refresh code blocks, are always assumed to begin on word boundaries.

EMT Code: 42 (Hex)

Parameter Block:

WORD	CONTENTS
0-3	(standard)
4	Element start address within user task

Status Return Codes:

0 - Request satisfied

2 - Insufficient memory

5.3.3 Update Display Element

Update may be used to provide a revised copy of a display element which was previously defined via the PASS function. The length of the element may not be changed. The user task is responsible for avoiding conflicts with the Graphic Controller in the case of updating an active (enabled) display element. This service is oriented toward fixed format refresh elements such as textual tables.

EMT Code: 43 (Hex)

Parameter Block:

Standard Element Control Block
(words 0-4)

Status Return Codes:

0 - Request satisfied

5.3.4 Fetch Element Space

There may be instances where the user task must construct the display element according to data received during execution. For this circumstance, the system allows the user task to allocate display element space via the Fetch service. Obviously, the element should be fully defined (including the ending RETURN instruction) prior to being scheduled. Access to this element area will be via one of the relocation registers and, thus, the user task will generally need to be less than 8K in length. Construction of the required relocation register bias is left up to the user task. This may be done by isolating the three high order bits in the element address (returned in word 2 of the parameter block); moving these three bits to relocation register three and then replacing them by 011. The effect is to force mapping through relocation register three, into the target 4K block. (Caution: The address contained in word two of the parameter block should be considered "read only" to the user task.)*.

EMT Code: 44 (Hex)

Parameter Block:

- Standard Element Control Block -

Status Return Codes:

0 - Request satisfied
2 - Insufficient memory

* Remember that user refresh code should start at four bytes past the given start address.

5.3.5 Delete Display Element

The Delete function should be employed after the particular element is no longer required by the user task. This service will free the associated display element memory space and remove the control block from the user task control block list.

IMPORTANT: All schedule table entries referencing said element should be deleted prior to element removal. The system will not check for this error.

EMT Code: 45 (Hex)

Parameter Block:

- Standard Element Control Block -

Status Return Codes:

0 - Request satisfied
33 - No such element*

* This error would normally occur when the user fails to set R1 to point to subject parameter block prior to the Delete EMT.

5.3.6 Enable Display Element

The second word within the element space will determine the Enabled/Disabled element state. The Enable service simply inserts an IZPR instruction into this word.

EMT Code: 46 (Hex)

Parameter Block:

- Standard Element Control Block -

Status Return Codes:

0 - Request satisfied

5.3.7 Disable Display Element

The Disable service will insert a return in word two of the element space. (Users should note the differences in efficiency between disable of the element and disable of the associated schedule table entry or entries.)

EMT Code: 47 (Hex)

Parameter Block:

- Standard Element Control Block -

Status Return Codes:

0 - Request satisfied

5.3.8 Purge Display Elements

On task exit, users should ensure that all display elements and schedule table entries have been removed from the system. This service may be used to eliminate all display elements. (Clearly the complementary Scheduler service should be exercised first.)

EMT Code: 48 (Hex)

Parameter Block: N/A

(The elements are found by scanning the user's element control block list.)

Status Return Codes:

0 - Request satisfied
(The absence of any element is not considered an error.)

5.3.9 Revise Display Element

The Revise service may be used to replace an existing display element with a new version. This differs from Update in that the new element is installed in an area distinct from the old element, any referent schedule entries are adjusted and then the old element is deleted. Revise is oriented toward non-fixed-format elements which typically, reflect real time varying data.

EMT Code: 4B (Hex)

Parameter Block:

WORD	CONTENTS
0	Pointer to Control Block for current element version
1	Start address of new element version
2	Length of new element version (bytes)

Status Return Codes:

0 - Request Satisfied

2 - Insufficient Memory

GRAPHIC 7 MONITOR

DISPLAY SCHEDULER

6.0 DISPLAY SCHEDULING OVERVIEW

The Display Scheduler monitor functions will be used to cause a Display Element to be presented in the CRT image. Various parameters, provided during scheduling, will control the actual element characteristics. These characteristics are, for example, screen position, intensity and color. The job of the Display Scheduler is to process all such display requests and cause the Graphic Controller to sequence through each of the Display Elements selected. This is affected by constructing, for each display request, a start-up block of refresh code containing the parameter initialization instructions and a call to the selected display element code. These start-up code blocks must then be linked together according to color and user-supplied priority.

Along with the basic scheduling capability, the monitor also allows enable/disable, delete and modify of selected scheduler control blocks. These capabilities will allow the user to tailor the display to fit a changing real-time environment. In summary then, display scheduling represents the second step in image creation, following display element definition. These two monitor functional areas will hopefully provide sufficient flexibility for most display creation/manipulation tasks.

6.1 STANDARD SCHEDULER CONTROL BLOCK

As in all task-monitor communications, a parameter block must accompany each monitor request. The address (user task relative) of such block is always supplied in register 1. Figure 6.1-1 outlines the parameter list which is standard for all Display Scheduler requests. Only the last word is optional. This parameter block supplies several important quantities. First, the linkage to the target display element is provided in word 3 as a pointer to the respective Element Control Block. This mechanism allows for assembly-time definition of the requested element. Second, the parameter block contains the various attributes relevant to this incidence of the selected element. Table 6.1-1 shows possible values for each of the attributes. Associated with each schedule request block, the monitor will build a companion Schedule Table entry as described in Section 6.2. It is important to remember that this association (between the request parameter block and the Schedule Table entry) must be maintained until the user task specifically requests the schedule entry be deleted. This is necessary since, as in Display Element Management, the monitor does not maintain any internal inventory of the current schedule table status.

The user may directly reference his Schedule Table entry via the address contained in word one of his request block. This should be an un-mapped address. The first word of the request block is a link field used to maintain all Scheduler request blocks belonging to this task on a linked list. The list header is at T$SLS within the user task header. This list is maintained by the monitor and will generally be of no interest to the user.

	WORD	15	0
0	0	Reserved (Link Field)	
2	1	Schedule Entry Address	
4	2	Flags	Status
6	3	Element Control Block Address	
10	4	CRT's	Priority
12	5	Color	Intensity
14	6	Line Structure	Character Size
16	7	X Start	
20	8	Y Start	
22	9		Modify Control

	STATUS BITS		FLAG BITS
0		8	Photopen Enable (CRT 1)
1		9	Photopen Enable (CRT 2)
2		10	Character Rotate
3	GC Status Pending	11	
4	Sch.Entry Caused Halt	12	
5	Sch.Ent.Caused Out-of-Bounds	13	Blink
6	Parameter Block in use	14	
7	1 ⇒ Disabled	15	

Figure 6.1-1 Standard Display Scheduler Parameter Block

ATTRIBUTE	OFFSET (8)	SIZE	POSSIBLE VALUES
CRT's	11	Byte	0 (No CRT's)
			1 CRT 1
			2 CRT 2
			3 CRT 1 & 2
Priority	10	Byte	1 → 127
Color	13	Byte	0 - Red
			1 - Orange
			2 - Yellow
			3 - Green
Intensity	12	Byte	0 → 7
Line Structure	15	Byte	0 - Solid
			1 - Dotted
			2 - Dashed
			3 - Center Line
Character Size	14	Byte	0 → 3
X Start	16	Word	-1023 → +1023 (Decimal)
Y Start	20	Word	-1023 → +1023 (Decimal)

Table 6.1-1 Schedule Request Attribute Values

6.2 SCHEDULE TABLE STRUCTURE

The Schedule Table is a linked list of Graphic Controller initiation blocks. Space for the construction of these blocks is allocated by the Memory Management services. Within each block are links for both scheduler management functions and Graphic Controller execution. In particular, Figure 6.2-1 shows the overall Schedule Table layout. You will note that each color has its own linked list of blocks. This structure provides for a minimum of color changes and also allows for the priority implementation discussed below.

Figure 6.2-2 shows the layout of the individual schedule table blocks.

The various refresh instructions, LDDZ, LDDP, etc. are constructed based on the attributes provided in the parameter block in the user task. The element start address is retrieved from the element control block contained within the user task. The "forward link" and "backward link" fields serve to doubly link (in circular fashion) all existing schedule table blocks. These linked lists, one for each color, are maintained in priority order.

The priority specification may be used to indicate the relative importance of this display request, and as such, help in avoiding image clutter and Graphic Controller overload. The implementation of the priority scheme is quite simple. First, when a new display request is received, the generated table entry is linked into the existing structure according to the indicated color and priority. Then, when constructing the Graphic Controller execution links (words 10 & 11), the routine will first

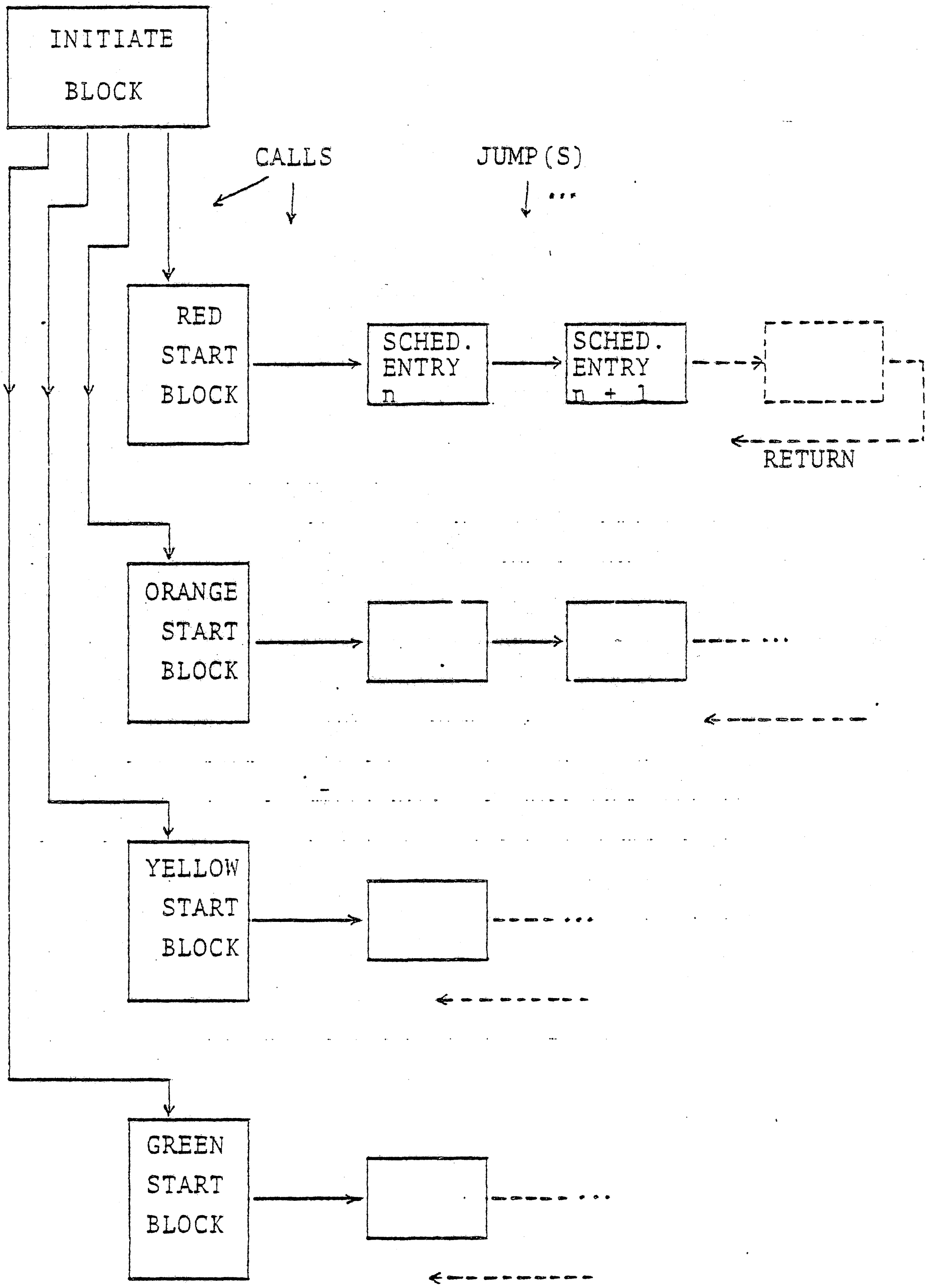


Figure 6.2-1 Schedule Table Structure

WORD	OFFSET(8)	
0	0	PRIORITY | TASK ID
1	2	FORWARD LINK
2	4	BACKWARD LINK
3	6	LDDZ
4	10	LDDP
5	12	LDTI
6	14	LDXA
7	16	MVYA
8	20	CALL
9	22	ABS. ELEMENT ADDRESS
10	24	JUMP
11	26	JUMP (LINK) ADDRESS

LDDZ	-	LOAD DISPLAY Z REGISTER
LDDP	-	LOAD DISPLAY PARAMETER REGISTER
LDTI	-	LOAD TEXT INCREMENT REGISTER
LDXA	-	LOAD X ABSOLUTE
MVYA	-	MOVE Y ABSOLUTE

Figure 6.2-2 Schedule Entry Structure

6.2 SCHEDULE TABLE STRUCTURE (CONTINUED)

scan the existing table to find the highest enabled priority (ignoring priority 127). Any blocks with priorities less than highest-3 are then excluded from the current image. Finally, those blocks with sufficiently high priority are linked together using the Graphic Controller jump address contained in word 11. In summary, the Schedule Table represents the Graphic Controller command mechanism and also the Display Scheduler monitoring scheme. Normally this structure will be completely transparent to the user tasks.

6.3 DISPLAY SCHEDULER MONITOR REQUESTS

User tasks will communicate with the Display Scheduler via the EMT monitor calls discussed below. The parameter block is the same for all calls, as discussed in Section 6.1.

6.3.1 Schedule Element

This service will be used to bring a display element into the current image, subject to the given priority. (Priority 127 may be used if the element is to be displayed regardless of the current highest priority schedule entry.) The associated schedule table entry address is returned in Word 1 of the parameter block.

EMT Code: 51 (Hex)

Parameter Block:
- Standard Scheduler Control Block

Status Return Codes:*

0 - Request Satisfied
2 - Schedule Table Space Exhausted

*Parameter limit checks are not performed. All parameters are masked to ensure a valid value for the respective Graphic Controller instructions.

6.3.2 Delete Schedule Table Entry

Delete will remove a schedule table block and deallocate the associated memory space. Note that all schedule table entries referencing a particular display element should be deleted prior to deleting the element.

EMT Code: 52 (Hex)

Parameter Block:
- Standard Scheduler Control Block

Status Return Codes:
 0 - Request Satisfied
32 - Control Block Not Found

6.3.3 Enable Schedule Table Entry

Enable will allow the current display of a table entry subject to its relative priority. The enable/disable state is indicated by the sign of the priority state. The respective flag in the user parameter block is also updated.

EMT Code: 53 (Hex)

Parameter Block:
- Standard Scheduler Control Block

Status Return Codes:
 0 - Request Satisfied

6.3.4 Disable Schedule Table Entry

Disable will remove a schedule table entry from the Graphic Controller execution list. This is indicated by making the priority field negative. With regard to the priority scheme, disabled entries are effectively non-existent.

EMT Code: 54 (Hex)

Parameter Block:
- Standard Scheduler Control Block

Status Return Codes:
0 - Request Satisfied

6.3.5 Modify Schedule Table Entry

The Modify service will allow for efficient update of selected schedule entry attributes. Modify attributes are selected by the bits of word 9 in the parameter block as follows:

BIT	ON IMPLIES
0	Update CRT Assignments
1	Change Intensity
2	Change Line Structure
3	Change Character Size
4	Change Character Rotate
5	Change Blink
6	Change X-Start
7	Change Y-Start

The new values for any modify requests are simply taken from the assigned parameter block locations. Note that changes to any parameters not listed above (color, priority, etc.) must be done via a delete/re-insert sequence.

EMT Code: 55 (Hex)

Parameter Block:
- Standard Scheduler Control Block

Status Return Codes:

0 - Request Satisfied

6.3.6 Purge Schedule Table

The Purge entry may be used to delete all schedule table entries belonging to the calling task. This, for example, should be done prior to task exit.

EMT Code: 56 (Hex)

Parameter Block: N/A

Status Return Codes:
0 - Request Satisfied

6.3.7 Graphic Controller Status (7)

The Status entry may be used to determine the X,Y beam coordinates at any point in a refresh file. This service operates by replacing the specified location by a halt instruction and then fetching the X,Y coordinates when the halt interrupt occurs. Thus, the X,Y position will not include the effects of the word at the specified display element offset.*

EMT Code: 57 (Hex)

Parameter Block:

This service operates through an extension to the standard scheduler control block, as follows:**

WORD	USAGE
10	Reserved
11	Element offset
12	X coordinate (returned)
13	Y coordinate (returned)

Status Return Codes:

0	-	Request satisfied
32	-	Block not active

Status bit three will be used to indicate the service is busy. Users will need to monitor this bit to ascertain request completion. Note also that words 12 and 13 are used by the monitor while the status request is active.

* The offset <u>includes</u> the element control words.

** Register one should, as always, point to word zero.

6.4 DISPLAY ERROR HANDLING

The two display interrupts corresponding to display halt and display out-of-bounds are serviced by the Display Scheduler.* If either of these events occur, the following procedure is followed:

a. The parent schedule table entry is located by an examination of the Graphic Controller Stack Pointer.

b. The element call is "nulled out" and appropriate status bits are set in the user's Schedule Control Block.

c. The Graphic Controller is restarted. The user task will need to delete the associated schedule table entry - enable will not be effective.

* Photopen handling is discussed in a separate monitor section.

6.5 MONOCHROME SCHEDULER

A second version of the Display Scheduler has been generated to provide for the monochrome display situation. The monochrome scheduler is similar to the color version in most respects. The major difference is in the area of scheduling priority. In the color case, a single priority scheme applies to all current entries, i.e., the priority is not independent for each color or CRT. For the monochrome case we have taken the list structure depicted in Figure 6.2-1 and used a separate list for each CRT. The priority value (specified in your parameter block) is now applied to the particular list indicated by your CRT specification. For the case of multiple CRT's specified for one schedule entry a default list assignment must be made. The list assignments for each possible CRT combination are given by Table 6.5-1.

Other special characteristics of the monochrome scheduler are as follows.

a) The CRT assignment byte in the caller parameter block is now extended to include 4 CRT's:

BIT	SET YIELDS
0	CRT 1
1	CRT 2
2	CRT 3
3	CRT 4

b) The Color byte in the caller parameter block is ignored,

c) There are no parameter limit checks (associated status return codes are not used),

d) "Modify" not supported for CRT assignments byte.

CRT BYTE (binary)	EXECUTION LIST ASSIGNMENT
0000	1
0001	1
0010	2
0011	1
0100	3
0101	1
0110	2
0111	1
1000	4
1001	1
1010	2
1011	1
1100	3
1101	1
1110	2
1111	1

(In other words, the schedule request is assigned to the list of the lowest numbered CRT selected.)

TABLE 6.5-1 EXECUTION LIST ASSIGNMENTS

GRAPHIC 7 MONITOR

INPUT-OUTPUT SERVICES

7.0 I/O SYSTEM STRUCTURE

Input/output processing generally represents one of the more difficult areas of computer operating system endeavors. This difficulty arises from the diversity of devices and required operating modes. The current system implementation is no different in this regard; however, the assumption of mostly dedicated applications will permit some simplifying assumptions. One of these assumptions is that the number of peripheral devices is relatively small, thus allowing a rather de-centralized approach to device handler design. An additional assumption is that there will generally be little device sharing between tasks. This assumption and the lack of any local mass storage releases us from any serious I/O queueing requirement. The Monitor I/O structure is depicted in Figure 7.0-1.

It is likely that users will want to add specialized devices/interfaces to the existing monitor configuration. For this reason we present, in the following, a discussion of the basic I/O monitor functional area. As mentioned above, our approach has been to de-centralize the I/O requirements, thus leaving considerable freedom to the individual device drivers. In fact, all the central I/O section does is determine which handler is to receive the request and then transfer control to such. Within the individual handlers, we have defined some standardized procedures which will make life easier for application task developers. In particular, a standard set of entry point assignments has been defined. Also, some attempt has been made to be consistent with respect to status bit assignments and error return codes. These standards are listed in the next section.

Users wishing to implement their own device handlers will be advised to follow the conventions which have been established even though the overall Monitor operation is little affected by your decisions. Addition of device handlers to the Graphic 7 Monitor is discussed in Appendix B.

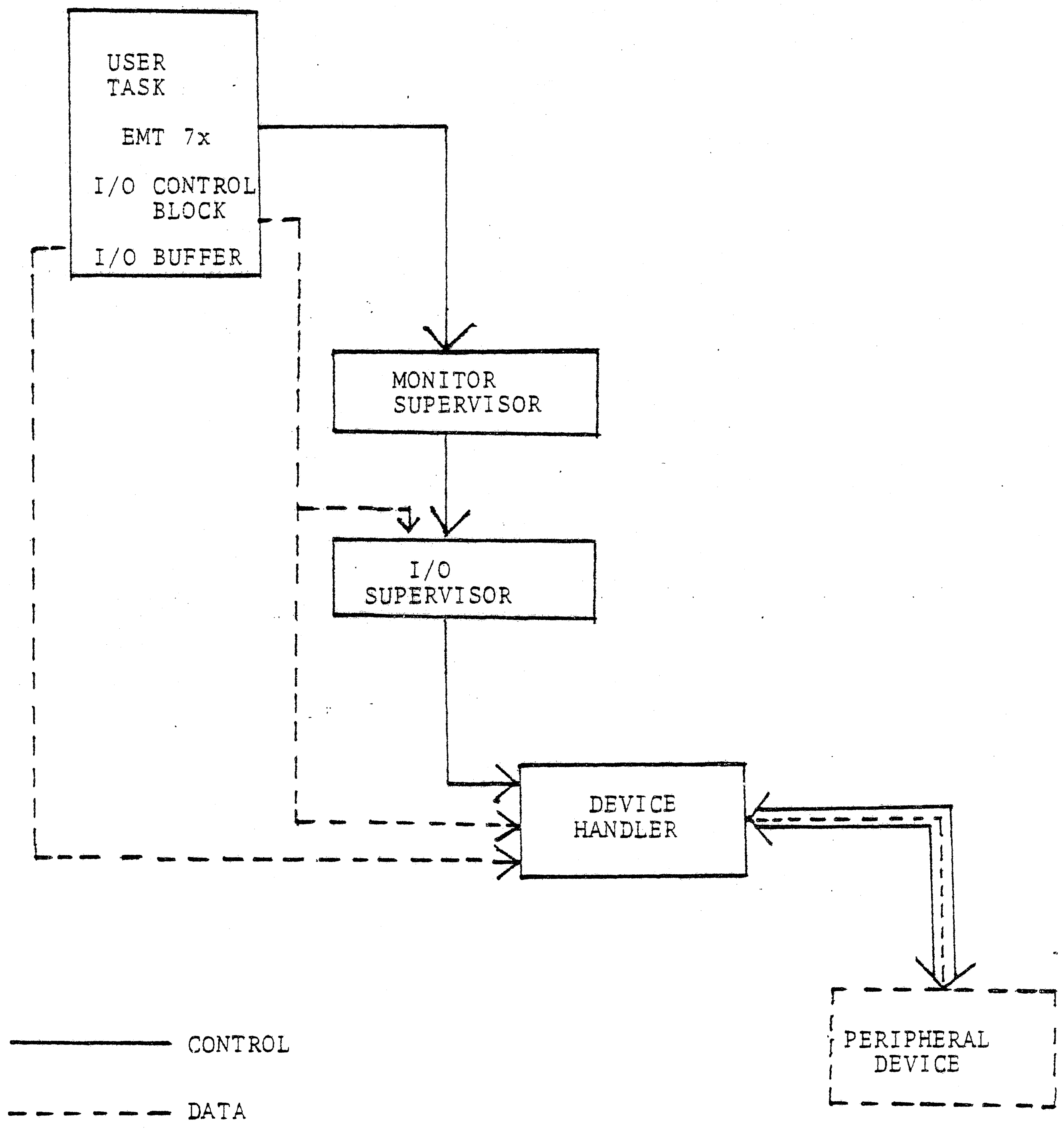


FIGURE 7.0-1. G7 MONITOR I/O STRUCTURE

7.1 USER TASK I/O

With respect to I/O from application tasks, users will find the facilities relatively conventional and easy to manipulate. I/O may be issued in either wait or no-wait mode with appropriate bits to indicate current I/O transfer status. The Standard I/O Control Block, discussed below, will be used to communicate all I/O requests to the monitor. The address of the parameter block should be placed in register one prior to the EMT instruction, the same as in other monitor calls. Specific EMT codes are discussed in Section 7.2.

7.1.1 Standard I/O Control Block

Figure 7.1.1-1 outlines the I/O Control Block structure. The first word (word 0) is reserved for monitor usage and should never be utilized by the application task. Other fields in the I/O block are discussed below. Keep in mind that the specific I/O operation desired is encoded in the EMT instruction. Thus, for some operations the whole I/O block is not needed although it will generally encompass a full eight words. Note that an I/O control block is dedicated to a particular data transfer as long as such transfer is in progress.

7.1.1.1 I/O Control Block, Word 1

Word 1 will contain the device specifier. This is in the form of a logical unit number which is used as an index into the I/O table in the task header. (See Section 4.3). The logical unit number should be an integer in the range one to eight. The high order byte of word 1 is not currently used (check individual device handler).

7.1.1.2 I/O Control Block, Word 2

The user buffer address should be placed in word two of the I/O block. Note that some devices require a buffer which is of fixed length or slightly larger than the expected transfer.

WORD	BYTE 1	BYTE 0
0	RESERVED	
1	UNUSED	LOGICAL UNIT NUMBER
2	BUFFER START ADDRESS	
3	TRANSFER LENGTH (Bytes)	
4	TRANSFER FLAGS	STATUS FLAGS
5	ERROR RETURN ADDRESS	
6	UNUSED	TIME OUT (SECONDS)
7	DEVICE SPECIFIC	

FIGURE 7.1.1-1. I/O CONTROL BLOCK STRUCTURE

7.1.1.3 I/O Control Block, Word 3

The expected number of bytes to be transferred should be placed in Word 3. Obviously, the buffer address specified in Word 2 should point to an area of sufficient capacity to allow the expected transfer. This should normally be specified as a byte count even though some devices may transfer only words.

7.1.1.4 I/O Control Block, Word 4

The various status bits controlling the I/O transfer and completion code reside in Word 4. The low order byte is used for device status returns (when requested). The high order byte is used to indicate the desired type of transfer and resultant transfer execution status. The bit assignments are as follows:

Transfer Flags		Device Status Flags	
	On Implies		On Implies
15	No-wait	7	
14	Busy*	6	
13	Device	5	
12	Dependent	4	
11	Transfer Options	3	Busy
10		2	Attached
9	Completion	1	Allocated
8	Code	0	On/Off line **

* Used for no-wait transfers only.

** This information is not always available.

7.1.1.5 I/O Control Block, Word 5

An optional error return address may be provided in Word 5. This has meaning for wait I/O mode only. If the error return point is exercised, the user may query the completion code to determine the specific fault. One should note that this optional error return applies only to the data transfer itself and not to possible rejection of the I/O request.

7.1.1.6 I/O Control Block, Word 6

An optional time out value, in seconds, may be placed in the low order byte of Word 6. This is considered to be a positive number from one to 255*. If time out is not desired, then a zero should be placed in said byte. The high order byte is currently unused.

7.1.1.7 I/O Control Block, Word 7

Word 7 is reserved for device-specific usage. For some devices, it is not needed at all; others may only use it for certain operations. Check the individual device descriptions in Section 7.3

7.1.2 Standard Status Return Codes

Every monitor function returns a status code to indicate the monitor's response to the user task's request. For I/O operations, this status code only indicates whether or not the request was accepted. Proper completion of the I/O transfer must be verified by examining the completion code

* The accuracy of the time interval is zero to minus one second.

in Word 4 of the I/O Control Block. With regard to device handler design, the following standard return codes should be used whenever possible:

Status Return Code (decimal)	Meaning
0	Request accepted.
4	Requested function not available.
5	No such device
33	Device not available or device busy.
34	Device not allocated/attached to calling task.
35	I/O operation completed prior to receipt of cancel request.
36	Resource Exhaustion.

Standard I/O completion codes (contained in bits 8, 9, 10 of Word 4 of I/O block) are as follows:

Completion Code	Meaning
0	Successful I/O transfer.
1	Buffer overflow.
2	Transmission error.
3	
4	I/O was cancelled.
5	
6	Time out.
7	

7.2 MONITOR I/O SERVICES

A group of standard monitor services has been defined which will correspond to entry points available in each device handler. As with other monitor functions, the desired service is encoded in the EMT instruction as the sub-function number. Some handlers may not support all of the services listed below due to logical or physical constraints. However, adherence to the standard entry numbers when designing new device handlers will improve the overall system image. In addition and of more practical significance, switching between devices via the logical unit numbers will be greatly simplified if all handlers use the same function codes. One may also note that several additional codes are available for use with specialized devices or for other user-specific requirements.

7.2.1 Allocate Device

The allocate service may be used to assign control of a device to a particular user application task. This service is used when the user wishes to issue monitor-independent I/O. Any interrupts received by the monitor while the device is allocated are ignored. The only way in which the device can be returned to general monitor usage is to have the task issue a de-allocate request.

EMT Code: 71 (Hex)

Parameter Block:
Standard I/O Control Block;
only the logical unit number is relevant.

Status Return Codes:

0	- Request satisfied
4	- Function not supported
33	- Device not available

7.2.2 Deallocate Device

The deallocate service should be used when the task has completed its operations. Devices should not be kept in the allocated state when not needed.

EMT Code: 72 (Hex)

Parameter Block:

Standard I/O Control Block;
only the logical unit number is relevant.

Status Return Codes:

0	- Request satisfied
4	- Function not supported
34	- Device not allocated or not allocated to the calling task.

7.2.3 Attach Device

The attach service will be used when an application task wants to reserve a device for its exclusive use. This differs from allocate in that attach implies usage of standard monitor I/O. Attach will be used when a task wishes to guarantee access to a device for a period of time, spanning several I/O operations.

EMT Code: 73 (Hex)

Parameter Block:

Standard I/O Control Block;
only the logical unit number is relevant.

Status Return Codes:

0	- Request satisfied
4	- Function not supported
33	- Device not available

7.2.4 Detach Device

Detach will be issued when the application task wishes to relinquish control of the subject device. This makes the device available for use by other tasks.

EMT Code: 74 (Hex)

Paramater Block:

Standard I/O Control Block;

only the logical unit number is relevant.

Status Return Codes:

0 - Request satisfied

4 - Function not supported

33 - Device busy

34 - You're very mixed up

7.2.5 Read Data

The Read service will be used to transfer data from an external device to a program-local buffer. The amount of data transferred is determined by the transfer count in word 3 of the I/O control block. Check the particular device handler description in Section 7.3 for the exact functioning of this servie.

EMT Code: 75 (Hex)

Paramter Block:

Standard I/O Control Block

Status Return Codes:

0 - Request satisfied

4 - Function not supported

33 - Device not available

7.2.6 Write Data

Write will transfer data from a user buffer area to an external device. There will commonly be a variety of device-specific charactersitics in relation to this service.

EMT Code: 76 (Hex)

Parameter Block:

Standard I/O Control Block

Status Return Codes:

0 - Request satisfied
4 - Function not supported
33 - Device not available

7.2.7 Control Device

The Control service would be provided with devices which require setup prior to data transfer. Some devices, such as programmable switches, may support only the Control function.

EMT Code: 77 (Hex)

Parameter Block:

Standard I/O Control Block

Status Return Codes:

0 - Request satisfied
4 - Function not supported
33 - Device not available

7.2.8 Fetch Device Status

This service will return the current device/handler status to the low order byte of word 4 in the specified I/O control block. Standard bit assignments are given in Section 7.1.2. Other bits may vary between devices.

EMT Code: 78 (Hex)

Parameter Block:

Standard I/O Control Block

Status Return Codes:

None

7.2.9 Cancel I/Ø

Cancel may be used to prematurely terminate an I/Ø transfer.

EMT Code: 79 (Hex)

Parameter Block:

The I/O block related to the transfer to be cancelled.

Status Return Codes:

0 - Request satisfied

35 - I/O has already completed

7.2.10 I/O Purge

The Purge entry will be used by the Graphic/7 Monitor to ensure that a task which is being aborted (or is doing a voluntary exit) does not have any outstanding I/O in process. If the handler is doing I/O for the subject task, it is cancelled. In addition, any attach or allocate conditions, if relevant to the subject task, are erased. This entry will normally be of no interest to user tasks.

EMT Code: 7E (Hex)

Parameter Block: N/A

(RR1 is assumed to point to the task being aborted.)

Status Return Codes: None

7.2.11 File Query

The status of a data file may be determined via the File Query entry point. This entry may not be supported by some device handers or may be supported in modified form. For the entry, the user buffer area has the following specific format:

Word	Contents
0	File Status/File Type
1	File Length (Records)*
2	Record Length (Bytes)
3 - 7	File Name (ASCII, terminated by null)

The exact meaning of the above parameters is dependent on the particular device and handler implementation.

EMT Code: 7F (Hex)

Parameter Block:

Standard I/O Control Block

Status Return Codes:

0 - Request satisfied

33 - Device not available

* For task images and display elements, the number of records should be one with the number of bytes equal to the total amount of data.

7.3 Graphic/7 Device Handlers

Following sections outline specific capabilities of each device driver. Users should note particular characteristics of each device in regard to entry points, transfer options and completion codes. Entry points listed for each handler do not include 0 (initialization) and 14 (I/O Purge) since these must be provided for all handlers.

7.3.1 Alphanumeric Keyboard

The keyboard will be used to input alphanumeric data and function key requests. The entry points supported are:

EMT	Function
71	Allocate
72	Deallocate
73	Attach
74	Detach
75	Read
78	Fetch status
79	Cancel I/O

The following transfer options are supported with the Read function:

Bit	Usage
11	Accept only function key input
12	Mask out parity bit
13	Accept only alphanumeric input

Status bits related to the keyboard device handler are:

Bit	On Implies
1	Allocated
2	Attached
3	Busy

With regard to the user buffer area, the following points should be noted:

a) The count of the number of characters input (not including the carriage return) is placed in word 7 of the user I/Ø control block.

b) Except in cases of buffer overflow, the carriage return will always appear in the user buffer.

7.3.2 POSITION ENTRY DEVICE (PED)

The PED handler will service either a trackball or joystick device. The handler is set up to be used in conjunction with a cursor or other screen-position-relevant entity. Thus, the handler uses the delta x, delta y values returned by the device to update virtual x,y position coordinates. These coordinates may be given to the user task as binary integers or in the form of LOAD-X/MOVE-Y refresh instructions. Furthermore, the data may be provided on a "one-time" basis or via the notify service (discussed below). Finally, the actual PED x,y delta values may be supplied by choosing one of the option bits listed below.

The handler entry points supported are:

EMT	FUNCTIONS
71	Allocate
72	Deallocate
75	Return current X,Y position
76	Set current X,Y position
77	Set X,Y to 0,0
78	Fetch status
7A	Define notify requirement
7B	Cancel notify

The following transfer options are supported:

BIT	USAGE
13	Setting this bit will cause the X,Y coordinates to be returned as LOAD-X,MOVE-Y (absolute) refresh instructions.
12	This bit will be set by the handler whenever new x,y values are supplied by the notify service.
11	Setting this bit will cause the actual x,y delta values to be returned as received from the PED. These are signed integer words. This option is only relevant to the notify service.

Status bits related to the PED handler are:

BIT	ON IMPLIES
1	Allocated
2	Notify Active

Several points should be noted in regard to the PED device/ handler:

a. The notify service will return the new X,Y position each time new coordinates are received from the PED device. As with entry 5, these coordinates may be binary integers or refresh (load X, move Y) instructions. Bit 12 in the user I/Ø transfer flags will be set whenever new X,Y values are supplied. The user may also, optionally, specify a subroutine address in word 7 of his I/O control block. This subroutine will be called (JSR PC,XXXX) whenever a new PED X,Y is supplied. The subroutine will need to be kept reasonably short. Furthermore, any registers used must be saved/restored and no monitor services may be employed. If the optional subroutine is not desired, then a zero should be placed in word 7 of the I/O control block.

b. The data buffer used with the PED device should be exactly two words in length and begin on a word boundary (X in first word, Y in the second).

c. All transfers related to the PED handler are satisfied immediately, thus, the wait/no-wait transfer flag is ignored.

d. Entry points 6 & 7 assume 16 bit binary X,Y values.

e. Time out and optional error return specifications are not supported.

f. Bytes 0, 1 and 3 of the user I/O control block are used for PED notify linking.

g. On calling the user specified subroutine, the following register situation will exist:

   1) User relocation registers are installed
   2) R0,R1 contain binary X,Y
   3) R2,R3 contain LØADX,MØVEY
   4) R4 points to user I/O control block
   5) R5 points to user subroutine
   6) R6 - system stack pointer

   Again we emphasize that this state must be preserved across the call.

7.3.3 Paper Tape Reader

The Paper Tape handler is configured for reading task images or other data in file format. The handler is not a general purpose capability. The expected file format is that employed in standard PDP paper tape load images. The general form is:

```
        .
        .
        .
        header block
        .
        .
        .
        data block(s)
        .
        .
        .
        start block
        .
        .
        .
```

The header block has format:

byte	contents (octal)
0	1
1	0
2	32 (block length)*
3	0
4, 5	Start Address (typically the same as the program load point)
6	1
7	7
10, 11	Program load point
12, 13	Total program length (bytes)

*In all cases, the block length includes everything except the checksum byte. The checksum is the negative of the accumulated preceding bytes in the block.

byte	contents (octal)
14, 15	Program Transfer Address
16-31	Unused
32	Checksum

Data blocks have the general form:

byte	contents
0	1
1	0
2, 3	block length (n)
4, 5	load address
6	data bytes
.	.
.	.
.	.
n	checksum

The trailer block is a data block with no data bytes, the program transfer address as the load address and a block length of six. Tasks to be loaded by the GRAPHIC 7 monitor should not have a specified transfer address (argument in the .END statement) since this specification is obtained from the task header. In this case the transfer address in the start block appears as unity.**

**If examining a paper tape recall that the low order byte of a word (the "right" byte) comes first on the tape.

The entry points supported by the paper tape handler are:

EMT	Function
75	Read
7F	File Query

All transfers are assumed to be binary data. Wait/No-Wait and I/O time-out are supported in the normal manner.

I/O status returns of the paper tape handler are:

0 -	I/O success
1 -	Checksum error
2 -	Transmission error
3 -	Invalid file format
6 -	Time out

NOTES:

a. For task loads, the target address is taken from the data block on the tape. This is necessary since the tape image does not typically reflect space allocated by .BLKB or .BLKW directives. For this reason, all task images must be linked to start at 20000 (octal).

b. For data input, the user generally wants the information deposited at an address which he specifies in his I/O control block. To accomplish this, have the header punched with a zero in bytes 10, 11. This flag causes the handler to load successive data bytes into successive locations in the user buffer.

7.3.4 4923 Cassette Handler

The Tektronix 4923 cassette drive is a general purpose data storage device utilizing the 3M DC-300 data cartridge. This device is ASCII oriented with all data stored as ASCII characters and several device control functions activated by character codes. The monitor cassette handler is a minimal facility allowing recording and fetching of task images or data files.

All data stored on the cassette cartridge is in the form of a file. Multiple files may be stored on the same cartridge. The basic file format employed is:

character	contents
0	M
1	U
2,3,4,5	Data load address, stored as four ASCII, hex digits; most significant 4 bits first,
6,7,8,9	File length as 4 ASCII hex digits, this is total data words
10,11,12,13	Data Word 1
14,15,16,17 etc.	Data Word 2

Files are generally terminated by a STOP character, octal 23. This causes the cassette to pause until receiving a START character, octal 21, from the device handler.

Entry points supported by the cassette handler are:

EMT	Function
75	Read
76	Write
7F	File Query

All data retrieved from the cassette must be via a file query followed by a file read. The write entry supports the following options: (word 4 of the I/O block)

Bit	On Implies
13	Data words are output as 4 ASCII hex digits. Otherwise data words are output directly as two bytes, assumed to be valid ASCII characters.

I/O status returns for the cassette handler are:

0 - I/O completed successfully

2 - Transmission error

3 - Invalid file format

6 - Time out

NOTES:

a. The I/O count for the cassette must be an even number of bytes.

b. I/O time-out and Wait/no-wait options supported in normal manner. Optional error return (word 5 of I/O block) is not implemented.

c. There is no checksum in the file format, however valid ASCII hex codes are ensured on reads.

d. A small delay may be necessary between the reads of successive files.

GRAPHIC 7 MONITOR
TIMING SERVICES

8.0 TIMING SERVICES OVERVIEW

The maintenance of current real time is always an important, though usually transparent, feature of computer operating systems. Current time is useful in many applications for providing a reference frame for various ongoing activities. Also, the clock is used to measure timed intervals for application task functions. On the Graphic 7, the fundamental clock pulse originates from the 60 cycle power source and thus the timing precision is to the nearest 1/60 second. The interrupt handler for the basic clock pulse will be contained within the Timing Services module. This handler will update the current stored time-of-day and adjust any timers currently outstanding. There are three distinct types of timers as discussed in the following sections.

8.1 TASK SUSPENSION TIMERS

User tasks may request timed suspension for periods up to 53 minutes. These requests, submitted to the Task Management section, will be specified in 1/10 second units.* The Timing Services section will maintain a linked list of all tasks currently under timed suspension. On each 1/10 second boundary, the suspension list is scanned and all associated counters decremented. When a counter reaches zero, the respective task is removed from the timer control list and again made available for execution. Timing services related to task suspension are normally employed only by other monitor functions.

*The accuracy of these timers is plus zero to minus one 1/10 second.

8.2 I/O SUSPENSION TIMING

The problem of an un-responsive external device is typically handled by allowing a certain maximum time interval for an I/O transfer. Timers used for this problem are updated, on a one second basis, by the timer interrupt handler.* All I/O timers are maintained on a linked list, with individual nodes within the various device handlers. It should be noted that such timers are assumed to be within the direct address space of the monitor.

*The accuracy of these timers is plus zero to minus one second.

8.3 PRIVATE TIMERS

It is sometimes necessary for an application task to perform a function on a timed interval basis. This capability is provided by defining a private timer which is updated on a 1/10 second interval.*

Such private timers are maintained on a linked list within the user task space. Another list is then used to link all tasks which currently have outstanding private timers.

As an option with the private timer service, users may specify a subroutine which will be executed on timer expiration. This subroutine will be executed as an extension of the clock interrupt handler. Several restrictions must be placed on such a routine as a result of the execution context. Thus, such a subroutine must be kept very short (less than 100 usec execution time), cannot reference any monitor services, and will save/ restore any registers used. Obviously, excessive use of this service could potentially degrade system performance.

Private timers are re-settable within the optional user subroutine. Thus, if the user places a non-zero value in the timer entry before returning to the clock handler, then the timer is not deleted. This mechanism can be used to ensure that a particular event occurs at a fixed repetition rate.

* The accuracy of these timers is plus zero to minus one 1/10 second.

8.4 MONITOR TIMING SERVICES

Timing services subfunctions are discussed in the following sections.

8.4.1 Set Date/Time

This service may be used to initialize the current date and time. The Operator Communications module will use this entry when processing the related operator command.

EMT Code: 81 (Hex)

Parameter Block:

Word	Contents
0	Hours/Minutes (binary)
1	Seconds/Month (binary)
2	Day/Year (binary)

Status Return Codes:

0 = Request satisfied (The validity of the supplied values is not checked).

8.4.2 Fetch Date/Time

The Fetch service will return the current date and time to the caller parameter block.

EMT Code: 82 (Hex)

Parameter Block:

Word	Contents
0	Hours/Minutes (binary, returned)
1	Seconds/Month
2	Day/Year

Status Return Codes:

0	Request satisfied

8.4.3 Define Task Suspension Timer

This service is used by the Task Management module to set up for timed task suspension. The words at T$TIM and T$TIM+2 within the task header are used to maintain the task suspension timer.

EMT Code: 83 (Hex)

Parameter Block: N/A

Status Return Codes: None

8.4.4 Cancel Task Suspension Timer

Cancel will be used by Task Management when a "Continue" request is received for a task which is currently in timed suspension.

EMT Code: 84 (Hex)

Parameter Block: N/A

Status Return Codes: None

8.4.5 Define I/O Timer

This entry will be used by device handlers when the user task specifies a time-out value in his I/O control block. The parameter block is assumed to reside in direct address memory space. The second word in the parameter block should point to the handler entry point which will be called on timer expiration. The call, at priority seven, will be of the form JSR PC,xxxx. The handler's Resume entry shall not destroy registers 4 and 5.

EMT Code: 85 (Hex)

Parameter Block:

Word	Contents
0	Reserved (used for linking)
1	Resume Entry Address
2	Time-out Value (seconds)

Status Return Codes:

0 = Request satisfied

8.4.6 Cancel I/O Timer

Cancel should be used by I/O handlers when an I/O operation is completed prior to expiration of the time-out value.

EMT Code: 86 (Hex)

Parameter Block:

- Same as used for EMT 85

Status Return Codes:

0 = Request satisfied

8.4.7 Define Private Timer

User tasks may set up a timed interval via this Timing Service entry. The time interval value should be a positive binary number in 1/10 second units. The second parameter is an optional user subroutine address which is called when the timer expires. This subroutine may not reference any monitor services and must save/restore any registers used. Total execution time of this optional subroutine should be held to 100 usec. If the subroutine call is not desired, then a zero should appear in the second parameter. Note that the timer value is destroyed; however, it may be reset within the optional user subroutine.

EMT Code : 87 (Hex)

Parameter Block:

Word	Contents
0	Reserved (Linking)
1	Optional Subroutine Address*
2	Timer Value (1/10 seconds)

Status Return Codes:

0 = Request satisfied

*The call is of the form: JSR PC,xxxx.

8.4.8 Cancel Private Timer

If a private timer is no longer required, then it should be deleted with this service. The specified parameter block should be exactly the same as that used in the definition call to EMT 87.

EMT Code: 88 (Hex)

Parameter Block:

- Same as used for EMT 87

Status Return Codes

0	= Request satisfied
32	= No such timer found

8.4.9 Purge Task Timers

The purge service will be used by Task Management to ensure that all private timers are eliminated on task exit. A user task could use this entry to remove all currently defined private timers.

EMT Code: 89 (Hex)

Parameter Block: N/A

Status Return Codes: None

GRAPHIC 7 MONITOR

DATA TRANSFER SERVICES

9.0 DATA TRANSFER INTRODUCTION

The efficient manipulation of data within the real t
system is, naturally, a prime objective of the Graphic 7
monitor. Within the context of the monitor there exists
three fundamental mechanisms for the transmission of int
task data. (Input/output functions are used to transmit
accept data to/from an external medium). The simplest f
of inter-task data exchange is based on the task status
contained in the task header. These status bits and ass
ciated manipulative services are discussed in the Task M
ment area. Generally, the status bits will prove satisf
for those situations which require the notification of c
states or operating conditions. The second form of inte
task data exchange may be set up on the basis of a globa
mon data area. Such areas are valuable when several tas
must have frequent access to the same information. An e
would be the system data base which typically contains t
various parameters and data defining the current operat
situation. The third form of inter-task data communicat
is the subject of the monitor service described in the f
lowing.

The Data Transfer Services will allow for the trans
of variable length data blocks between task memory areas
Specification of the desired transfer is done via a star
format parameter block, similar to an I/O control block.
In order for a transfer to take place, there must be a r
ing sender/receiver pair. For the receive entry, the us
may indicate a specific sender task or allow for accepta
from "any sender". Senders must specify a particular re
task (i.e., no "broadcast" mode is supported). If no ma
ing transfer is pending, the user may elect to have his
quest queued or stacked. Separate send and receive queu
are maintained: queueing results in attachment of the r
to the end of the list; stacking results in attachment t

beginning of the list. In addition, users may, as in I/O operations, specify wait or no-wait transfer mode. If wait mode (the default) is used, then the requesting task is suspended until the transfer is completed.

The actual data transfer is performed, byte-by-byte, from the sender task area to the receiver task area. The receiver buffer must be large enough to accommodate the sending task's requirement. Insufficient buffer space results in setting status bit zero in the sender if the receiver request was already pending. Conversely, if the send request was already pending, then the error status is returned to the receiver on inadequate receiver buffer length. An error code will also be returned if neither queueing nor stacking is specified and no matching transfer is pending. Potential data receivers may query the send queue to determine if any outstanding transfers are pending. The sender task ID and transfer length are returned. The standard Data Transfer parameter block is discussed in the following section.

9.1 DATA TRANSFER CONTROL BLOCK

All requests for inter-task data transfer will be accompanied by a pointer (in register one) to a standard parameter block as shown in Figure 9.1-1. Users will note the similarity of this structure and the I/O control block discussed in Section 7.1. In particular, note that the transfer may be performed in either wait or no-wait mode. In no-wait mode, users will need to be careful to ensure that the transfer has been completed before disturbing the control block or the data buffer.

Let us briefly review the individual words in the control block. Word 0 and byte 1 of word 1 are used for send/receive queueing and are of no concern to the application programmer. Byte 0 of word 1 should contain the destination task ID for send or the source task ID on receive. Additionally, the user may set this byte to zero for receives to indicate that any sender is acceptable. In this case, the monitor will place the sending task ID into this byte when a transfer occurs. Word 2 should contain the data buffer address. The transfer count, as a number of bytes, will appear in word three. Obviously, the maximum transfer length is effectively determined by the maximum task extent.* The Flags/Status bits are outlined in Figure 9.1-1. Clearly the Queue/Stack bits are mutually exclusive. The count of the number of bytes actually moved is placed in word 5. This word is only required for receivers.

*The Data Transfer mechanism does not detect departure from the task space. Such errors will generally be fatal.

WORD	BYTE 1	BYTE 0
0	Reserved	
1	Reserved	Task ID
2	Buffer Address	
3	Transfer Count (Bytes)	
4	Flags	Status
5	Bytes Transferred Count	

Flag Bits (set implies)		Status Bits (set implies)	
15	No-wait	7	
14	Busy	6	
13	Queue	5	
12	Stack	4	
11		3	
10		2	Request was cancelled
9		1	No sender/receiver
8		0	Inadequate buffer length

FIGURE 9.1-1 DATA TRANSFER CONTROL BLOCK.

9.2 DATA TRANSFER ENTRY POINTS

The various services provided under the heading of Data Transfer are discussed in the following sections. As with all monitor functions, the address of the applicable parameter block should be placed in register one prior to the EMT. With regard to these services, users should carefully note the difference between the monitor call status return and the completion codes returned in the status byte of the parameter block. These considerations are identical to the related I/O control functions in that the monitor request status (in byte T$MRST of the task header) only indicates acceptance of the request and does not imply successful data transfer.

9.2.1 Receive Query

The Receiver Query entry may be used by a prospective receiver task to determine if any outstanding send requests exist. The task ID (byte 0 of word 1) may specify a particular sending task or may be zero to indicate that any sender is acceptable. Discovered sender task ID is placed in this byte.* Sender transfer length is placed in word 5.

EMT Code: 94 (Hex)
Parameter Block:
Standard Data Transfer control block.
(Words 2 & 3 are ignored).
Status Return Codes:
0 - Request Satisfied

* Note carefully that, if said byte was given as zero, then this action results in somewhat destructive modification of your parameter block.

9.2.2 Receive Data

Receive will be used to accept data from another application task. If no send is pending, then the user may have his request placed on the receive queue. The no-wait bit may then be used to have control returned to the calling task.* The count of the number of bytes transferred is placed in word 5. Sender task ID is placed in byte 0 of word 1.

EMT Code: 95 (Hex)
Parameter Block:
Standard Data Transfer control block.
Status Return Codes:
0 - Request Satisfied.

* If neither queueing nor stacking is requested the wait/no—wait specification is ineffectual.

9.2.3 Send Data

Transfer of data from the calling task to a receiving task may be accomplished by the Send Data entry. The sending task must specify a target receiver task ID. If no receive is pending, then attachment of this request to the send queue may be specified. Again, wait or no-wait mode may be employed.

EMT Code: 96 (Hex)

Parameter Block:
Standard Data Transfer control block.

Status Return Codes:
0 - Request satisfied

9.2.4 Cancel Receive

Cancel may be used to terminate a pending receive request belonging to the calling task.

EMT Code: 97 (Hex)

Parameter Block:
The receive to be cancelled.

Status Return Codes:
0 - Request satisfied
32 - No such request in existence

9.2.5 Cancel Send

The Cancel Send entry may be used to remove a pending send request which was previously defined by the calling task.

EMT Code: 98 (Hex)

Parameter Block:
The send to be cancelled

Status Return Codes:
0 - Request satisfied
32 - No such request in existence

9.2.6 Purge Receive Requests

All Receive requests belonging to the calling task are removed from the Receive queue.

EMT Code: 99 (Hex)

Parameter Block: None

Status Return Codes:
0 - Request Satisfied

9.2.7 Purge Send Requests

All Send requests belonging to the calling task are removed from the Send queue.

EMT Code: 9A (Hex)

Parameter Block: None

Status Return Codes:

0 - Request Satisfied

9.2.8 Purge Requests

All Send & Receive requests belonging to the calling task are removed from the respective queues. This entry is exercised by Task Management on task exit or abort.

EMT CODE: 9E (Hex)

Parameter Block: None

Status Return Codes:

0 - Request Satisfied

GRAPHIC 7 MONITOR

MONITOR SERVICES

10.0 MONITOR SERVICES INTRODUCTION

The Monitor Services consist of a set of generally useful facilities which may be employed from user tasks. The exact content of this monitor functional area will depend on the particular application requirements and availability of monitor memory space. The various sub-functions within Monitor Services are easily accessed via the standard EMT calling sequence in combination with the appropriate parameter block.

10.1 Query Buffer Allocation

There arise circumstances in which the availability of direct-address-space memory is highly desireable. One such instance is the need for a buffer area for issuing a query/response sequence via the display. Direct address space is necessary here since such area must be accessible by all of the graphic controller, the user task and the keyboard device handler. The Query Buffer Allocation service is intended to satisfy this need by providing for the allocation, within the monitor memory area, of 128 byte working areas. These areas will typically be used with the Query/Response service discussed in Appendix C, however, other uses are possible. Users should be careful to release these buffers when they are no longer required. The following monitor call is used for both allocate and deallocate of a Query Buffer.*

EMT Code: 11 (Hex)

Parameter Block:

Word	Contents
0	If zero then the buffer starting address is returned. If non-zero, then deallocation of the associated buffer is assumed. On deallocation, this word is zeroed prior to return.

Status Return Codes:

0	- Request satisfied
32	- No buffers available

*Users may also want to note that small direct-access working areas may be provided via the schedule table space in memory block zero. Allocation/deallocation of this space, in fixed 12-word units, may be accomplished via Memory Management entry points 5 and 6. As with the Query buffers, users should be careful to deallocate this space when no longer needed.

10.2 Alphanumeric Decode Service

The conversion of text inputs to numeric values is a common requirement for problems involving operator interaction. The Decode service will provide for variable length conversions of octal, decimal and hexadecimal quantities. Specifications to this service consist of:

a. Argument count
b. Format string address
c. Source string address
d. Error return address
e. EØF return address
f. Variable addresses

The format string is of the form:

item,item,...,item/null

where "item" is one of: 0?,H?,I?,A?. The format string should be terminated with a zero byte.

The source string should be standard ASCII characters with the high order (parity) bit zero. If the first source string character is a carriage return, then the EØF return address is taken. If invalid characters are encountered, the error return address is employed.

The numeric variables may be either single or double words. If a double word qualtity, then preceed the respective format item with the letter "D".

Alphanumeric (A?) strings will be terminated with a null (zero) byte.

A count of the number of characters processed in the source string is returned in byte 1 (i.e., the second byte) of the parameter block.

The total number of arguments should appear as the first byte of the parameter block.

EMT Code: 12 (Hex)

Parameter Block:

Word	Contents
0	No. of arguments
1	Format address
2	Source-string address
3	Error return address
4	EØF return address
5	
.	Variable arguments
.	
.	
n	

Status Return Codes:

0 - Request satisfied
32 - Invalid format syntax or format/argument list mis-match

As an example of the Decode service, the following code may be used to extract the first argument in string INPUT.

```
INPUT:    .ASCII    /12,5/
FØRMAT:   .ASCIZ    /I?/
DBLK:     .WØRD     5,FØRMAT,INPUT,ERRØR,EØF,ARG1
ARG1:     .WØRD     0

MØV       #DBLK,R1
G7CALL    1,2
```

(ARG1 should equal 12_{10} on return.)

Note that the input string fields should be separated by commas and/or blanks and (typically) terminated by a carriage return.

10.3 DISPLAY DATA SUPPORT

Display Data Support allows for conversion of display elements from standard format to refresh format. This facility, accessed as a standard monitor service, is discussed in Appendix E.

GRAPHIC 7 MONITOR
PHOTOPEN SERVICES

11.0 PHOTOPEN SERVICES INTRODUCTION

The Graphic 7 PHOTOPEN may be used for list selection, display element editing and task control functions. One or two PHOTOPENS may be connected to the basic Graphic 7 mainframe. The Monitor PHOTOPEN Services will make this hardware resource available to user tasks for the various above mentioned usages. PHOTOPEN requests will bear a close relationship to schedule table entries as depicted in Figure 11.0-1.

To employ the PHOTOPEN, the user task must first have specified PHOTOPEN sensitivity in his schedule table control block (see Section 6.1). After scheduling, the PHOTOPEN request is made known via the "Define" service discussed below. This request may be issued in either wait or no-wait mode, similar to an I/O function. If issued in wait mode, then the relevant task status bit is set and task execution is suspended until the PHOTOPEN strike is received. In no-wait mode, the user will need to monitor the request status bits to ascertain request completion.

The functioning of the PHOTOPEN interrupt handler is as follows: On receiving the interrupt, the handler will search for the related schedule table entry by examining the Graphic Controller stack. The associated user Schedule Table Control Block is located via a scan of the task list and then a search of the parent task's Schedule Table Control Block list. A search is then made of the currently outstanding PHOTOPEN requests belonging to the subject task. If an associated PHOTOPEN control block is not found, then the interrupt is ignored. Assuming a control block does exist, the appropriate status bits will be set (corresponding to the CRT's activated) and the element offset is computed and placed in the user's PHOTOPEN control

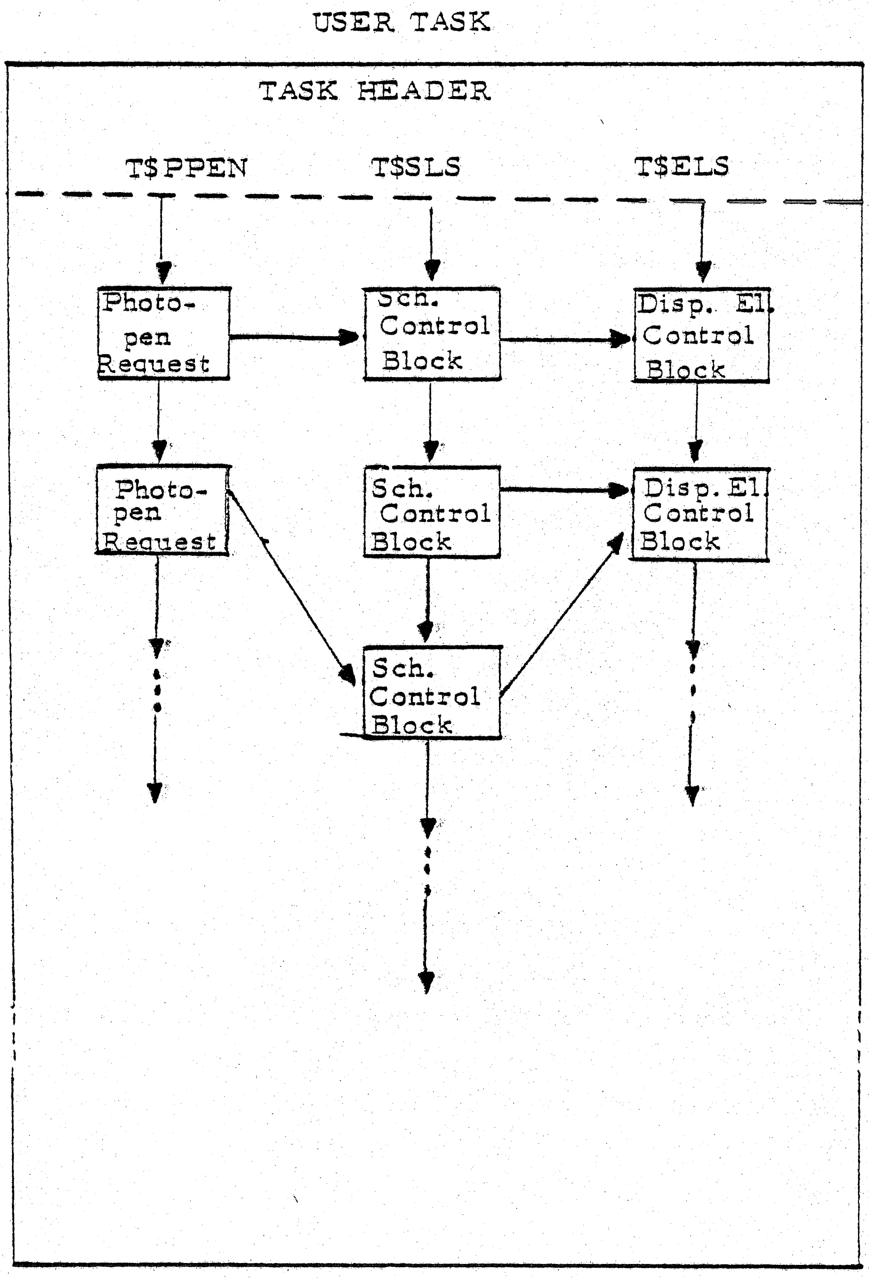


Figure 11.0-1 Display Control Block Associations

block. The Graphic Controller is then restarted. If the user has selected wait mode PHOTOPEN operation, then his task is again made eligible for CPU time.

The standard parameter block used with all PHOTOPEN monitor requests is outlined in the following section. Prospective users should carefully note the definitions of the various fields and operation of the PHOTOPEN related monitor functions.

11.1 STANDARD PHOTOPEN CONTROL BLOCK

All parameter blocks used with the PHOTOPEN services will assume the format shown in Figure 11.1-1. The first word is used for linking of outstanding PHOTOPEN control blocks. The second word should contain a pointer to the relevant schedule entry control block. (The PHOTOPEN routines assume the subject display has been scheduled. Note also that Display Scheduler functions have no deliberate effect on possibly related PHOTOPEN requests.) Word two of the PHOTOPEN control block contains flags and status conditions as outlined in Figure 11.1-1. Note in particular, the high order bit (15) which should be set to enable no-wait mode. The following difference between wait and no-wait mode must be remembered: In wait mode the user request (i.e. the PHOTOPEN control block) is automatically deleted from his list on receiving the PHOTOPEN input. In no-wait mode, the user is responsible for deleting the request via the PHOTOPEN delete service.

The actual PHOTOPEN input information consists of the offset (bytes) within the subject display element. This offset, which is with respect to the start address stored in the second word of the element control block*, is placed in word three of the PHOTOPEN control block. The offset will be odd if bit 5 of the Graphic Controller sense register was set.

*E.G. - includes the control words.

WORD	byte 1	byte 0
0	reserved	
1	Sch. Control Block Pointer	
2	Flags	Status
3	Element Offset (Returned)	

Flag Bits

15	No Wait
14	
13	
12	
11	
10	
9	
8	

Status Bits

7	Parameter Blk in Use
6	
5	
4	
3	
2	
1	Strike (PP 2)
0	Strike (PP 1)

Figure 11.1-1 Standard PHOTOPEN Control Block

11.2 PHOTOPEN MONITOR SERVICES

The various PHOTOPEN functions are accessed via the entry points discussed below. As customary for Monitor functions, the address of the applicable parameter block should be placed in register one prior to the EMT.

11.2.1 Define PHOTOPEN Request

The Define function will serve to place a PHOTOPEN request block on the user's PHOTOPEN request list. If wait mode (the default) is used, the request block will be deleted when the request is satisfied. When the strike occurs, the Display Element offset is placed in word three of the parameter block.

EMT Code: 61 (Hex)

Parameter Block:
Standard PHOTOPEN Request Block

Status Return Codes:*
0 - Request satisfied

*As in I/O operations, the Status Return Code only indicates acceptance or rejection of the monitor request; availability of the relevant data is not implied.

11.2.2 Delete PHOTOPEN Request

An outstanding PHOTOPEN request may be cancelled via the Delete entry. This entry (or the following Purge entry) will need to be used for elimination of no-wait requests.

EMT Code: 62 (Hex)

Parameter Block:
Standard PHOTOPEN Control Block

Status Return Codes:

0	Request Satisfied
32	No such request found

11.2.3 Purge PHOTOPEN Request

The Purge service may be used to erase all outstanding PHOTOPEN requests.*

EMT Code: 63 (Hex)

Parameter Block: N/A

Status Return Codes:

0 - Request satisfied

*The absence of any such request is not considered an error.

APPENDIX A

GRAPHIC 7 MEMORY MANAGEMENT

A.1 THE ADDRESSING PROBLEM

Being a 16 bit machine, the basic Graphic 7 has a direct addressing capability of 32K words. The evasion of this limitation has proved to be a major objective of similar machines and is currently the aim of the Graphic 7 Memory Management option. This hardware will allow users to address up to 128K words, or, effectively, an 18 bit address space. In as much as the basic word size (16 bits) has not changed, a function must exist for mapping a 16 bit address into an 18 bit realm. Obviously, the two extra bits must be supplied somewhere between the operand address and an actual memory reference. In the case at hand, processor re-design was ruled out, thus leaving the mapping function to the memory itself.

In addition to providing access to a greater volume of memory, mapping functions are also sometimes used to solve the problem of code relocation. In this role, the relocation "value" not only provides the additional addressing bits but also yields an address base. This results in all program-local addresses being just an offset with respect to the base value. Finally, the combination of address relocation and extension is sometimes central to the system protect mechanism.

A.2 GRAPHIC 7 MEMORY MANAGEMENT

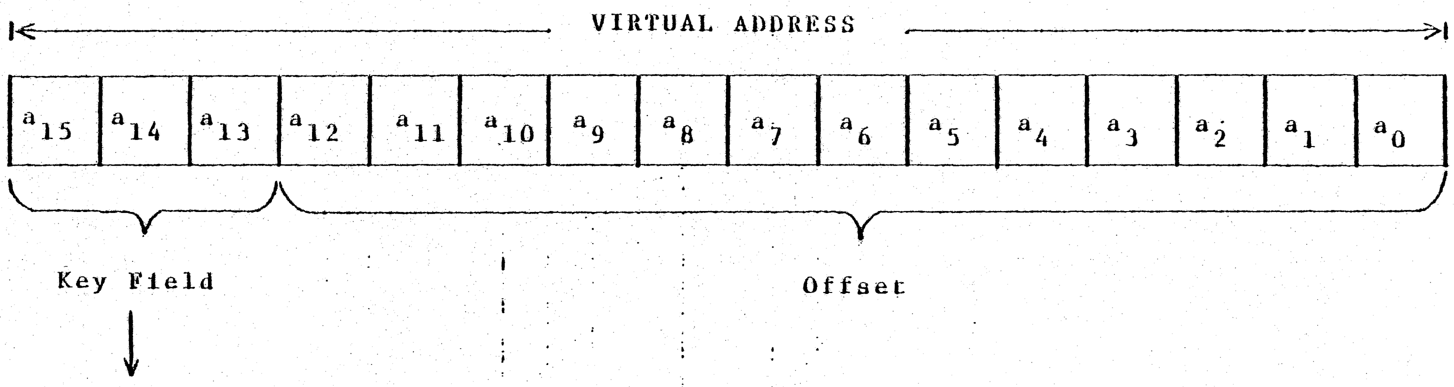
Memory expansion will be accomplished on the Graphic 7 by supplying relocation registers within the memory logic. First, the 128K (word) maximum address space will be supplied as two 64K modules which are identical. Each module will contain three relocation registers which can be addressed as regular device registers. In order to understand the usage of the relocation function, we consider a sample 16 bit address.

Figure A.2-1 depicts the 16 bit address as presented to the memory management logic. The three high order bits (labeled "Key field") will indicate, as shown, whether or not relocation is to be used and which register will be applied. The relocation registers are five bits long, with the final physical address determined as indicated in Figure A.2-2. It is important to note that the offset portion of the address remains the same in the final physical address. The result is that the mapping function can point to any particular 4K (word) block with user access to the entire block.

In summary, the 32K (words), 16-bit address space has been divided into 8, 4K blocks. Three of the blocks can be mapped into any of the 32, 4K blocks in a 128K word, 18 bit memory space. The four blocks of 0-4K, 16-20K, 20-24K and 24-28K will continue to reference respective physical memory areas. Finally, the device register addresses continue to occupy the highest 4K block.

A.3 OPERATING SYSTEM IMPLICATIONS

The performance of the relocation function is usually left to the operating system. This involves inserting the relevant values into the relocation registers prior to giving control to the user task. User tasks generally do not concern themselves


VIRTUAL ADDRESS
a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
Key Field
Offset

0	0	0	0	Direct Addressing
1	0	0	1	Combine Offset with Relocation Register 1
2	0	1	0	Combine Offset with Relocation Register 2
3	0	1	1	Combine Offset with Relocation Register 3
4	1	0	0	Direct Addressing
5	1	0	1	Direct Addressing
6	1	1	0	Direct Addressing
7	1	1	1	Device Registers

FIGURE A.2-1 MEMORY ADDRESSING

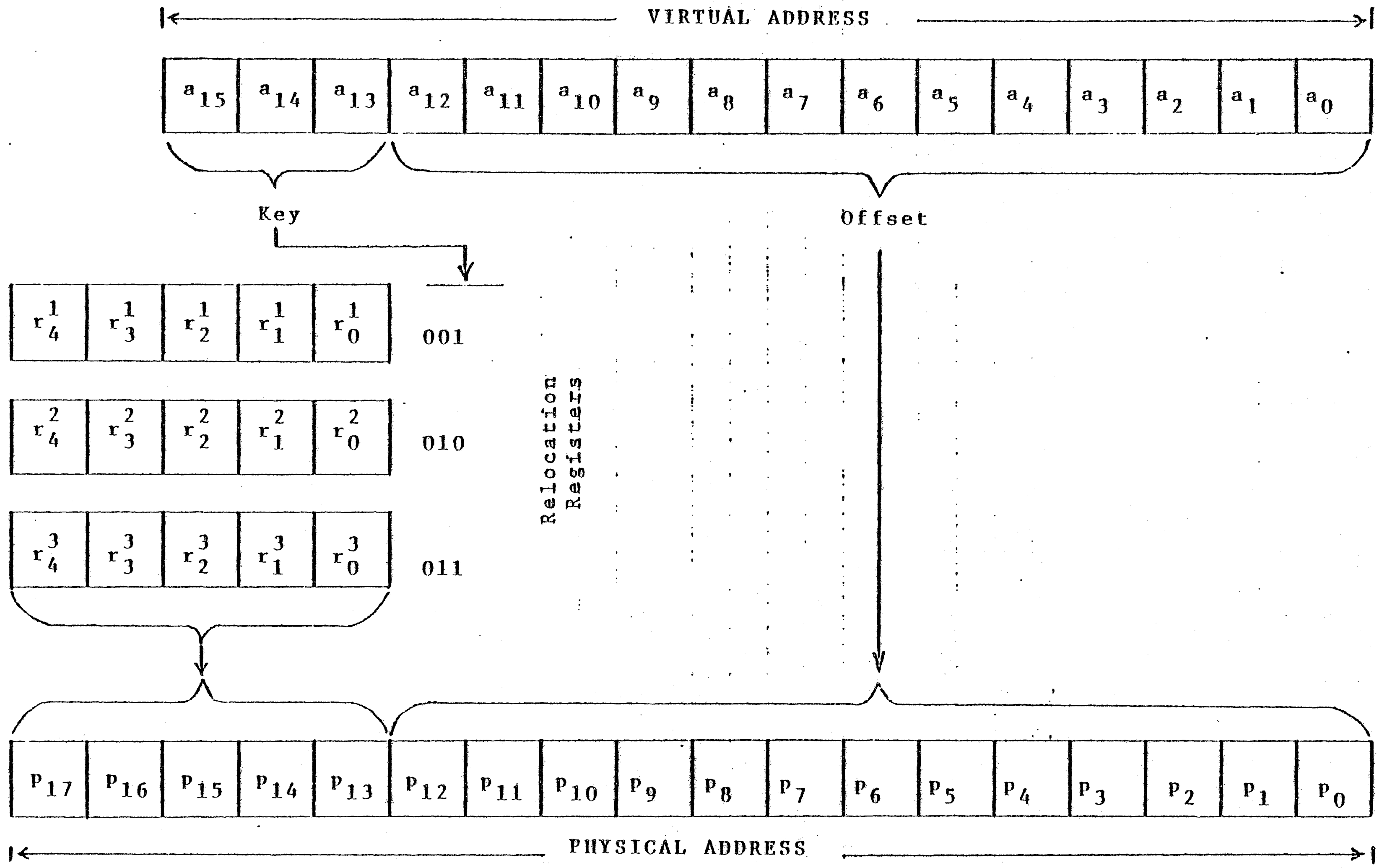

VIRTUAL ADDRESS
a_{15} a_{14} a_{13} a_{12} a_{11} a_{10} a_9 a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0
Key
Offset
r^1_4 r^1_3 r^1_2 r^1_1 r^1_0
001
r^2_4 r^2_3 r^2_2 r^2_1 r^2_0
010
r^3_4 r^3_3 r^3_2 r^3_1 r^3_0
011
Relocation Registers
p_{17} p_{16} p_{15} p_{14} p_{13} p_{12} p_{11} p_{10} p_9 p_8 p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0
PHYSICAL ADDRESS

FIGURE A.2-2 PHYSICAL ADDRESS CONSTRUCTION

with the relocation problem thus allowing independence from actual physical load-point considerations. The proposed Graphic 7 operating system will satisfy this requirement by storing the user relocation values in the task header.

Several aspects of the particular Graphic 7 memory management facility result in operating system constraints. One such restriction is the 12K task size limit, based on the existence of only three relocation registers. Also, due to the rather coarse discrimination of the relocation values, the allocation of core will tend to be somewhat inefficient. These limitations can probably be accepted for most applications.

Once in execution a task will generally be ignorant of its actual physical memory location. However, the monitor must be able to access the task areas as well as itself. For this reason the monitor will need to execute in direct address space, leaving the relocation registers free for references to user program space.* This requirement is reflected in our decision to place the monitor in the direct address area of 16-28K.** Most monitor functional areas may thus assume that the calling task's relocation registers are currently installed.

Recording of task relative addresses either by the monitor or by other user tasks is generally accomplished by saving the 16 bit task-relative address and the task RR1 value. The RRI value provides a pointer to the task header, hence access to the task's relocation registers. Thus, saving the RR1 value yields , in effect, addressing to the entire task image.

* There are implications with respect to interrupt vectors also.
** The problem of collision with dedicated ROM space in this area is yet to be addressed.

As an example of address manipulation within the memory mapping mechanism we present the code necessary to provide access to a display element. The display element address returned by the monitor is a 16 bit physical memory address. For the user task to gain access to the element storage area it is necessary to construct a relocation register value and a modified 16 bit address which will be mapped through the relocation register. The following code assumes the display element address has been loaded into general register one and relocation register three is not needed for task execution.

```
        .
        .      Load Element Addr. Into R1
        .
        CLR    R0
        ASHC   #3,R0       ;Isolate 4K Block Number
        MOV    R0,@#RR3    ;Set Relocation Value
        BEQ    .+6
        MOV    #6,R0       ;Force Mapping Through R3
        ASHC   #-3,R0      ;Create Revised Address in R1.
        .
        .
```

A.4 GRAPHIC CONTROLLER MEMORY ACCESS

The Graphic Controller has an 18 bit address capability, however, only the low order 16 bits are "active." Effectively this amounts to dividing a 128K (word) memory into 4 , 32K blocks which, during a continuous refresh cycle, the Graphic Controller is restriced to one. This mechanism realistically prohibits users from having refresh code within their task areas.

APPENDIX B

SYSTEM GENERATION

B.0 SYSTEM GENERATION - INTRODUCTION

System generation includes those activities necessary to tailor the monitor to a particular user application. This involves the specification of a memory configuration and inclusion of the required monitor functional areas. For the case of the I/O function, the user will also specify the device handlers required by the expected peripheral/interface population.

The following discussion assumes the user has available a cross assembler/taskbuilder to be applied once the necessary source file modifications have been made.

B.1 MEMORY CONFIGURATION ADJUSTMENT

The basic memory allocation is discussed in Section 3.1. The user has some latitude in the actual assignments made to the various basic memory areas. All of the parameters effecting memory allocation are within task MMGMT.MAC. Users should review the normal parameter values and make adjustments as necessary. MMGMT should be re-assembled following adjustment of the allocation parameters. The adjustable points are:

B.1.1 Refresh Space Allocation

The normal allocation for refresh space is 4K blocks 1, 2 and 3. This is indicated by parameter REFBGN and by the pre-setting of appropriate BMASK bits (to indicate these blocks are not available for task space allocation). The user may, depending on the particular application requirements, restrict the refresh space to only block 1 or blocks 1 and 2. (Refresh space must be adjacent to schedule table, block zero.) Any change to the refresh allocation is easily accomplished by adjusting parameter REFBGN and clearing respective BMASK bits.

B.1.2 Monitor Space Allocation

The monitor image is assembled to begin at octal 100000 and occupy 4K blocks 4 and 5. The scaling services, if used, occupy block 6. These blocks are reserved by the setting of the respective BMASK bits. If the scaling services are not required or the monitor image needs only one 4K block, then the appropriate 4K block bits in BMASK should be reset (thus making these blocks available for task space allocation).

A word concerning the monitor stack. The stack area used by the monitor routines is normally located at the top of 4K block 5. The initial stack value is specified by parameter SYSR6 in module MNTRSP.MAC. Obviously this value must be adjusted if you restrict the monitor to block 4. Also, the stack pointer initial value may be adjusted to allow for a (small, typically) global common area within the direct address space of blocks 4 & 5. Such an area, assuming the overall monitor memory requirement permits, can be particularly convenient in light of its direct-address nature and consequent ease of access by all user tasks. (This is independent of the global common discussed below.)

B.1.3 Global Common Allocation

During system generation the user may reserve up to 4 4K blocks as global common space. This is accomplished by placing the block numbers in array GBLCOM and then adjusting the respective BMASK bits (see also section 3.5).

B.1.4 Task Space Allocation

Any 4K blocks not consumed by the above requirements may be employed by user tasks. For this purpose, all 4K blocks are identical and are assigned on an as-needed basis by the appropriate MMGMT entry points.

The maximum memory size is indicated by parameter MAXBLK which is typically set to either 16 (64K memory) or 32 (128K memory). The initial state of all 4K blocks (allocated or unallocated) is indicated by BMASK.

B.2 MONITOR FUNCTION SELECTION

Inclusion of the basic monitor functions is generally dependent on the user configuration and available monitor memory space. Some of the functional areas are rather basic to the overall monitor structure and must be included (memory management, task management, etc.).

Other functions may be included at the option of the user. Those which are easily excluded are photopen services and data transfer services. Display management involves Display Element Management and the Display Scheduler. These two elements generally need to be included or excluded together. In summary, the system generation activity should include a review of each functional area with a decision to either include or exclude each from the final system image.

These decisions result in modifications to the function access table, FUNTBL, in the supervisor module, MNTRSP. Appropriate adjustments to the GLOBAL declarations in MNTRSP will also need to be made so that the selected modules are included at task build time.*

B.2.1 Display Scheduler Selection

Two versions of the Display Scheduler are available; one for color (SCHDLR) and one for monochrome (MSCHDL). Users should include the one appropriate to their situation. This may be done during system task build.

*Note that entries in FUNTBL are position dependent. Unused entries may be employed by individual users for specialized monitor functions.

B.3 DEVICE HANDLER SELECTION

The monitor I/O functional area consists of an I/O supervisor, module IOSPVR, and a variable number of device handlers. Access to a particular device handler, by a user task, is on the basis of a "physical"* device number contained in one of the eight entries of the user task header. This physical device number is simply used to index the table of device driver addresses, DEVADR, contained in the supervisor module, MNTRSP. Table DEVADR should be adjusted by the user, at system generation time, to reflect the actual device drivers required. Each entry will also have a companion GLOBAL declaration. Parameter NUMDEV should reflect the number of device handlers in the final system image. Entries in DEVADR have positional significance in that user tasks expect a particular device to be associated with a unique positive integer which is used to index into the table.

*We use the term physical in order to distinguish this quantity from the "logical" device number contained in the user I/O control blocks.

B.4 MONITOR SERVICES SELECTION

The monitor functional area known as "Monitor Services" contains a variety of sub-modules which are of general interest to many user application tasks. The system generator should review the standard entries in this area and dispose of those which are not required by this application. In addition, users may add specialized services to this section as required by particular situations.* Addition/Deletion of specific services entails the following adjustments:**

a) adjust table SRVTBL in module MTRSRV

b) adjust GLOBAL declarations in MTRSRV

c) adjust the initialization entry (zero) of MTRSRV to reflect any initialization requirements of the added/dropped services.

*Users will frequently face decisions regarding inclusion of certain functions as monitor services or as user-task-resident subroutines. There are no generally applicable rules here, however, inclusion of new monitor services should take into account the overhead inherent in the monitor call and expanded monitor memory requirements.

**In addition to adding sub-functions within the Monitor Services section, users may also add new monitor functional areas. This is easily accomplished by adjustments to table FUNTBL in the supervisor module, MNTRSP. It is advisable to review the EMT servicing section of MNTRSP before proceeding with a functional area addition.

B.5 MISCELLANEOUS CONSIDERATIONS

B.5.1 Initial Task Load

The standard monitor structure is set up to do a task load as the last action of system initialization. The usual task is OPCOM (task 01), from the paper tape reader. The task and/or device may be changed by modifying the LDTSK block in the supervisor module, MNTRSP.

B.5.2 Execution Status Monitor

The standard monitor displays a system identification message at the top of the screen. This line contains an execution indicator (alternating XO pattern) which can be of value during system implementation/test to roughly judge CPU load. Suppression of the monitor title line is accomplished by changing the priority of the schedule entry or nulling-out the schedule request. Reference MNTRSP, initialization section.

B.6 SYSTEM GENERATION SUMMARY

The overall sequence for generating a tailored monitor image is then:

a) Adjust memory layout
b) Select monitor functions
c) Select device drivers
d) Select monitor services
e) Edit necessary source modules
f) Assemble source modules *
g) Task build the system image to start at octal 100000. Module MNTRSP should be first in the load image; ordering otherwise unimportant.

* Monitor source modules require, on assembly, the inclusion of file HDROFF.MAC which contains the task header offset definitions and the G7CALL macro.

APPENDIX C

QUERY/RESPONSE SUBROUTINE

C.0 Query/Response Discussion

The ability to display a query to the operator and accept his response is a common requirement of real time processing systems. To this end, the monitor provides the basic resources necessary to implement the query capability with some of the requirement left to the user application routine. The monitor contribution to the query problem consists of the necessary display functions, keyboard handler and query buffer resources. The user is then responsible for combining these facilities in such fashion as is necessary to satisfy the particular query/response requirement.

As an example of the possible manipulation of these resources, we present a somewhat generalized query/response subroutine which the user may link into his application routine. This routine will take a reasonably simple parameter block and thus relieve the user of many details regarding the various monitor services employed. While the routine discussed below will serve a wide variety of query situations, it may need to be modified or extended to suit particular application requirements.

In addition to its usage here as a query subroutine, users may also reference the included listing for examples of application of the various monitor functions.

C.1. Query Subroutine Usage

The Query/Response subroutine (QRYRSP) may be used to display an alphanumeric query line in the lower left corner of the display and accept a response from the keyboard. Characters entered on the keyboard are reflected in the display at the end of the query line. The user response is always terminated with a carriage return. Logical device zero is assumed to be the relevant keyboard.

The call to this routine will consist of loading the address of a parameter block into register one and then executing:

```
JSR    PC,QRYRSP
```

The parameter block is of the form:

Word	Contents
0	Address of query line
1	Address of response buffer
2	Response length (byte 1)
	Max. Response Length (byte 0)

The query line should be terminated with a null (zero) byte. The response characters are returned with the parity (high order bit) cleared. The response will be terminated with a carriage return (octal 15); however, the count returned in byte 5 of the parameter block does not include the carriage return character.

Users may modify this routine to suit their particular requirements. As an example, the schedule entry control block may be adjusted to change the location, intensity, color, etc. of the displayed query line. Time-out on the response might also be required in some instances.

The subroutine assembly listing is shown in Figure C.1-1. One might also note that QRYRSP may be used to display non-response messages by simply placing a zero in the response buffer address parameter.

```
                                        .TITLE  QRYRSP
                                        .NLIST  BEX,ME
                                ;**************************************************************
                                ;*                                                            *
                                ;*      QUERY/RESPONSE USER TASK SUBROUTINE.                  *
                                ;*                                                            *
                                ;**************************************************************
                                ;
                                ;   PROGRAMMER: M. FRY
                                ;
                                ;   DATE:
                                ;
                                ;   PURPOSE: TO ALLOW A USER ROUTINE TO DISPLAY A QUERY LINE
                                ;       AND ACCEPT A RESPONSE VIA INPUT REFLECTED ON THE DISPLAY.
                                ;
                                ;   ENTRY CONDITIONS:
                                ;       R1 POINTS TO PARAMETER BLOCK OF THE FORM:
                                ;       WORD        CONTENTS
                                ;       ----        --------
                                ;         0         QUERY BUFFER ADDRESS
                                ;         1         RESPONSE BUFFER ADDRESS
                                ;         2         NO. CHARS ENTERED/MAX CHARS TO ACCEPT
                                ;
                                ;   NOTES:
                                ;       - QUERY LINE SHOULD BE TERMINATED BY NULL(ZERO BYTE).
                                ;       - COMBINED LENGTHS OF QUERY AND RESPONSE SHOULD BE NO
                                ;         GREATER THAN 126 BYTES (INCLUDING CARRIAGE RETURN).
                                ;       - SUBROUTINE WILL USE TIMED SUSPENSION IF EITHER OF
                                ;         QUERY BUFFER OR INPUT DEVICE IS UNAVAILABLE.
                                ;
                                        .GLOBL  QRYRSP
        000000                          R0=%0
        000001                          R1=%1
        000002                          R2=%2
        000003                          R3=%3
        000004                          R4=%4
        000005                          R5=%5
        000006                          SP=%6
        000007                          PC=%7
000000                          QRYRSP:
000000  010046                          MOV     R0,-(SP)        ; SAVE REGISTERS AS NEEDED
000002  010146                          MOV     R1,-(SP)
000004  010246                          MOV     R2,-(SP)
000006  010346                          MOV     R3,-(SP)
000010  010102                          MOV     R1,R2           ; SAVE PARM BLK POINTER
                                ;
                                ;   ALLOCATE A QUERY/RESPONSE BUFFER.
                                ;
000012                          TRY2:
000012  012701  000252'                 MOV     #QRYBLK,R1      ; POINT TO PARAMETER BLOCK
000016                                  G7CALL  1,1             ; REFERENCE MONITOR SERVICES
000020  105737  020004                  TSTB    @#T$MRST        ; SUCCESSFUL ALLOCATION ?
000024  001404                          BEQ     HAVEIT
```

FIGURE C.1-1   (Page 1 of 4 pages)

```
                                        ;
                                        ;   ON FAILURE, SUSPEND FOR ONE SECOND.
                                        ;
 000026  012701  000254'                        MOV     #TMOUT,R1
 000032                                         G7CALL  3,5             ; REQUEST TIMED SUSPENSION
 000034  000766                                 BR      TRY2            ; RETRY THE BUFFER ALLOCATION
                                        ;
                                        ;   DEFINE THE DISPLAY ELEMENT FOR QUERY LINE EXPOSURE.
                                        ;
 000036                                 HAVEIT:
 000036  016767  000210  000226                 MOV     QRYBLK,QBUFAD   ; INSERT CALL ADDR. IN ELEMENT
 000044  012701  000256'                        MOV     #DECTLB,R1      ; POINT TO PARAMETER BLOCK
 000050                                         G7CALL  4,2             ; PASS THE DISPLAY ELEMENT
                                                                        ; WE IGNORE THE POSSIBILITY OF
                                                                        ;  DISPLAY ELEMENT ALLOCATION FAILURE.
                                        ;
                                        ;   MOVE QUERY LINE TO BUFFER AREA.
                                        ;
 000052  011200                                 MOV     (R2),R0         ; PICK UP QUERY LINE START ADDRESS
 000054  016701  000172                         MOV     QRYBLK,R1       ; PICK UP BUFFER START ADDR.
 000060                                 GETALL:
 000060  112003                                 MOVB    (R0)+,R3        ; PICK UP A QUERY LINE CHARACTER
 000062  001404                                 BEQ     BUFEND          ; CHECK FOR NULL TERMINATION
 000064  052703  000200                         BIS     #200,R3         ; MAKE INTO TEXT COMPATIBLE FORM
 000070  110321                                 MOVB    R3,(R1)+        ; PLACE CHAR IN QUERY BUFFER
 000072  000772                                 BR      GETALL
 000074                                 BUFEND:
 000074  112721  000240                         MOVB    #240,(R1)+      ; NEED BLANK BEFORE RESPONSE
 000100  010167  000174                         MOV     R1,RESPIO+4     ; SET RESPONSE ADDR IN I/O BLOCK
 000104  116267  000004  000170                 MOVB    4(R2),RESPIO+6  ; SET EXPECTED LENGTH IN I/O BLOCK
                                        ;
                                        ;   ATTACH THE KEYBOARD.
                                        ;
 000112                                 ATHAGN:
 000112  012701  000274'                        MOV     #RESPIO,R1      ; POINT TO THE I/O BLOCK
 000116                                         G7CALL  7,3             ; ISSUE ATTACH REQUEST
 000120  105737  020004                         TSTB    @#T$HRST        ; ATTACH ACCEPTED ?
 000124  001404                                 BEQ     SCHIT           ; CONTINUE IF ATTACH SUCCESS
 000126  012701  000254'                        MOV     #TMOUT,R1       ; TIME OUT FOR ONE SECOND
 000132                                         G7CALL  3,5             ; REQUEST TIMED SUSPENSION
 000134  000766                                 BR      ATHAGN          ; TRY ATTACH AGAIN
                                        ;
                                        ;   SCHEDULE THE QUERY LINE.
                                        ;
 000136                                 SCHIT:
 000136  012701  000314'                        MOV     #SCHBLK,R1      ; POINT TO PARAMETER BLOCK
 000142                                         G7CALL  5,1             ; ALERT THE DISPLAY SCHEDULER
                                        ;
                                        ;   REQUEST INPUT FROM THE KEYBOARD.
                                        ;
 000144  012701  000274'                        MOV     #RESPIO,R1      ; POINT TO I/O CTL BLK
 000150                                         G7CALL  7,5             ; REQUEST KEYBOARD INPUT
                                                                        ; POSSIBLE I/O ERROR IGNORED
```

FIGURE C.1-1 (Page 2 of 4 Pages)

```
ORYRSP                         MACRO M1110  31-JUL-79 06:23  PAGE 5

                                     ;
                                     ;  MOVE RESPONSE TO USER BUFFER.
                                     ;
    000152  016200  000002                   MOV     2(R2),R0          ; PICK UP RESPONSE BUFFER ADDR.
    000156  001414                           BEQ     IGNORE            ; IGNORE IF ZERO
    000160  016701  000126                   MOV     RESPIO+16,R1      ; PICK UP RESPONSE COUNT
    000164  110162  000005                   MOVB    R1,5(R2)          ; INSERT IN CALLER PARAMETER BLOCK
    000170  005201                           INC     R1                ; ACCOUNT FOR CARRIAGE RETURN
    000172  016703  000102                   MOV     RESPIO+4,R3       ; PICK UP START OF ENTERED TEXT
    000176                           MOVIT:
    000176  112302                           MOVB    (R3)+,R2          ; PICK UP RESPONSE CHARACTER
    000200  042702  000200                   BIC     #200,R2           ; ERASE PARITY BIT
    000204  110220                           MOVB    R2,(R0)+          ; PLACE IN USER BUFFER
    000206  077105                           SOB     R1,MOVIT          ; MOVE ALL RESPONSE CHARACTERS
                                     ;
                                     ;  DETACH THE KEYBOARD.
                                     ;
    000210                           IGNORE:
    000210  012701  000274'                  MOV     #RESPIO,R1        ; POINT TO I/O BLOCK
    000214                                   G7CALL  7,4               ; ISSUE DETACH RECUEST
                                     ;
                                     ;  DELETE THE SCHEDULE TABLE ENTRY.
                                     ;
    000216  012701  000314'                  MOV     #SCHBLK,R1        ; POINT TO SCH. CONTROL BLOCK
    000222                                   G7CALL  5,2               ; REFERENCE THE DISPLAY SCHEDULER
                                     ;
                                     ;  DELETE THE DISPLAY ELEMENT.
                                     ;
    000224  012701  000256'                  MOV     #DECTLB,R1        ; POINT TO ELEMENT CONTROL BLK
    000230                                   G7CALL  4,5               ; REFERENCE DISPLAY ELEMENT MGMT
                                     ;
                                     ;  RELEASE THE QUERY BUFFER.
                                     ;
    000232  012701  000252'                  MOV     #QRYBLK,R1        ; POINT TO PARAMETER BLOCK
    000236                                   G7CALL  1,1               ; REFERENCED MONITOR SERVICE
                                                                       ;  WILL ZERO QRYBLK SO AS TO BE
                                                                       ;    READY FOR THE NEXT ALLOCATION.
                                     ;
                                     ;  RESTORE REGISTERS AND EXIT.
                                     ;
    000240  012603                           MOV     (SP)+,R3
    000242  012602                           MOV     (SP)+,R2
    000244  012601                           MOV     (SP)+,R1
    000246  012600                           MOV     (SP)+,R0
    000250  000207                           RTS     PC                ; RETURN TO CALLER
```

FIGURE C.1-1   (Page 3 of 4 pages)

```
                                        ;
                                        ;   STORAGE AND CONSTANTS FOR QUERY/RESPONSE SUBROUTINE.
                                        ;
                                        ;   PARAMETER BLOCK FOR QUERY BUFFER ALLOCATION.
                                        ;
    000252  000000                      QRYBLK: .WORD   0
                                        ;
                                        ;   PARAMETER BLOCK FOR 1 SECOND TASK SUSPENSION.
                                        ;
    000254  000012                      TMOUT:  .WORD   10.                  ; 1 SECOND IN TENTH SECONDS
                                        ;
                                        ;   PARAMETER BLOCK FOR DISPLAY ELEMENT DEFINITION.
                                        ;
    000256  000000                      DECTLB: .WORD   0                    ; RESERVED WORD
    000260  000000                              .WORD   0                    ; ELEMENT START ADDRESS(RETURNED)
    000262  000000                              .WORD   0                    ; ELEMENT FLAGS/STATUS
    000264  000004                              .WORD   4                    ; ELEMENT LENGTH
    000266  000270'                             .WORD   .+2                  ; POINT TO ELEMENT
    000270  002100                              .WORD   2100                 ; CALL REFRESH SUBROUTINE
    000272  000000                      QBUFAD: .WORD   0                    ; QUERY BUFFER ADDR. TO BE INSERTED
                                        ;
                                        ;   I/O BLOCK FOR KEYBOARD INPUT.
                                        ;
    000274  000000                      RESPIO: .WORD   0                    ; RESERVED
    000276  000001                              .WORD   1                    ; LOGICAL DEVICE ONE ASSUMED
    000300  000000                              .WORD   0                    ; RESPONSE BUFFER ADDR.
    000302  000000                              .WORD   0                    ; RESPONSE EXPECTED LENGTH
    000304  020000                              .WORD   20000                ; IGNORE FUNCTION KEYS
    000306  000000                              .WORD   0                    ; NO ERROR RETURN
    000310  000000                              .WORD   0                    ; NO TIME-OUT
    000312  000000                              .WORD   0                    ; TRANSFER COUNT (RETURNED)
                                        ;
                                        ;   SCHEDULE CONTROL BLOCK FOR QUERY LINE.
                                        ;
    000314  000000                      SCHBLK: .WORD   0                    ; RESERVED
    000316  000000                              .WORD   0                    ; SCHEDULE ENTRY ADDRESS (RETURNED)
    000320  000000                              .WORD   0                    ; STATUS AND FLAGS
    000322  000256'                             .WORD   DECTLB               ; ELEMENT CTL BLK POINTER
    000324     177     003                      .BYTE   127.,3               ; PRIORITY,CRTS
    000326     007     003                      .BYTE   7,3                  ; INTENSITY,COLOR
    000330     000     000                      .BYTE   0,0                  ; LINE STRUCTURE,CHAR SIZE
    000332  177076                              .WORD   -450.                ; X START
    000334  177045                              .WORD   -475.                ; Y START
                                                                             ; MODIFY NOT NEEDED
            000001                              .END
```

FIGURE C.1-1   (Page 4 of 4 pages)

APPENDIX D

GRAPHIC 7 MONITOR

OPERATOR COMMUNICATIONS

D.0 OPERATOR COMMUNICATIONS

Operating system interaction with the operator or application task programmer takes place through the Operator Communications module. Typically, the system provides for monitoring task execution, displaying task status and dumping/modifying memory. This facility will normally be heavily used during application system development and may or may not be actually present during final system operation. The following section describes the implementation of the Operator Communications module in the Graphic 7 monitor.

D.1 Operator Communications Implementation

In the Graphic 7 monitor, the Operator Communications module (herein after referred to as ØPCØM) executes as an independent task, the same as any application task. This configuration allows for deleting the ØPCØM module in the final system implementation (and thus gaining an extra 4K block of memory space) while also reducing requirements on resident monitor direct address space. During system development the ØPCØM task will be automatically loaded during system initialization. ØPCØM is then activated by simply depressing Control/C on the system keyboard. Being at the highest priority, the request prompt is immediately displayed. ØPCØM then utilizes the various standard monitor I/O services to satisfy the request.

Commands to OPCOM take the following standard format:

```
verb   arg1, arg2,...., argn
```

Argument formats are dependent on the particular verb and are discussed in Section D.2. The verb may be spelled out completely or only to the extent necessary to distinguish between the various commands. The following conventions should be observed:

a. Task ID's are always specified as two hexadecimal digits, (must be upper case).

b. Addresses are specified in octal; up to six digits. Task offsets are in the range 0-57776.

c. All inputs are terminated by a carriage return.

d. A carriage return must be entered to release OPCOM in the event of a non-query message (i.e., an error message, memory dump, etc.).

Error messages from OPCOM take the following general form:

```
ERRØR:  nn message
```

"nn" is the status code returned by the particular monitor function which was exercised. For example, the "Disable Task" command will use the respective Task Management entry point. Any error returned from such would be reflected in the "nn" field. This is a decimal number. The "message" field will serve to indicate the command type which failed or the failure reason.

D.2 Operator Commands

The various commands are outlined in the following sections. Users are cautioned to pay particular attention to the argument list and argument types which are expected to be specified with each verb. If there is some doubt about the required argument list, then just specify the verb and a separate query will be issued for the arguments.

D.2.1 Abort Task

Abort may be used to forcefully terminate task execution.

```
Syntax:  ABØRT taskid
```

In order to be run again, the task must be reloaded.

D.2.4.1 Task Memory Dump

It is often convenient to display memory relative to a particular task area. This may be accomplished via the subject task ID and the "TDUMP" command:

Syntax: TDUMP taskid, offset

Following display of the initial 128 word block, the following three single character options are available:

a. Slash - step forward 64 words

b. Up arrow - step back 64 words

c. Carriage return - exit

D.2.2 Continue Task

A task which is currently suspended (indefinite or timed) may be activated by the continue command. An optional task offset may be specified if the resumption point is not to be the current task execution location.* The offset is a "zero-relative" value which OPCOM will convert to a task address by adding octal 20000. After possibly adjusting the saved PC value, OPCOM utilizes the Task Management continue service to awaken the task.

Syntax: CONTINUE taskid [,offset]

D.2.3 Disable Task

Disable may be used to cause a task to be ineligible for execution. Activation of the task would not be possible until an Enable was issued.

Syntax: DISABLE taskid

D.2.4 Dump Memory

The Memory Dump facility will cause a 128 word block of memory to be displayed. The start address, to the nearest even eight word boundary, is the only argument. Following display of the initial 128 word block, the following three single character options are available:

a. Slash - step forward 64 words
b. Up arrow - step back 64 words
c. Carriage return - exit

Syntax: DUMP address

* One should exercise care in modifying the execution sequence since, for obvious reasons, some task code is not re-executable. For example, re-execution of display schedule requests or display element requests would typically be unacceptable and yield mystifying system behavior.

D.2.5 Enable Task

Enable will erase the relevant task status bit, via the associated Task Management entry. Depending on relative priorities and other status conditions, the task will now be eligible for CPU time.

Syntax: ENABLE taskid

D.2.6 Load Task

Load will be used to introduce new tasks to the system.

Syntax: LØAD taskid, device (,filename)

Notes:

a) DEVICE should be a physical device number. (Physical device numbers are determined during system generation.)

b) FILENAME is optional, depending on the medium. Maximum of nine characters.

D.2.7 Patch Memory

The Patch service allows for modifying a single memory word or a block of words. This command has two forms depending on whether two or three arguments are specified. The two argument form is:

Syntax: PATCH address, value
(VALUE is placed at ADDRESS)

The three argument form is:

Syntax: PATCH addr1, addr2, value
(VALUE is placed in locations ADDR1 thru ADDR2.)

D.2.7.1 Patch Task Memory

Memory can be modified on a task relative basis by using the TPATCH command. This command, as in the normal PATCH, has two forms. For a single word patch:

```
Syntax:  TPATCH taskid, offset, value
         ("value" is placed at "offset")
```

For a block of words:

```
Syntax:  TPATCH taskid, offset 1, offset 2, value
         ("value" is placed in locations "offset 1"
          thru "offset 2")
```

D.2.8 Display Task Status

Task status includes the various parameters in the task header and the general registers. The header parameters are identified by their standard offset notations. For a description of these entities, see the Task Management section.

```
Syntax:  STATUS taskid
```

D.2.9 Suspend Task

Suspend may be used to inhibit task execution for a timed or indefinite interval. The time argument, if supplied, should be an integral value in 1/10 second units.

```
Syntax:  SUSPEND taskid (, time)
```

D.2.10 Update Task Status

The update facility may be used to modify the low order eight bits in the subject task's status word. The associated Task Management entry is exercised.

```
Syntax:  UPDATE taskid, status
```

D.2.11 Display/Set Date/Time

The TIME command may be used to either enter the current time or display the current time. The syntax for entering the current time is:

```
TIME  HH:MM:SS {& MM/DD/YY}
```

To display the current date/time simply issue the TIME command with no arguments.

D.2.12 Step-Thru-Memory

TSTEPM and STEPM allow a user to address and display a memory word, step to and display the preceding or succeeding word location, and optionally replace the value of the displayed memory word. TSTEPM addresses the word relative to its location within a task memory space, while STEPM addresses the word directly or by its absolute location in memory. Command syntax is:

```
TSTEPM taskid, offset  (within task)  (task relative)
STEPM  address                        (direct)
```

```
where taskid is hex; address is:  16 bit octal for task relative
                                  18 bit octal for direct.
```

Once the memory word is displayed the user may step to a preceding or succeeding memory word by depressing the uparrow or slash key, respectively.

The displayed memory word may be replaced by entering a 6-digit octal number, and depressing the slash, uparrow, or carriage return key. Carriage return will display the new value at the presently displayed location, while slash and uparrow conceal the change from the user because a different location is next displayed.

Exit from the service is achieved by depressing the carriage return key when no new value is entered.

The TSTEPM/STEPM service is useful when several closely located changes are to be made and the TPATCH and PATCH service may be awkward.

D.2.13 Device Register Display

The memory dump commands naturally reference the memory block for addresses which cover the device register page. The Register command may be used to examine any direct address in the range 160000 to 177776. An error message is displayed if the address is not within this range or the device is non-responsive.

```
Syntax:  REGISTER address
```

D.3 BREAKPOINT SERVICES

The OPCOM breakpoint services allow a user to interrupt a task and examine task status or memory contents. This facility will be valuable for determining the execution sequence or intermediate results of user tasks during initial debug.

D.3.1 BREAKPOINT OPERATION

Task breakpoints are inserted by specifying the task ID and an octal task offset (0-...). When the breakpoint is encountered during task execution, the task is suspended and OPCOM is notified. OPCOM displays a message indicating:

a. task ID (2 hexadecimal digits)
b. task offset (5 octal digits)
c. breakpoint number (decimal, 1-n)

The breakpoint will need to be deleted if the task is to be continued at the given offset. The optional offset parameter of the task continue directive may be used to cause execution to resume at a different point.

D.3.2 BREAKPOINT SIDE EFFECTS

The breakpoint capability is implemented using function code zero of the standard EMT monitor call structure. The sub-function bits are used to store the breakpoint number. (Thus yielding a maximum of 16 breakpoints with OPCOM possibly restricted to less.) As a result of this implementation scheme the following side effect should be noted: the T$MRST status byte in the user's task header is cleared when the breakpoint occurs.

D.3.3 BREAKPOINT RESTRICTIONS

The breakpoint service cannot be used within the monitor or any portion of a user task called as an appendage to the monitor.

D.3.4 OPCOM BREAKPOINT COMMANDS

The following command verbs may be used to manipulate breakpoints.

D.3.4.1 Insert Breakpoint

The insert directive will place a breakpoint in the indicated task, at the specified offset. Remember that task offset values are always relative to zero whereas task addresses will typically be biased by octal 20000. The integral breakpoint number is displayed following saving of the current offset contents and insertion of the breakpoint instruction.

Syntax: BINSERT taskid, offset

D.3.4.2 Delete Breakpoint

Breakpoints are never automatically deleted by OPCOM. The delete command, specifying the breakpoint number, will be used to eliminate previously defined break locations. The action of the delete is to replace the break instruction with the previous offset contents. If this is the current task execution point then the task may now be continued in the normal manner. (When the break occurs the task PC is adjusted to force re-execution of the break offset location.)

Syntax: BDELETE breakpoint-number

D.3.4.3 Purge Breakpoints

The purge command may be used to eliminate all currently defined breakpoints. The purge request is advisable whenever any task exists or aborts since the breakpoints are not automatically eliminated. Purge takes no arguments.

Syntax: BPURGE

APPENDIX E

DISPLAY DATA SUPPORT

E.0 DISPLAY DATA SUPPORT - INTRODUCTION

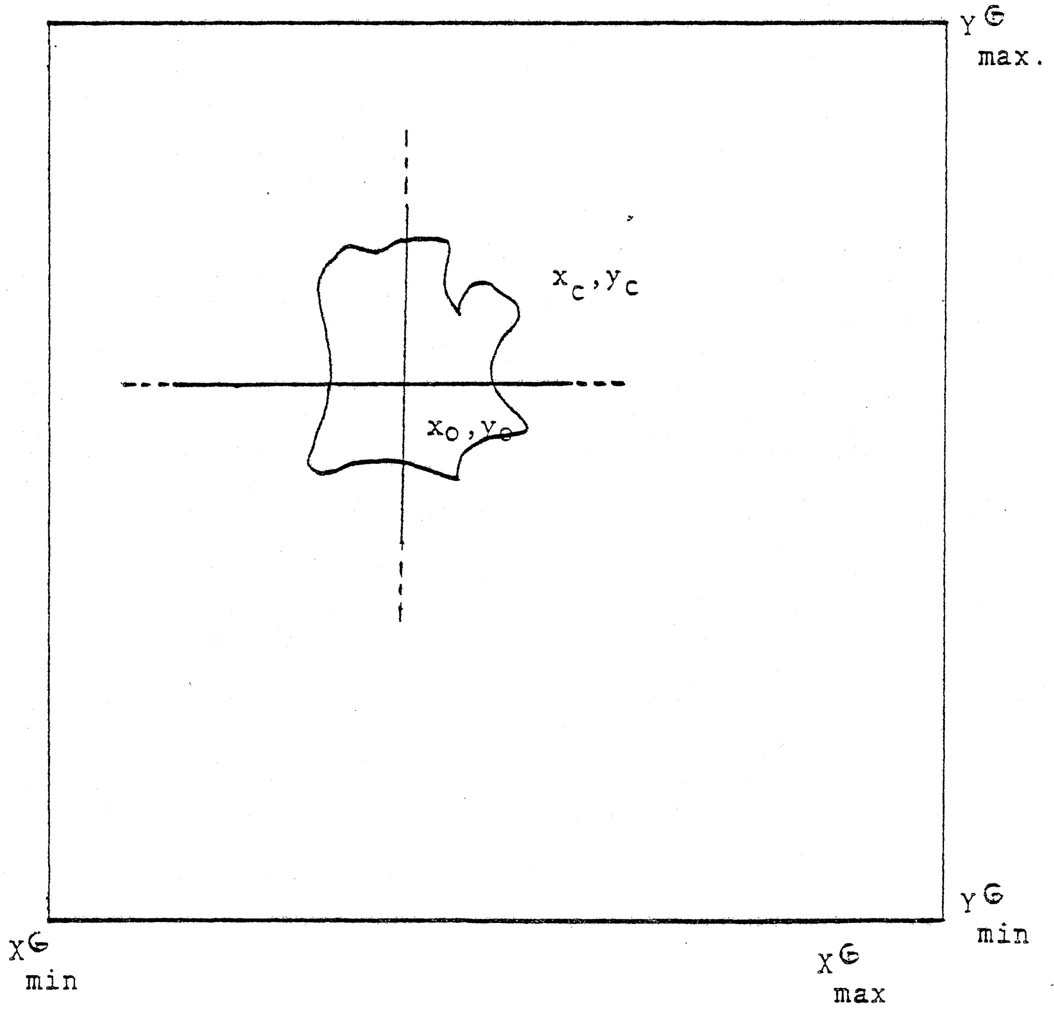
The objective of the Display Data Support facility is to provide monitor based services relevant to general display data manipulation. The following sections discuss a standard display data format and associated support subroutines. The standard display data format is intended to allow greater precision than a typical refresh instruction format as well as to allow the user to supply graphic data independent of any particular machine code. The data format discussed herein will provide all of the flexibility of a refresh representation while being purposely tailored to geographic display elements. The subroutines associated with supporting the standard data format will allow for converting to refresh code while applying coordinate transformations and scaling. These routines will be callable in the standard Graphic 7 monitor context although not considered part of the resident monitor. Details regarding the subroutines are covered in Section E.3. Before proceeding to the standard display data format, we review the assumptions behind the coordinate system used for specification of the input display data.

E.1 COORDINATE SYSTEMS/COORDINATE CONVERSIONS

Inherent in the specification of a common display data format is the definition of a global coordinate system. This global coordinate system should allow for data description at precision levels consistent with the 16 bit word size and system design objectives. Also, the global coordinate system will allow for relating display entities from various data sources. Given the global coordinate system, a function must then be provided for converting to display coordinates. This conversion involves the specification of the display origin and a scale factor. In this area, we feel a careful definition of terms is necessary to ensure consistency and thus allow for the application of generalized, monitor-based services.

Our view of the global coordinate system with respect to a sample display element is shown in Figure E.1-1. The overall scheme is one of cartesian coordinates.* A local origin (X_o, Y_o) is defined for each display element in terms of the global coordinates. This element origin is stored in the display element header. Specifications relevant to a particular display element (move to point X,Y; etc.) will then be relative to the local element origin. A scale adjustment may be made to the local coordinates via factors contained in the element header. This adjustment may be necessary, for example, if data are obtained from different library or real

* Other coordinate systems (polar, etc.) could be supported on a special case basis. The important point is that we do not expect to support adjustments for earth curvature or other map related transformations.



X^G_{min}, X^G_{max}, Y^G_{min}, Y^G_{max}, - Global coordinate system boundaries

FIGURE E.1-1. Global Coordinate System

time sources. At this point, the relationship between some particular map point (for example, Xc, Yc in Figure E.1-1) and the global coordinate system should be clear. How then do we establish this element/point in a screen image?

Several mechanisms exist for defining the relationship between the global coordinates and a particular screen area. For our expected operating situation, it appears convenient to specify the mapping function in terms of a display origin, display boundary and a scale factor. Thus, in transforming a display element from the standard format, the user task will need to indicate what point on the display corresponds to the element origin (Xo, Yo in Figure E.1-1). A user supplied scale and clipping boundary will also be applied during the element transformation. The overall transformation is depicted in Figure E.1-2.

The steps involved in converting a display element specification in standard format to an image component are then:

(a) Apply the element origin and optional scale factors to convert each element point to the global coordinate system.

(b) Apply the requested display origin and display scale factor to arrive at screen-relative coordinates.

(c) Account for the specified clipping boundary.

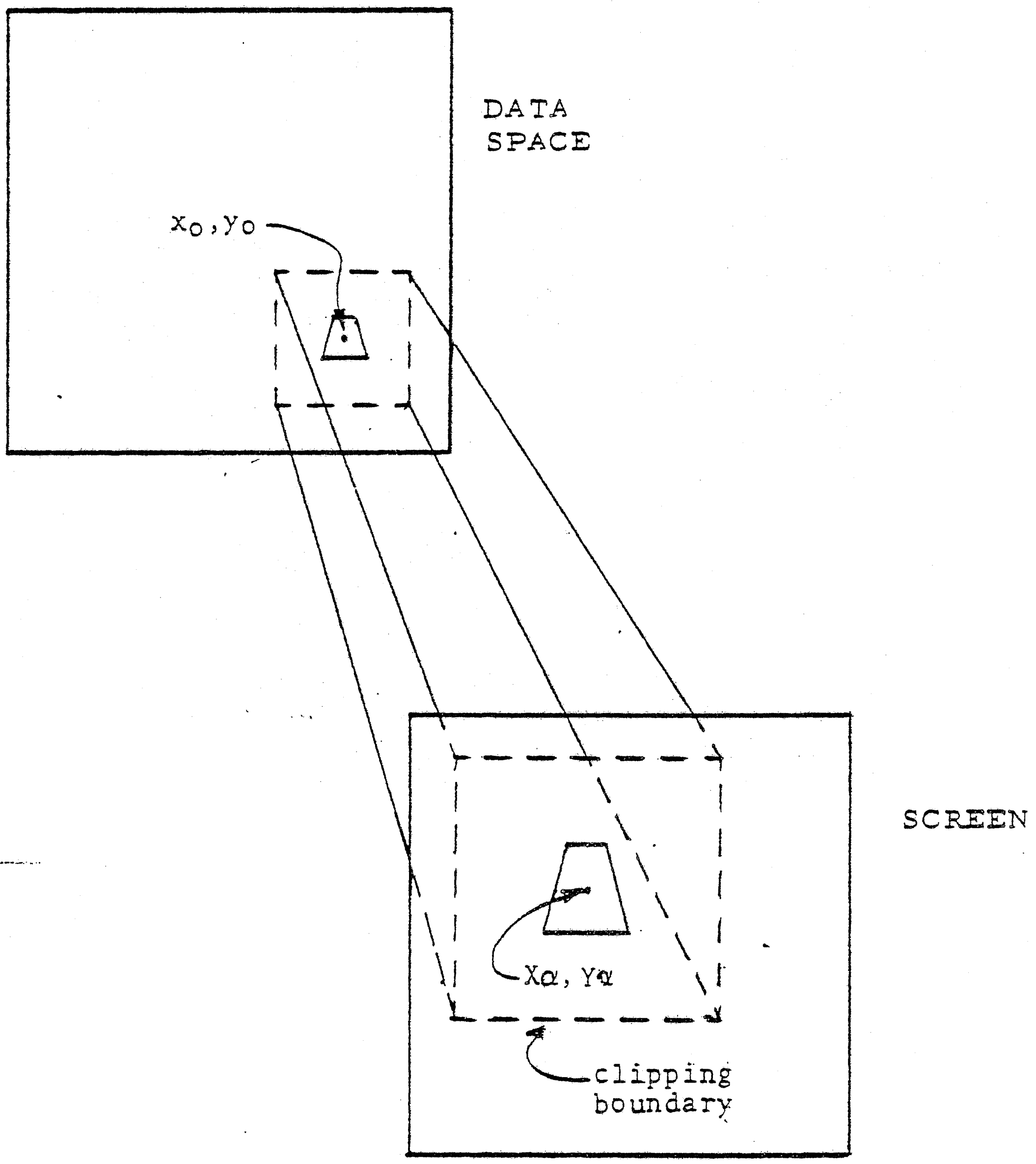


FIGURE E.1-2 Display Element Transformation

E.2 STANDARD DISPLAY ELEMENT FORMAT

The Standard Display Element Format is depicted in Figure E.2-1. The Element header will contain parameters relevant to element identification, conversion and display. These attributes are outlined in Figure E.2.1-1.

The data portion of the display element is composed of a series of subsections which specify the various move, draw, text, etc. image components. Each subsection is composed of a mode (command) control word followed by a group of parameters (typically x,y coordinates). The number of such parameters may be, depending on the mode type, fixed, adjustable, or variable. For variable modes, a terminator must be specified prior to the next mode control.

Users will want to note the definition of each mode very carefully before proceeding to generate a standard data file.

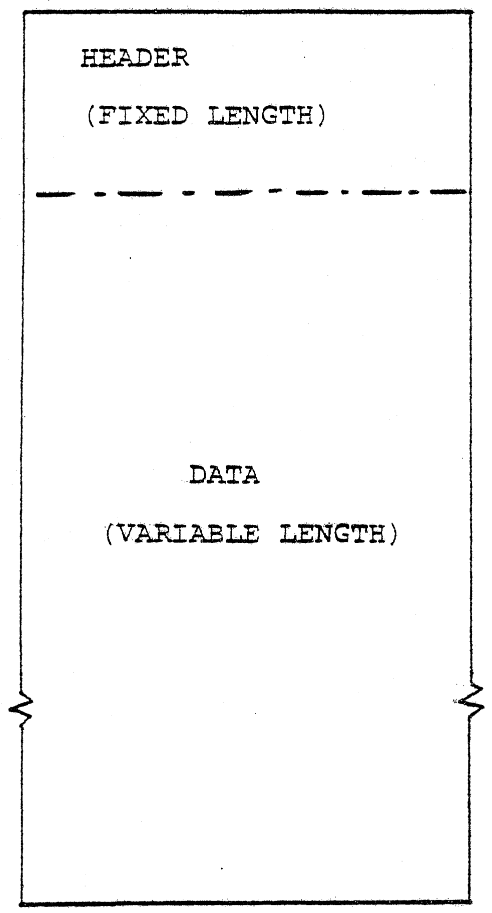


FIGURE E.2-1 Standard Display Element Format

E.2.1 Display Element Header

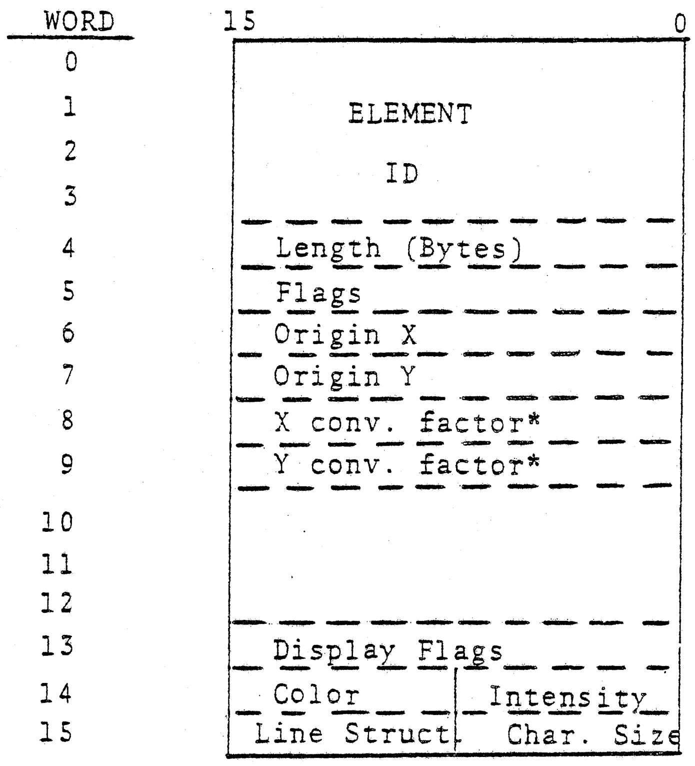
The Element Header contains parameters relevant to the identification and orientation of the image component. The various parameters are outlined in Figure E.2.1-1. Reviewing the various fields:

Element ID:	8 ASCII characters
Length:	total bytes including the header
Flags:	TBD
Origin X,Y:	element origin in global coordinate system units
X,Y Conversion Factors:	optional factors for converting from local units to global units.
Color:	0, 1, 2 or 3*
Intensity:	0-7*
Line Structure:	0, 1, 2 or 3*
Character Size:	0, 1, 2 or 3*

Display Flags:

Bit	Usage
0	
1	
2	Character rotate*
3	
4	
5	Blink*
6	
7	

* These factors would, of course, only be recommendations, i.e., not prohibiting the user routine from supplying its own attributes.



* The conversion factors have an assumed binary point between bits 7 and 8.

Figure E.2.1-1 Display Element Header

E.2.2 Mode Definitions

The mode definitions consist of a mode identifier (ID) and a list of applicable arguments. A new mode identifier is assumed to follow the end of each subsection. The various definitions are discussed on the following pages. Users will want to study the available options so as to allow the most efficient data representation possible.

The mode numbers, placed in subsection byte 0 in each case, are given in decimal. All x,y coordinates/deltas are assumed to be signed binary integers.

The COUNT Byte, where applicable, is taken to be an unsigned eight bit integer.

MODE = 1, MOVE ABSOLUTE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0*	1
1	X COORDINATE
2	Y COORDINATE

(FIXED LENGTH SUBSECTION)

* High order byte ignored.

MODE = 2, MOVE RELATIVE (SHORT)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0*	2
1, byte 0	X delta
1, byte 1	Y delta

(FIXED LENGTH SUBSECTION)

* High order byte ignored.

MODE = 3, MOVE RELATIVE (LONG)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0*	3
1	X delta
2	Y delta

(FIXED LENGTH SUBSECTION)

* High order byte ignored.

MODE = 4, DRAW ABSOLUTE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	4
0, byte 1	Count of coordinate pairs
1	X coordinate
2	Y coordinate
.	
.	
.	

The subsection length is 2 + 4* COUNT bytes.

Count = 0 is an error.

This mode yields a vector drawn between the current X,Y position and the first X,Y coordinate pair. If more than one coordinate pair is specified, then draws are defined to each successive point.

MODE =5, DRAW RELATIVE (SHORT)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	5
0, byte 1	COUNT of delta pairs
1, byte 0	X delta
1, byte 1	Y delta
.	
.	
.	

The subsection length 2 + 2* COUNT bytes.

If COUNT is zero, then the subsection length is variable and the subsection <u>must</u> be terminated by a zero word. Thus a draw relative with zero x,y delta is not allowed.

MODE = 6, DRAW RELATIVE (LONG)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	6
0, byte 1	COUNT of coordinate pairs
1	X Coordinate
2	Y Coordinate
.	
.	
.	

The subsection length is 2 + 4* COUNT bytes.

Variable length subsections of this mode (i.e., COUNT=0) must be terminated with X,Y = 0.

MODE = 10, POINT PLOT ABSOLUTE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	10
0, byte 1	COUNT of coordinate pairs
1	X Coordinate
2	Y Coordinate
.	
.	
.	

The subsection length is 2 + 4* COUNT bytes.

COUNT = 0 is an error.

MODE = 11, POINT PLOT RELATIVE (SHORT)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	11
0, byte 1	COUNT of x,y delta pairs
1, byte 0	X delta
1, byte 1	Y delta
.	
.	
.	

Subsection length is 2 + 2* COUNT bytes

If COUNT is zero, then the subsection length is variable and the subsection <u>must</u> be terminated by a zero word. Thus, a point plot relative with zero x,y delta is not allowed.

MODE = 12, POINT PLOT RELATIVE (LONG)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	12
0, byte 1	COUNT of x,y coordinate pairs
1	X coordinate
2	Y coordinate
.	
.	
.	

Subsection length is 2 + 4* COUNT bytes.

If COUNT is zero, then the subsection length is variable and the subsection <u>must</u> be terminated by a zero x,y coordinate pair. Thus, a plot point relative at the current beam position is not supported in a variable length subsection.

MODE = 20, SET TEXT INCREMENT

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	20
0, byte 1	Text Increment

(FIXED LENGTH)

The usage of this mode if discouraged since the system will supply a default text increment appropriate to the particular screen/character size being used.

The text increment is given in screen units.

MODE = 21, SET LINE INCREMENT

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	21
0, byte 1	Line Increment

(FIXED LENGTH SUBSECTION)

The usage of this mode is discouraged since the system will supply a default line increment appropriate to the particular screen/character size being used.

The line increment is given in screen units.

MODE = 22, SET CHARACTER SIZE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	22
0, byte 1	Character Size

(FIXED LENGTH SUBSECTION)

Usage of this mode is not recommended since it implies the insertion of LDDP instructions within the generated refresh element. Specification of the character size may be done in the element header.

MODE = 23, SET TEXT ORIENTATION

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	23
0, byte 1	Character Orientation 0 = horizontal 1 = vertical

(FIXED LENGTH SUBSECTION)

Usage of this mode is not recommended since it implies the insertion of LDDP instructions within the generated refresh element. Specification of the character orientation may be done in the element header.

MODE = 24, TEXT

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	24
0, byte 1	COUNT of TEXT bytes
1, byte 0	Text (ASCII) character
.	
.	
.	

Subsection length is 2 + COUNT* bytes

If COUNT = 0, then the subsection must be terminated by a zero byte. Carriage returns result in insertion of Load X/Move Y instructions to effect normal CR/LF action.

* Rounded up to include whole words.

MODE = 25, TEXT (Alternate Char. Set)

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	25
0, byte 1	COUNT of TEXT bytes
1, byte 0	Text (ASCII) character
.	
.	
.	

Subsection length is 2 + COUNT* bytes.

If COUNT = 0, then the subsection must be terminated by a zero byte. Carriage returns result in insertion of LOAD-X/MOVE-Y instructions to effect normal CR/LF action. This mode is similar to normal TEXT but a shift-in/shift-out sequence is implied.

---

* Rounded up to include whole words.

MODE = 26, SINGLE CHARACTER

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	26
0, byte 1	ASCII Character

(FIXED LENGTH SUBSECTION)

MODE = 27, SINGLE CHARACTER WITH BLINK

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	27
0, byte 1	ASCII Character

(FIXED LENGTH SUBSECTION)

MODE = 28, SYMBOL

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	28
0, byte 1	SYMBOL Code

(FIXED LENGTH SUBSECTION)

SYMBOL is the same as "SINGLE CHARACTER" except that a shift-in/shift-out sequence is implied.

MODE = 29, SYMBOL WITH BLINK

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	29
0, byte 1	SYMBOL CODE

(FIXED LENGTH SUBSECTION)

SYMBOL is the same as "SINGLE CHARACTER" except that a shift-in/shift-out sequence is implied.

MODE = 30, SET INTENSITY

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	30
0, byte 1	INTENSITY

(FIXED LENGTH SUBSECTION)

Usage of this mode is not recommended since it results in insertion of LDDZ instructions within the refresh. Intensity may be specified in the element header.

MODE = 31, SET BLINK

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	31
0, byte 1	BLINK SPEC. 0 = no blink 1 = blink

(FIXED LENGTH SUBSECTION)

Usage of this mode is not recommended since it results in insertion of LDDZ instructions within the refresh file. In addition, the application task will typically want to have global control of this parameter.

MODE = 32, SET LINE STRUCTURE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	32
0, byte 1	LINE STRUCTURE

(FIXED LENGTH SUBSECTION)

Usage of this mode is not recommended since it results in insertion of LDDZ instructions within the refresh. Line structure may be specified in the element header.

MODE = 40, CONICS

SUBSECTION STRUCTURE:

WORD	CONTENTS
0, byte 0	40
0, byte 1	Quadrant inhibit bits 0 = inhibit quad. 1 1 = inhibit quad. 2 2 = inhibit quad. 3 3 = inhibit quad. 4
1	X radius
2	Y radius

(FIXED LENGTH SUBSECTION)

MODE = 50, END-OF-FILE

SUBSECTION DEFINITION:

WORD	CONTENTS
0	50

This mode may be used to indicate end of the display element data prior to the end of the data block (as indicated by the byte count in the header).

MODE = 60, SWITCH TO SIZE ABSOLUTE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0	60

(FIXED LENGTH SUBSECTION)

This command will allow for inserting a fixed-size object within an otherwise scalable display element. This command will typically be followed, eventually, by a mode 61. Code within a size-absolute section will not be effected when the operator requests a zoom operation.

MODE = 61, SWITCH TO SIZE RELATIVE

SUBSECTION STRUCTURE:

WORD	CONTENTS
0	61

(FIXED LENGTH SUBSECTION)

This command will allow for returning to size relative mode after a previously issued mode 60. Size relative is the default element type and need not be explicitly specified unless a switch from absolute mode is necessary.

E.2.3 A Note on Text Usage

A caution with respect to text usage within a display element must be noted. Data within a standard format element is generally given in local coordinate system units which are translatable to global system units via the X, Y conversion factors given in the header. Text however represents, effectively, data in display units which may not be exactly represented in local system units. Thus, following a string of text characters, the local (virtual) beam position will not normally be equivalent to the display beam position. For this reason, we strongly encourage users to follow a text block with an absolute move. This will ensure that subsequent relative vectors are properly positioned.

E.3 Display Data System Support Subroutines

From the Monitor viewpoint, Display Data Support consists of providing the necessary facilities for transforming a standard format display element into refresh code. Associated with this transformation is the application of the current display origin and scale factor. The description of these services which follows will need to be carefully considered by prospective users as the control structure will seem somewhat complex at first glance.

As previously discussed, the display element support routine will reside in a separate 4K memory block. Communication with this facility will be via a standard Monitor Service call with the usual parameter block pointer in Register 1. Execution within the support subroutine will be indirect via Relocation Register 2. The overall task-monitor relationship is depicted in Figure E.3-1.* We proceed now to a discussion of the control blocks necessary to utilize the support routine.

* Loading mechanism for the Display Element support routine is TBD.

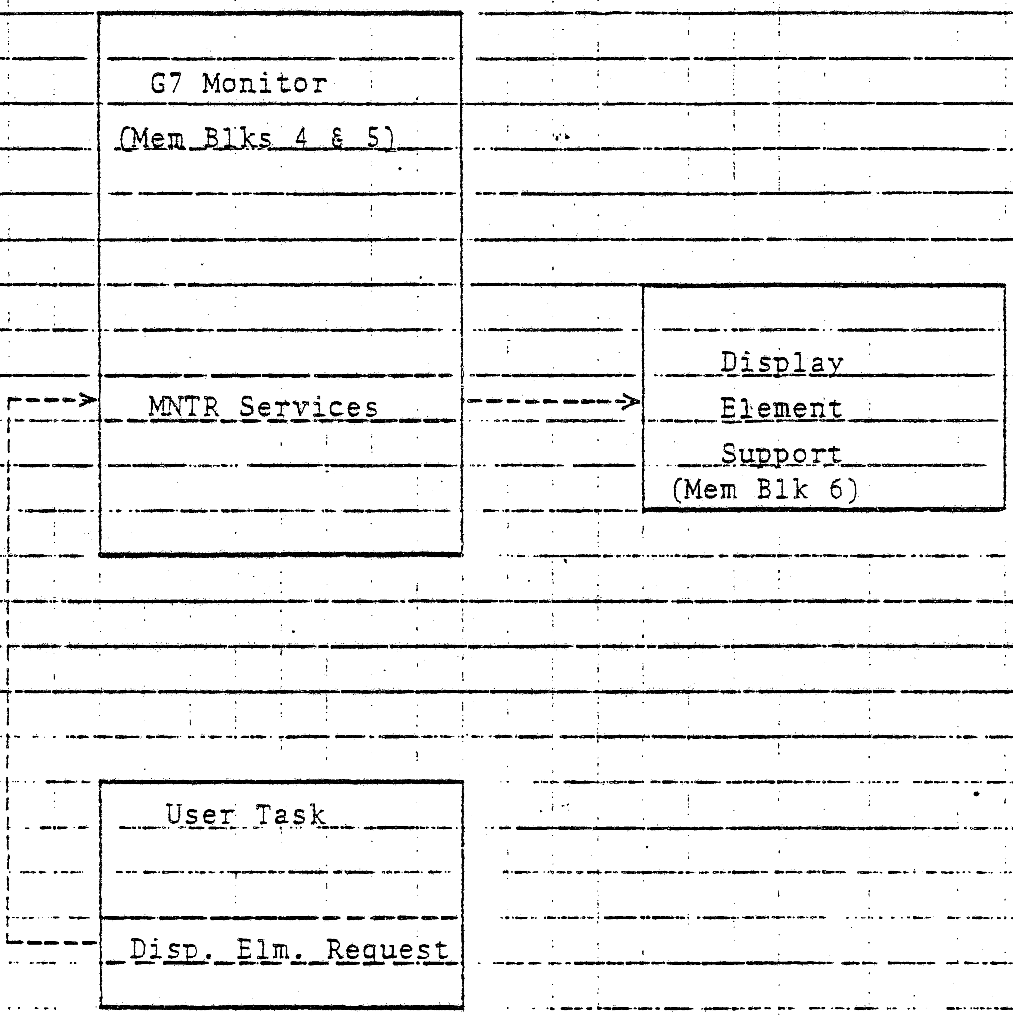


Figure E.3-1 System Configuration

E.3.1 Control Block Structures

Associated with a Display Element Support request is a parameter block which specifies the data to be acted upon and the transformation parameters to be used. In this case, the input data consists of a standard format display element. The transformation parameters provide the current screen origin, scale factors and clipping boundary to be used in creating the refresh code. The relationships among the various control blocks are depicted in Figure E.3.1-1.

The primary parameter block contains three pointers as depicted in Figure E.3.1-1. This block is outlined in Figure E.3.1-2. The first word is a pointer to the relevant element control block for the display element under consideration. The element may or may not be currently scheduled.* The second word contains a pointer to a conversion control block, discussed below. The third word is a pointer to the standard format display element.

* Several points may be noted here. If the element has not yet been displayed, then the element control block will essentially be null. The important point is that the user must supply an element control block area (4 words) which the transformation routine will fill in when the refresh image is created. If a version of the element already exists, the revise service will be used to replace the existing element with the new version. Consequently, any existing schedule entries are automatically adjusted.

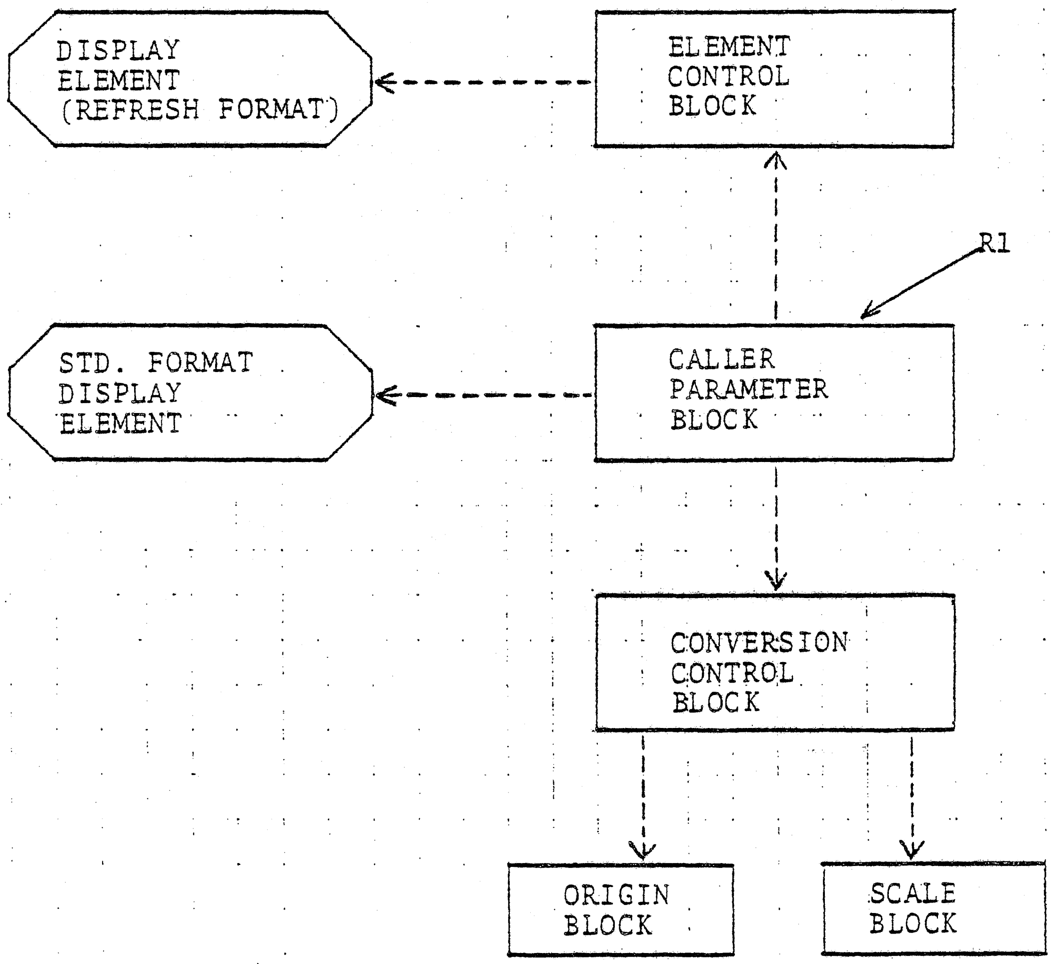


FIGURE E.3.1-1 CONTROL BLOCK RELATIONSHIPS

Word	
0	Element Control Block Pointer
1	Conversion Control Block Pointer
2	Std. Fmt Display Element Start Address

Figure E.3.1-2. Monitor Call Parameter Block

The conversion control block contains parameters which are used in transforming the standard format display element into refresh format. This block is outlined in Figure E.3.1-3. Reviewing the entries in this block:

a) Word 0 - The origin block pointer is used to fetch the definition of the current display origin.

b) Word 1 - The scale block pointer locates the current display scale factor.

c) Word 2 - The conversion factor is used to convert from global system units to display units, at SCALE = 1.0. For example, if the basic global system area spans a 1024 X 1024 mile area and the raw data is in 1/4 mile units, then a conversion factor of 1/8 would suffice to map the global area to the display area. The conversion factor and scale factor are represented as fixed point binary numbers with the binary point assumed between bits 7 and 8. Thus 1/8 would appear as 40 (octal). Generally, the conversion factor would be fixed at system design time.

d) Words 3, 4 - X min, X max define (in screen units) the X clipping boundary.

e) Words 5, 6 - Y min, Y max define (in screen units) the Y clipping boundary.

The Origin Block serves to define the current <u>display</u> origin in global system units. This block is outlined in Figure E.3.1-4. The X, Y origin values are binary integers.

The Scale Block, shown in Figure E.3.1-5, serves to define the current display scale factor. This quantity should initially be 1.0 and then adjusted to reflect operator zoom requests. The

Word	
0	Pointer to Origin Block
1	Pointer to Scale Block
2	Conversion Factor *
3	Minimum X**
4	Maximum X**
5	Minimum Y**
6	Maximum Y**

*Assumed binary point between bits 7 & 8.

**Screen Units

Figure E.3.1-3 Conversion Control Block

Word	
0	X Origin
1	Y Origin

(Global coordinate system units)

Figure E.3.1-4 Origin Block

Word	
0	Scale Factor

(Assumed binary point between bits 7 & 8)

Figure E.3.1-5 Scale Block

Scale and Origin blocks have been made independent of the main conversion control block so that multiple display elements may easily reflect new screen orientations.

The other control blocks shown in Figure E.3.1-1 (schedule, element, etc.) are standard monitor structures. Most of the various blocks can be established at assembly time.

E.3.2 Coordinate Conversions

In hopes of clarifying the operations involved in transforming a display element to refresh code, we review the equations used by the monitor service. Keep in mind that there are essentially three domains which participate in the conversion process. First, the display; second, the global system; and third, the local system for the particular display element. We define the following quantities:

X0, Y0 - Local element origin in global coordinate system units (words 6 & 7 of the element header)

CX, CY - Local X, Y conversion factors for transforming units in the local element space to global coordinate system units (words 8 & 9 of the element header)

DX0, DY0 - Display origin in global system units, i.e., point 0, 0 on the display (center screen) corresponds to DX0, DY0 in the global coordinate system (defined in the origin block)

CG - Global coordinate system conversion factor, this simply acts to convert units in the global space to screen units at SCALE (CS) = 1.0. (from word 3 of the conversion control block)

CS - The current system scale factor (from the scale block)

Then, given some point (x, y) in the local coordinate system, the corresponding screen position (dx, dy) is determined by:

G/7 Monitor Manual Addition

Manual: H-79-007

E.3.4 Notes (cont.)

(d) The number of binary places in the scale and global conversion factors is selectable at assembly time. Adjustment of these parameters is necessary in cases where the default selection (between bits 7 and 8) does not provide adequate dynamic range. The total number of binary places in the scale factor (CSPLC) and the global conversion factor (CGPLC) must equal sixteen.

$$dx = (CS \cdot CG \cdot CX) \cdot x + (CS \cdot CG) \cdot (X0 - DX0)$$
$$dy = (CS \cdot CG \cdot CY) \cdot y + (CS \cdot CG) \cdot (Y0 - DY0)$$

The various factors are combined in advance so that each coordinate conversion consists of one multiplication and one addition. (The clipping boundaries are converted to local system units and applied prior to any coordinate conversion).

E.3.3 Monitor Interface

The Display Element support routine will operate as an entry point within the Monitor services section. The external code will simply be called in subroutine fashion from the monitor area, after setting up relocation register two.

EMT Code: 13 (Hex)

Parameter Block:

Word	Contents
0	Element Control Blk Ptr.
1	Conversion Control Blk Ptr.
2	Std. Format Disp. Elmt Ptr.

Status Return Codes:

0	- Request satisfied
2	- Insufficient Memory
32	- Conversion Error

E.3.4 Notes

(a) The transformation routine does not effect the status of any schedule entries. Thus, if the element was scheduled when the call was made, it will still be scheduled on return.

(b) The standard element control block will be filled in by the transformation routine. Once the refresh image exists, the display element (refresh form) may be manipulated (enable, disable, etc.) via any of the standard monitor services.

(c) Display elements subject to scaling should be scheduled at x = 0, y = 0, since the conversion routine will apply any offset necessary to effect a new screen origin.

APPENDIX F

DEVICE HANDLER DESIGN

F.0 MONITOR I/O STRUCTURE REVIEW

The monitor I/O structure consists of an I/O supervisor and a variable number of device handlers. The I/O supervisor receives all user task I/O requests and vectors the requests to particular handlers. The specific handler is determined by mapping the logical device number (contained in the user specified I/O control block) into the task header physical device list. The physical device number is then used to fetch a particular handler address from table DEVADR in the monitor supervisor module, MNTRSP. All other decoding/processing of I/O control block entries is left to the handler.

The call to the handler is of the form "JSR PC, HNDLR"; thus an "RTS PC" should be used to exit the handler. On entry to the handler the following register configuration exists:

a) User task relocation registers are installed,
b) R0 - Subfunction (entry) number, this is just the low order hex digit of the EMT code,
c) R1 - User I/O control block address,
d) R2 - Status return code address (byte),
e) R3 - Unit number from the high order byte of the user task header entry (of significance only to multi-unit devices).

The I/O control block is the parameter block for I/O requests. The control block entries are discussed in Section 7.1.1. The I/O supervisor expects the logical unit number to be in byte 0 of the second word. Other entries are handler dependent but, in order to avoid confusion, new handlers should use the assigned entries to the extent possible.

In summary, the I/O supervisor only serves to direct the user request to a specific handler. This structure yields maximum independence for each handler and consequent efficient real time data reception. The following discussion of device handler composition is not intended as a tutorial on handler design but rather as a summary of considerations relevant to existence in the monitor framework.

F.1 HANDLER ENTRY POINTS

The basic handler entry points are outlined in Section 7.2. Of the various possible entry points, the standard monitor structure requires two; initialization (0) and I/O purge (14). The initialization entry is used during monitor initialization to prepare the handler and device for I/O operations. The I/O purge entry is used by Task Management on task exit or abort to ensure that all outstanding I/O operations are complete or cancelled. If task loads are to be done through the handler then the READ (5) and FILE QUERY (15) entries are also required. Other handler entry points may or may not be present depending on the particular device requirements and user operating situation.

The logic required for each handler entry is not generally easy to characterize due to the wide diversity of basic device types and operating modes. Typically, the devices presented are either simple(keyboards, etc.) or complex (host interface). The resulting handler structures must ultimately reflect this range of hardware complexity. In spite of obvious device-to-device differences it is possible to outline the usual handler logic flow. Figure F.1-1 presents a typical handler control section (the main entry point called by the I/O supervisor). Figures F.1-2 thru F.1-4 outline the basic logic necessary for several of the standard entry points.

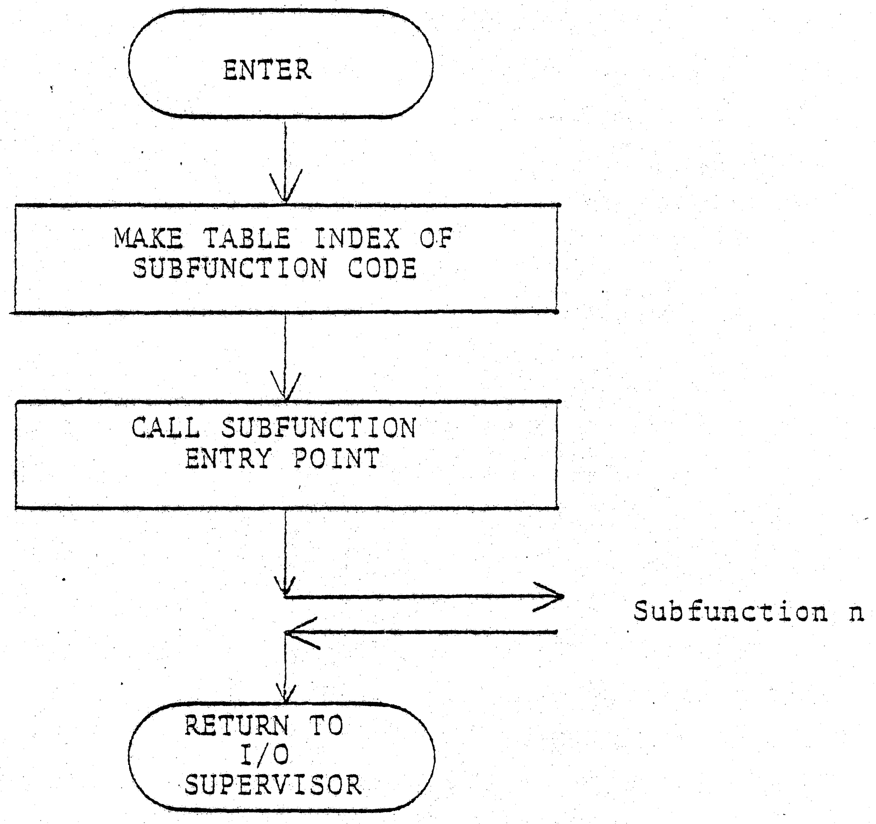
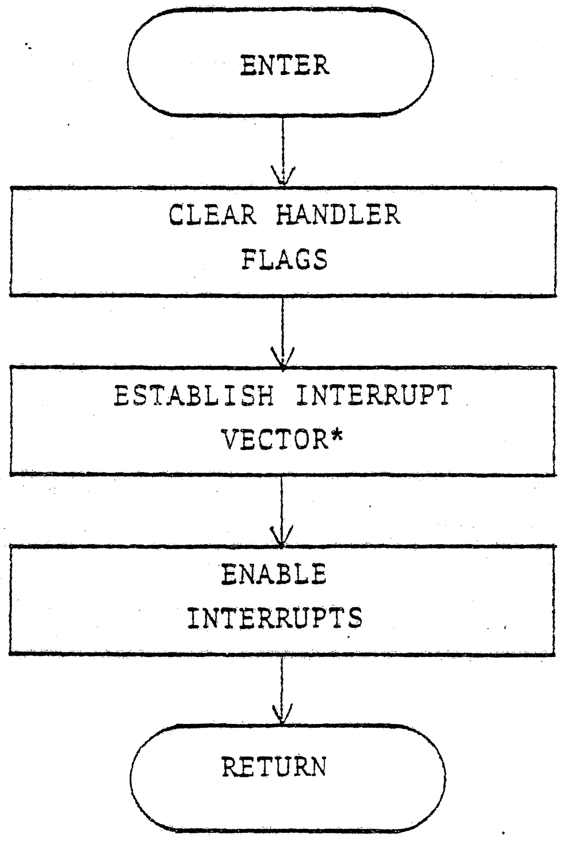

ENTER
MAKE TABLE INDEX OF SUBFUNCTION CODE
CALL SUBFUNCTION ENTRY POINT
Subfunction n
RETURN TO I/O SUPERVISOR

Figure F.1-1 Handler Main Entry



*Set PSW to priority 7 (!).

Figure F.1-2 Handler Initialization

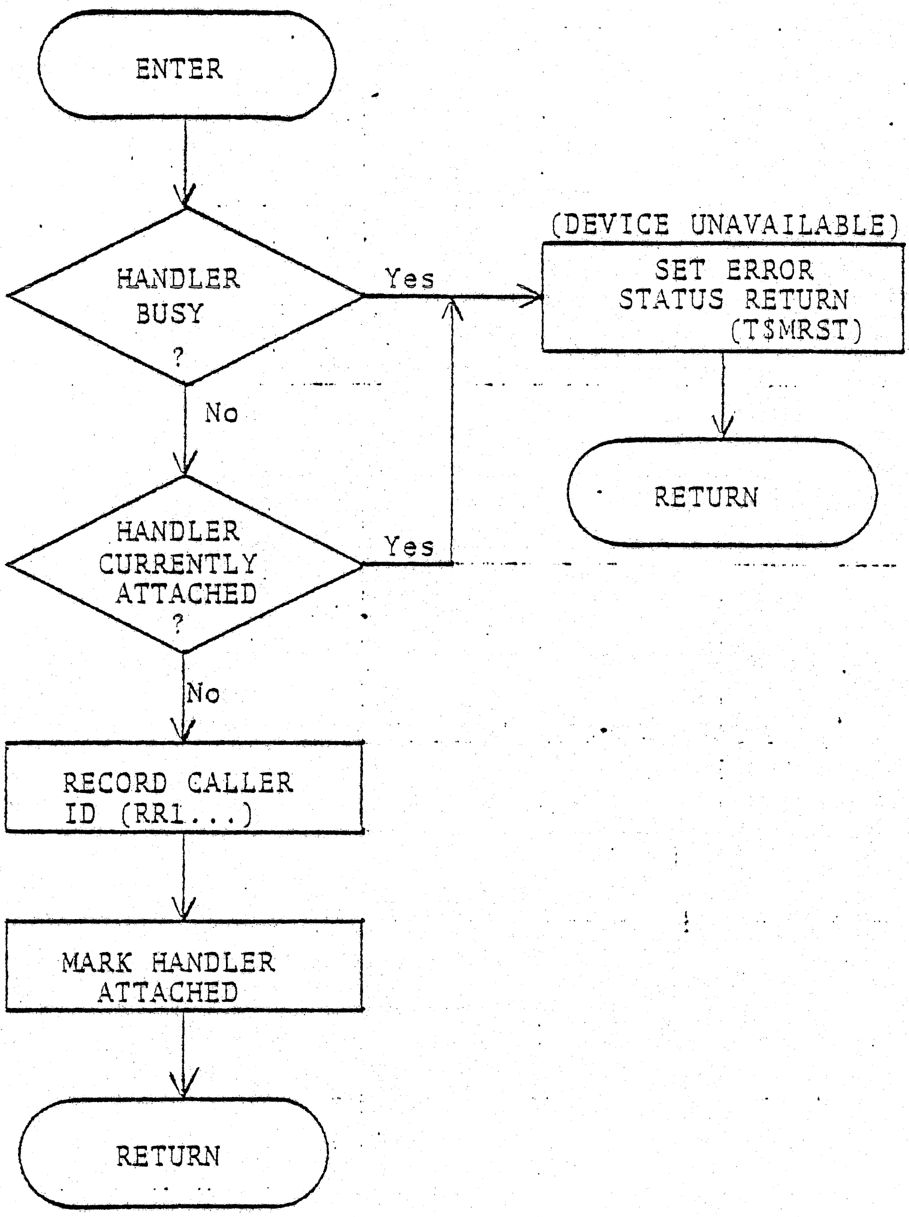

ENTER
HANDLER BUSY ?
Yes
No
(DEVICE UNAVAILABLE)
SET ERROR STATUS RETURN (T$MRST)
RETURN
HANDLER CURRENTLY ATTACHED ?
Yes
No
RECORD CALLER ID (RR1...)
MARK HANDLER ATTACHED
RETURN

Figure F.1-3 Attach Entry

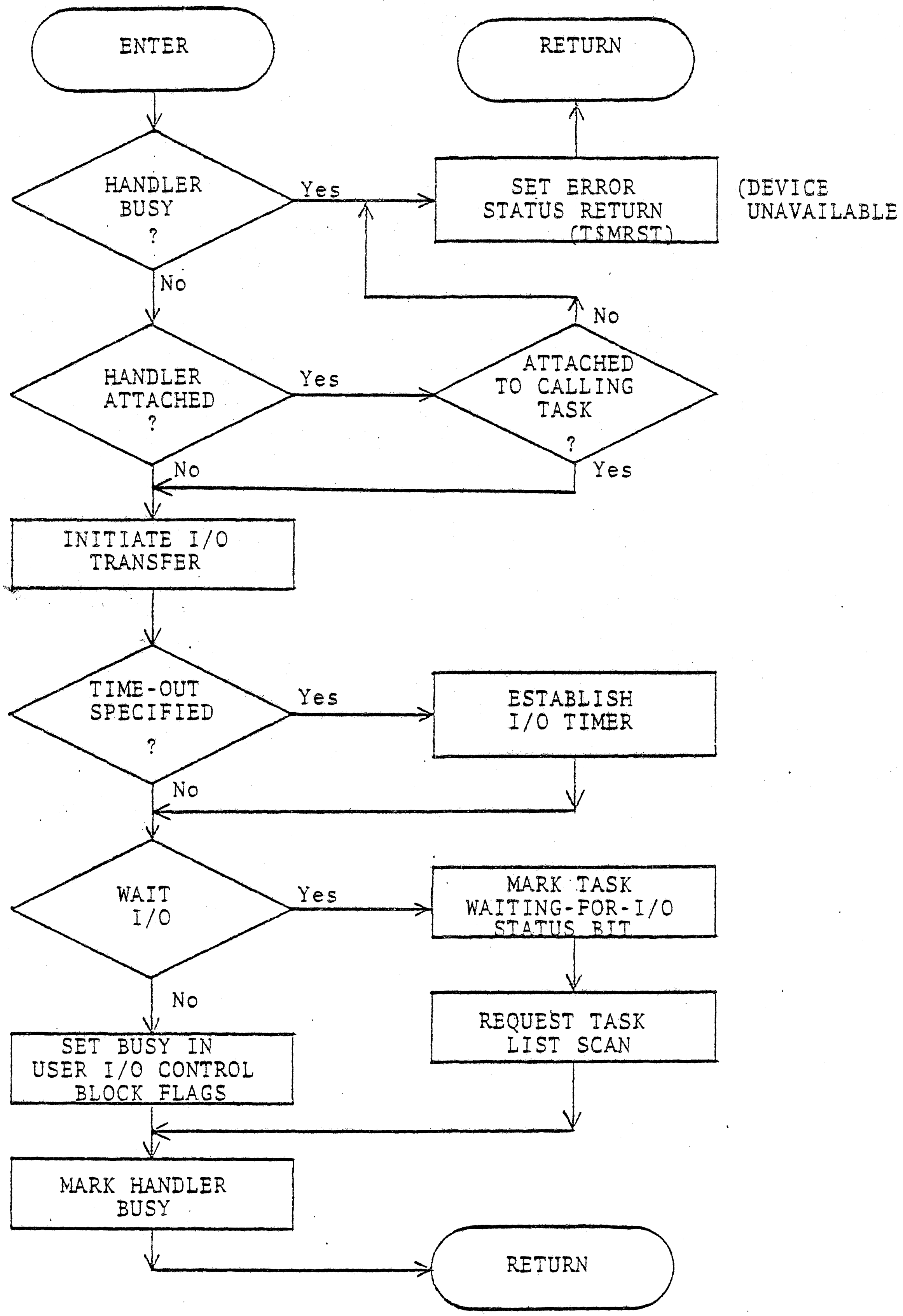


Figure F.1-4 Read Entry

F.2 INTERRUPT SERVICING

Interrupts indicate to the handler that an I/O transfer has completed. The job of the handler in the interrupt service routine is to check the final status of the transfer and indicate I/O completion to the user task. These activities are reviewed in Figure F.2-1. The common requirements of saving registers on interrupt are handled by two subroutines in the supervisor module, MNTRSP.

On interrupt the handler service routine should immediately call INTPSH to record the state of the interrupted task. (JSR PC, INTPSH).* On return from INTPSH the processor priority should be adjusted to allow, if appropriate, other interrupts. Interrupt servicing should be terminated through routine INTPOP. This is accomplished by: JMP @#INTPOP.

*There are some circumstances in which the call to INTPSH may not be necessary. For example, if the transfer always involves two words then the first interrupt can simply save the first word in a temporary and then RTI. In any case, the state of the interrupted process must be preserved.

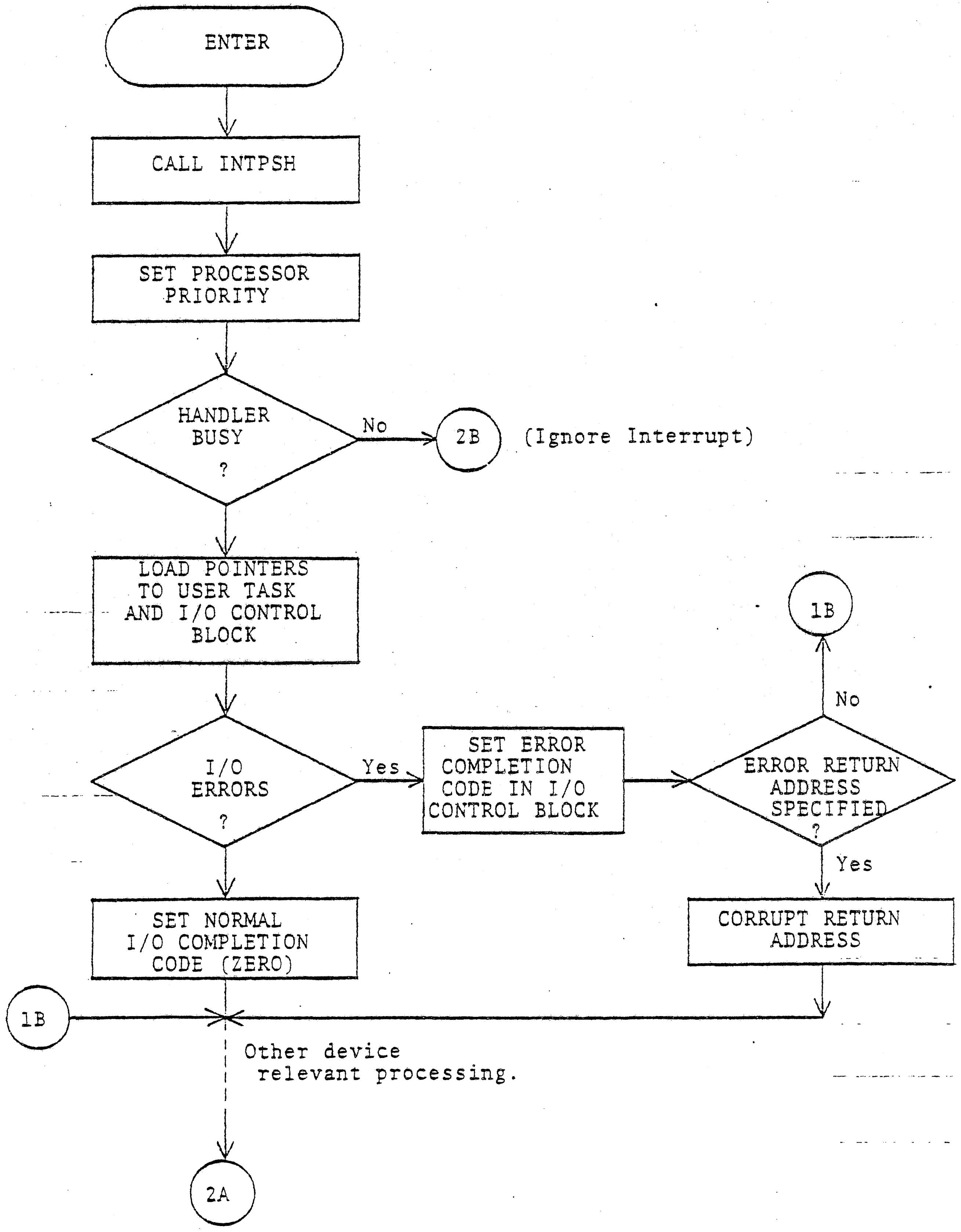

ENTER
CALL INTPSH
SET PROCESSOR PRIORITY
HANDLER BUSY ?
No
2B
(Ignore Interrupt)
LOAD POINTERS TO USER TASK AND I/O CONTROL BLOCK
I/O ERRORS ?
Yes
SET ERROR COMPLETION CODE IN I/O CONTROL BLOCK
ERROR RETURN ADDRESS SPECIFIED ?
No
1B
Yes
CORRUPT RETURN ADDRESS
SET NORMAL I/O COMPLETION CODE (ZERO)
1B
Other device relevant processing.
2A

Figure F.2-1 Interrupt Servicing
(Page 1 of 2 Pages)

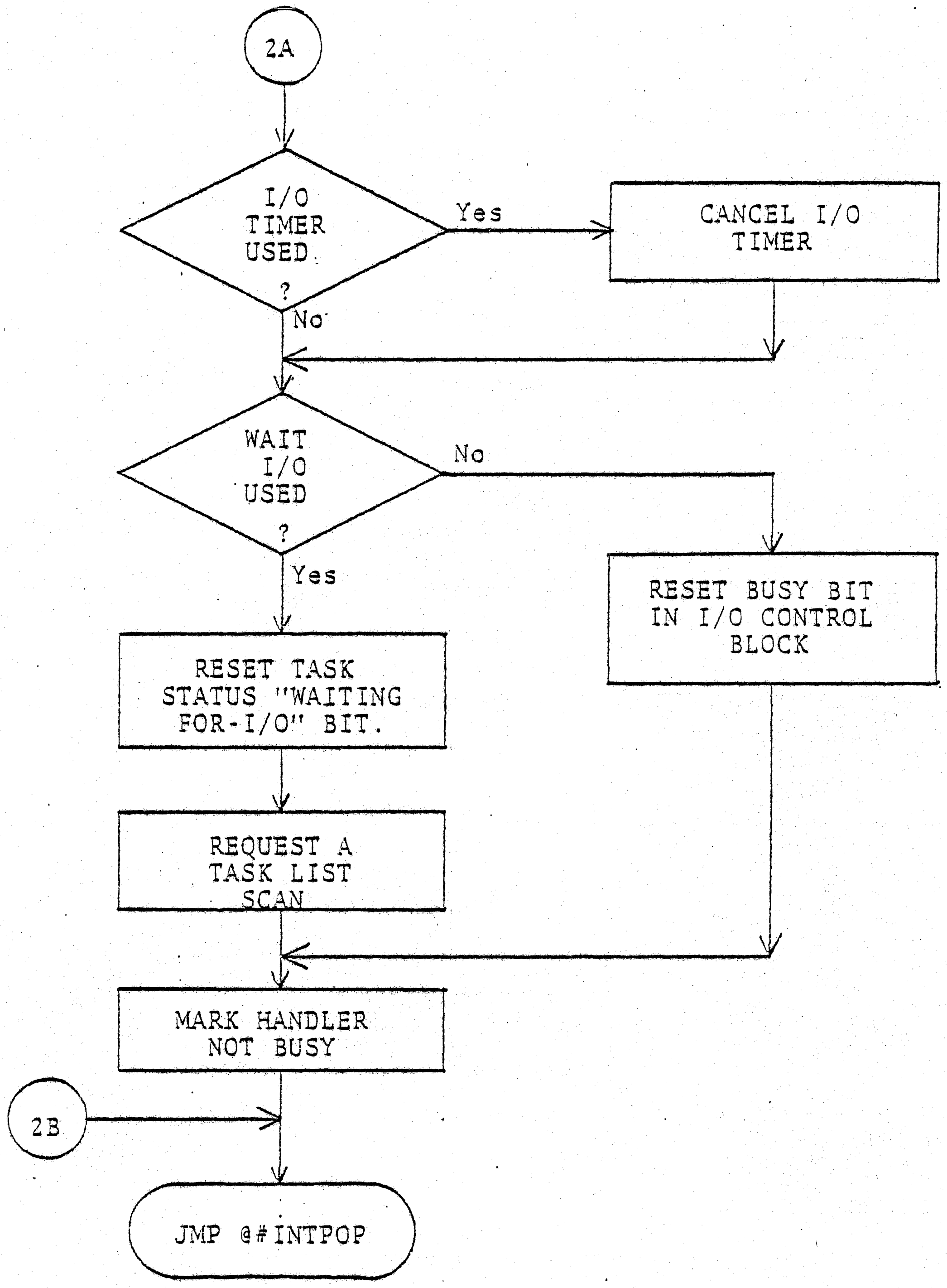


Figure F.2-1 Interrupt Servicing
(Page 2 of 2 Pages)

F.3 HANDLER INCLUSION IN SYSTEM IMAGE

New handlers are added to the system image by adjusting table DEVADR in the monitor supervisor, MNTRSP. Entries in DEVADR are simply the transfer address (label) for the handler main entry. The added label will also need to be declared global in MNTRSP. User access to the new handler (device) will be on the basis of its position in the DEVADR table. (See also Section B.3.)

F.4 INTER-PROCESSOR INTERFACING

The foregoing comments on device handler implementation are of critical importance in the design of the host computer interface and associated handler. The structure of the interface and user task access will depend on the specific data being processed. In particular, the data volume and display timeliness requirements will need to be carefully considered in order that appropriate queueing/buffering/blocking decisions are reached. A sample approach to the host interface problem is outlined in Figure F.4-1.

PARALLEL INTERFACE HANDLER

1.0 <u>HANDLER DESIGN OVERVIEW</u>

The parallel interface to the host processor serves as the primary medium for inter-processor exchange of programs, data, and system status. This role results in the necessity for defining a handler structure which goes somewhat beyond that required for the garden-variety peripheral device. Several specific operational characteristics yield requirements not generally encountered in other device handlers. First, the interface is bi-directional. This results in essentially two separate sections, one for input and one for output. Secondly, the necessity of processing real time data requires the provision of adequate buffering/queuing. Third, inasmuch as the parallel interface is our access to host mass storage, a means must be provided for accessing external files. Finally, our handler design provides a basic message distribution facility to ensure timely response to incident real time data. We next discuss the handler implementation with regard to the overall Graphic 7 Monitor framework.

2.0 <u>USER ACCESS TO THE PARALLEL INTERFACE</u>

In the interest of commonality and consequent ease of usage, the communication mechanism between user tasks and the handler generally follows that described in Section 7 of the User's Guide. To be exact, the same I/O control block is being employed with some additions/amplifications as noted below. Also, the standard entry point assignments are maintained. Before proceeding to a discussion of the specific input/output mechanisms a brief review of our data transfer assumptions.

All data on the inter-processor communication medium is transmitted in the form of a message. The standard message format to be used is shown in Figure 2.0-1. The important feature is the message type code. This code is fundamental to the handler design, as described below.

2.1 PARALLEL INTERFACE - DATA INPUT

All messages directed to the Graphic 7 will be accompanied by a message type code. The type code, one byte, is divided into message class/sub-class fields. The class field serves to direct the message to a particular task. This is accomplished by having each task indicate to the handler, via the "attach" entry, which messages are of interest to that task. Thus, in regard to input data, the parallel interface may be employed by a multiplicity of active tasks.

Figure F.4-1 Interface Discussion
(Page 1 of 5 Pages)

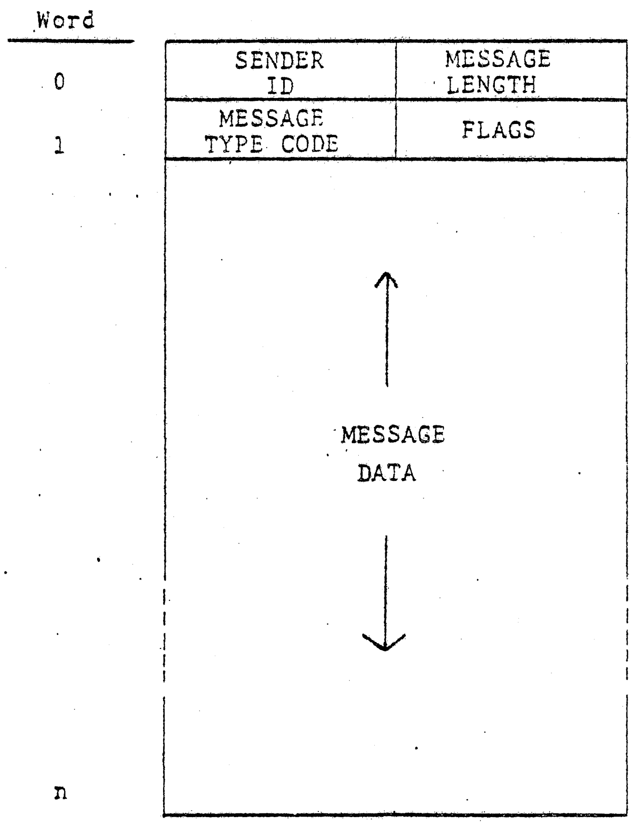


Figure 2.0-1 Message Format

Figure F.4-1 Interface Discussion
(Page 2 of 5 Pages)

In addition to message distribution, the handler also provides message queuing (in the event that the task does not have a "read" outstanding) and pointer mode. Pointer mode, indicated by setting bit 13 of word 4 of the I/O control block before the attach call, allows the user task to process the message data directly from the system buffer. In this mode the buffer pointer is placed in the user's I/O control block (word 2) instead of transferring the data to a user-specified buffer area. The pointer mode should yield significant savings in the processing of real time message data.

We review the procedure for accepting data from the parallel interface handler:

a. User selects his message class code* and places such in byte 1 of word 6 of the I/O control block. This same I/O control block must be used for subsequent read requests.

b. User selects pointer mode, if desired, by setting bit 13 of word 4 in his I/O control block. (Note that this must be selected prior to the "attach" call.)

c. The ATTACH entry (EMT 73) is used to assign the selected message class to the user task.

d. I/O read requests (EMT 75) are issued to fetch each successive message. These requests may utilize the standard I/O features of wait/no-wait and time out. For each message relayed to the user task, the length is placed in word 7 and the type code is inserted in byte 1 of word 6 of the I/O control block.

e. If messages arrive for an attached class with no outstanding read request then they are automatically queued to await the next read request.

f. The DETACH entry will be used to terminate the acceptance of a particular message class. DETACH should always be employed before a user task exits the system.

*Message class zero is a special case and cannot be attached.

Figure F.4-1 Interface Discussion
(Page 3 of 5 Pages)

2.2 PARALLEL INTERFACE - DATA OUTPUT

Data (messages) directed to the host processor will be conveyed via the WRITE entry. With respect to output data, the handler (parallel interface) represents a single device available to all tasks. Queuing is provided to allow for normal interface availability delays. Wait/No-Wait and time out are not honored on output since the data is transferred to a system buffer and control is immediately returned to the calling task. The following procedure outlines the data output sequence:

a. User inserts message (data) length in word 3 and buffer pointer in word 2 of his I/O control block. (Note: the length here does not necessarily correspond to the final, actual message length. This is simply the data portion as depicted in Figure 2.0-1)

b. The message code for the receiving processor is placed in word 7. This becomes the second word of the message header (corresponding to both the type code and flags byte).

c. A standard WRITE request (EMT 76) is issued to the parallel interface handler.

The handler output service cannot be attached or allocated.

2.3 PARALLEL INTERFACE - FILE ACCESS

It frequently becomes necessary to access data from the host machine mass storage resource. Such data will typically include task images, static display data or hardware diagnostics. The Graphic 7 Monitor assumes that any device used for task loading is capable of processing a "File Query" request. Thus, the parallel interface handler has been designed to include this feature using the standard entry point assignment. The procedure for loading an external disc file consists of first issuing the FILE QUERY (EMT 7F) and then a standard read request. The handler has been configured to use message class zero for all file related transfers. Thus, the procedure is:

a. User task configures a File Query block as described in Section 7.2.11 of the monitor user's guide. The File Query data block is relayed to the handler (EMT 7F) via an I/O control block specifying the File Query block as the I/O buffer. Obviously, the most important entry in the File Query is the file name.

Figure F.4-1 Interface Discussion
(Page 4 of 5 Pages)

b. The handler outputs the file query block to the host processor where the file status, length are filled in.

c. The Query Block is relayed back to the Graphic 7. The handler returns the data block to the user area.

d. If the user wishes to now input the file he issues a standard read request for message class zero. Since the file read mechanism must access the File Query block we make the stipulation that the File Query Block immediately follows the I/O control block employed in the file read. Thus, the read entry will assume the Query Block is at the address in R1 plus 16 bytes.

3.0 USAGE NOTES

a. All parallel interface transfers consist of an even number of bytes. User buffers should always begin on a word boundry.

b. Error return address option (word 5 of standard I/O control block) is not supported in the parallel interface handler.

Figure F.4-1 Interface Discussion
(Page 5 of 5 Pages)