

1037-R801-18-103

PRELIMINARY OMEGA

DESIGN MANUAL

Revision 1

**REGISTERED DOCUMENT  
DO NOT COPY**

Doc. # 57 Ser. # \_\_\_\_\_

Issued to Les O'Keefe

Company Confidential

Systems Programming  
March 1966

**UNIVAC**

DIVISION OF SPERRY RAND CORPORATION

UNIVAC PARK, ST. PAUL 16, MINNESOTA

## TABLE OF CONTENTS

### Purpose and Scope

1.0	OMEGA Control . . . . .	1-1
1.0	Design Philosophy . . . . .	1-1
1.1	Control Thread . . . . .	1-1
1.1.1	Task Addendum Definition . . . . .	1-3
1.1.2	Activity Addendum Definition . . . . .	1-5
1.1.3	Storage Module Definition . . . . .	1-7
2.0	Basic OMEGA . . . . .	2-1
2.1	Interrupt Processors . . . . .	2-1
2.1.1	Interrupt Processors . . . . .	2-1
2.1.1.1	Service Request . . . . .	2-3
2.1.1.2	Task Contingency . . . . .	2-10
2.1.2	Normal Hardware and Peripheral Interrupts . . . . .	2-12
2.1.2.1	Internally Specified Index . . . . .	2-13
2.1.2.2	Externally Specified Index . . . . .	2-17
2.1.2.3	Clock Interrupts . . . . .	2-23
2.1.2.4	Synchronization Interrupts . . . . .	2-25
2.1.3	Main Frame Contingency Interrupts . . . . .	2-26
2.1.3.1	Parity Error . . . . .	2-27
2.1.3.2	Power Loss . . . . .	2-28
2.2	Interrupt Support Routine . . . . .	2.2-1
2.2.1	C. P. Queue Control . . . . .	2.2-1
2.2.2	Storage Module Control . . . . .	2.2-3
2.2.3	Channel Control . . . . .	2.2-5
3.0	Input/Output Control . . . . .	3.1-1

3.1	General Description . . . . .	3.1-1
3.1.1	Random Access Storage List . . . . .	3.1-2
3.1.2	Unit Control Block . . . . .	3.1-3
3.2	I/O Director . . . . .	3.2-1
3.2.1	Preprocessing Function . . . . .	3.2-1
3.2.2	Console Handler . . . . .	3.2-4
3.2.3	Logical Lock of Mass Storage . . . . .	3.2-10
3.3	Device Handlers . . . . .	3.3.0-
3.3.0	General Description . . . . .	3.3.0-
3.3.1	Magnetic Tape Handler . . . . .	3.3.1-
3.3.2	High Speed Printer Handler . . . . .	3.3.2-
3.3.3	Card Reader Handler . . . . .	3.3.3-1
3.3.4	Card Punch Handler . . . . .	3.3.4-1
3.3.5	Mass Storage Handler . . . . .	3.3.5-1
3.3.6	1004 Subsystem Handler . . . . .	3.3.6-1
3.4	Auxiliary Routines . . . . .	3.4-1
3.4.1	Error Recovery Routines . . . . .	3.4-1
3.4.2	Interlock Routine . . . . .	3.4-2
3.4.3	Search Routines . . . . .	3.4-3
3.4.4	Initialization . . . . .	3.4-4
3.5	User Interface . . . . .	3.5-1
3.5.1	File Codes . . . . .	3.5-1
3.5.2	I/O Requests . . . . .	3.5.2-1
3.5.3	Status Codes . . . . .	3.5.3-1
4.0	Core Allocation . . . . .	4-1
5.0	Task Control Functions . . . . .	5-1

6.0	Secondary Exec Functions. . . . .	6.0-1
6.1	Content. . . . .	6.0-1
6.1.1	Method of Operation . . . . .	6.0-4
6.2	Service Functions 1. . . . .	6.2-1
6.3	Service Functions 2. . . . .	6.3-1
6.4	I/O Error Recovery . . . . .	6.4-1
6.5	Console Control. . . . .	6.5-1
6.6	Facility and Storage Assignment. . . . .	6.6-1
6.6.1	Method of Operation . . . . .	6.6-5
6.6.2	Peripheral Unit Assignment. . . . .	6.6-10
6.6.3	Random Access Storage Assignment. . . . .	6.5-15
6.6.4	Master File Directory . . . . .	6.6-22
6.6.5	Facility Assignment Initialization. . . . .	6.6-26
6.7	Service Functions 3. . . . .	6.7-1
6.8	Cooperative Service Routine. . . . .	6.8-1
6.9	Pre-Selection. . . . .	6.9-1
6.10	Selection. . . . .	6.10-1
6.11	Termination. . . . .	6.11-1
6.12	Remote Facility Assignment . . . . .	6.12-1
6.13	Library Service Routine. . . . .	6.13-1
6.14	Checkpoint and Restart . . . . .	6.14-1
6.15	Compactor. . . . .	6.15-1
6.16	Dump Routine . . . . .	6.16-1
6.30	Secondary Exec Tables . . . . .	6.30-1
6.30.1	Selection Job Stack. . . . .	6.30-1
6.30.2	Job Description. . . . .	6.30-5
6.30.3	Selection Summary. . . . .	6.30-11
6.30.4	Selection Facility Map . . . . .	6.30.18

7.0	I/O Cooperative Mechanism. . . . .	7-1
7.1	Cooperative Features. . . . .	7-1
7.2	I/O Cooperative Elements. . . . .	7-2
7.3	Input Unit Record Routines. . . . .	7-5
7.3.1	Sample Input Unit Record . . . . .	7-7
7.4	Cooperative Control . . . . .	7-11
7.4.1	Calls. . . . .	7-11
7.4.2	Cooperative Library. . . . .	7-21
7.4.3	Functional Routines. . . . .	7-24
7.4.4	Block Charts . . . . .	7-26
7.5	Cooperative Maps and Tables . . . . .	7-30
7.6	Output Unit Record Routines . . . . .	7-35
7.7	Edit Routines. . . . .	7-36
8.0	Real Time and Communications Control.. . . .	8.1-1
8.1	General Description . . . . .	8.1-1
8.1.1	Level 1 Control. . . . .	8.1-1
8.1.2	Level 2 Control. . . . .	8.1-2
8.2	Tables. . . . .	8.2-1
8.2.1	CTM Control Block. . . . .	8.2-1
8.2.2	Unit Control Block . . . . .	8.2-3
8.2.3	Communications Facility Map. . . . .	8.2-5
8.3	Buffers . . . . .	8.3-1
8.3.1	Communication Buffers. . . . .	8.3-1
8.3.2	Communication Buffer Chain Control . . . . .	8.3-3
8.3.3	Packing Buffers. . . . .	8.3-4

8.4	ESI Control. . . . .	8.4-1
8.4.1	ESI Interrupt Processor . . . . .	8.4-1
8.4.2	ESI Function Executor . . . . .	8.4-2
8.4.3	Channel Initialization and Termination. . . . .	8.4-4
8.5	Communication Handler. . . . .	8.5-1
8.5.1	Interrupt Processor Interface . . . . .	8.5-1
8.5.2	Data Handling . . . . .	8.5-3
8.5.3	User Interface (Level 1). . . . .	8.5-4
8.5.4	Communications Director Interface (Level 2) . . . . .	8.5-4
8.6	The Communications Director.....	8.6-1
8.7	User Program Interface Level 2 . . . . .	8.7-1
8.8	Communications Facility Assignment . . . . .	8.8-1
8.9	Level 2 Control Example. . . . .	8.9-1
9.0	Auxiliary System Routines . . . . .	9.0-1
9.1	Loader . . . . .	9.1-1
9.1.1	General Description . . . . .	9.1-1
9.1.2	Primary Control Statement . . . . .	9.1-2
9.1.3	Secondary Control Statement . . . . .	9.1-3
9.1.4	Relative Binary Code. . . . .	9.1-8
9.1.5	Loader Operation. . . . .	9.1-14
9.1.6	Input Element Format. . . . .	9.1-16
9.1.7	Function of Loader Phase 1. . . . .	9.1-23
9.1.8	Function of Loader Phase 2. . . . .	9.1-30
9.1.9	Function of Loader Phase 3. . . . .	9.1-37
9.1.10	Table Transitions . . . . .	
9.2	Library Maintenance. . . . .	9.2-1
9.3	Test Package . . . . .	9.3-1

9.4	Element Library Maintenance . . . . .	9.4-1
9.5	Utility Package . . . . .	9.5-1
9.6	RExecutor . . . . .	9.6-1
9.6.1	Method of Operation . . . . .	9.6-1
9.6.2	The REX Control Statement . . . . .	9.6.2-1
9.6.3	Secondary Control Statements . . . . .	9.6.3-1
9.6.4	RExecutor Elements . . . . .	9.6.4-1
9.6.5	RExecutor Phase 1 . . . . .	9.6.5-1
9.6.6	Diagnostic Messages . . . . .	9.6.6-1
9.6.7	RExecutor Phase 2 . . . . .	9.6.7-1
9.6.8	Execution Area . . . . .	9.6.8-1
	Blockcharts . . . . .	9.6.8-9

## Purpose and Scope

This document is an attempt to illustrate and explain the general composition of OMEGA from a technical aspect by showing the major routines encompassed and interaction between them. Tables, maps and flow charts are included where appropriate and available. In general, this document will serve as a guide for implementation and upon completion of the system serve as technical documentation for the exec. Therefore it will be, by nature, subject to constant change and/or refinements until date of release, hence, caution must be used in placing reliance on the contents of this document.

It is assumed the reader is familiar with the UNIVAC 494 hardware and has reviewed the functional description manual for the UNIVAC 494 operating system.



## OMEGA

### 1.0 Design Philosophy

OMEGA is a set of integrated routines providing the basic control for coordinating and executing UNIVAC and user provided programs. It provides a flexible and reliable foundation for the installation environment to build upon. OMEGA has been designed explicitly modular to facilitate future extensions, expansion of particular functions or selection of available variants of a basic function. In the following discussions of the executive functions, a distinction is made between "basic exec activities" those required for switching, queuing and hardware operation of the machine; and "secondary exec activities" those used for selection, task control, input/output and termination. All secondary exec activities, although required in some form, have been established by software convention and are replacable. To facilitate their replacement or change, all secondary exec activities operate as worker programs activated through formal service request with minimal communication between them. This document supplies necessary tables, calling sequences and logic of OMEGA to accomodate user modifications to the UNIVAC provided system.

### 1.1 Control Thread

OMEGA is dependent on random access storage as an operating base and upon primary input streams as a source of the control language definition of work to be performed. Each unit of work to be performed is called a task and represents the unit upon which the system performs its selection, allocation and activation functions.

For each task entered into the system, a Task Addendum is formed, which contains all pertinent information pertaining to the task (see figure 1-1). To control the running of the task and requested fragments, Activity Addendums are formed to order and control fragmentation.

In addition to Activity Addendums, free storage is needed to perform service requests or switching upon interrupts. This additional storage is assigned to the particular program when it is required. The core area assigned is called a storage module and is twenty octal words. This storage module is linked to the Activity Addendum through a push-pop chain. The storage module contains all the information necessary for any service request or loss of control through an interrupt. The format of an allocated storage module is shown in figure 1-3 and the entries are described in Storage Module Definition.

TASK ADDENDUM FIG. 1-1

0	29	22	21	18	17					0			
		PRIORITY		TASK ADDENDUM LINK									
1	29					18	17	ACTIVITY ADDENDUM LINK				0	
2	29	MAXIMUM # OF CP UNITS ALLOWED				TASK (200 us)				0			
3	29	# OF CP UNITS USED				(200 us)				0			
4	29	TIME TASK LAST ACTIVATED								0			
5	29	22	21	20	19	18	17	16	15	14	JOB NUMBER	0	
		U	O	A	B	C	D	E					
6	29	F.P. OVERFLOW ADDR.				15	14	F.P. UNDERFLOW ADDR.				0	
7	29	ERROR ADDR.				15	14	ILLEGAL OP ADDR.				0	
10	29	DRUM INCREMENT TO PARAMETERS										0	
11	29	# OF 100 WORD GROUPS				18	17	TASK ADDRESS BASE				0	
12	29					18	17	CORE CHAIN LINK				0	
13	29	# OF WORD REQUESTED				15	14	RECEIVE ADDRESS				0	
14		UNASSIGNED										0	
15													
16													
17													
20	29	PRIMARY OUTPUT ESTIMATE				15	14	SECONDARY OUTPUT EST.				0	
21	29	DRUM INCREMENT BEGINNING OF MODULE CHAIN										0	
22	29	DRUM INCREMENT END OF MODULE CHAIN										0	
23	29					18	17	CURRENT BUFFER ADDR.				0	
24	29	TOTAL # OF MODULES				15	14	# IN SYSTEM				0	
25	29	DRUM INCREMENT BEGINNING OF MODULE CHAIN										0	
26	29	DRUM INCREMENT END OF MODULE CHAIN										0	
27	29					18	17	CURRENT BUFFER ADDR.				0	
30	29	TOTAL # OF MODULES				15	14	# IN SYSTEM				0	
31	29	DRUM INCREMENT BEGINNING OF MODULE CHAIN										0	
32	29	DRUM INCREMENT END OF MODULE CHAIN										0	
33	29					18	17	CURRENT BUFFER ADDR.				0	
34	29	TOTAL # OF MODULES				15	14	# IN SYSTEM				0	
35	29	28	27					18	17	ADDRESS			0

SWITCHES  
RELATIVE TO  
TASK BASE

PRIMARY  
INPUT

PRIMARY  
OUTPUT

SECONDARY  
OUTPUT

FILE CODE A  
FILE CODE B-Z

67  
70  
71  
72  
73  
74  
75  
76  
77

PRIMARY INPUT UNIT RECORD
PRIMARY OUTPUT UNIT RECORD
SECONDARY OUTPUT UNIT RECORD
COOPERATIVE LIBRARY
SYSTEMS LIBRARY
JOB LIBRARY
SYSTEMS LOG
SCRATCH
SCRATCH

FILE CODES ZA → ZI  
RESERVED FOR  
SYSTEM USAGE

### 1.1.1 Task Addendum Definition

The task addendum is formed by cooperative service routine when a Job stream enters the system. Each task (program) described by the Job stream will utilize this addendum to contain I/O assignments and other control information at the task level. Upon completion of a task the addendum is purged of task dependent information and re-used for next task in the stream. This will allow select control information and I/O assignments to be carried from one task to the next within a Job stream.

The following describes content of task addendum:

- Word 0 Service priority is a four bit number assigned as the highest priority available to the task. Linked to next task addendum contained in the system.
- Word 1 Linked to activity addendum is a 18 bit address of first activity established for current task.
- Word 2 Contains the estimated # of time units required of the control processor to complete the current task. 7--7 indicates continuous processing.
- Word 3 The # of time 200 us units currently used by task at this point in time.
- Word 4 Time of day (in 200 increments) this task was activated.
- Word 5 Logical switches - bits  $2^{15}$ - $2^{19}$  are used for logical switches A through E respectively. If corresponding bit is set (1) switch is on; if bit is 0, switch is off. Overflow switches - if bit  $2^{20}$  is set (1) a floating point overflow has occurred. If bit  $2^{21}$  is set (1) a floating point underflow has occurred. Lower contains a 15 bit Job Number assigned by the OMEGA to identify the Job.
- Word 6 Contains relative addresses of alternate floating point overflow or underflow routines established by task.
- Word 7 Contains relative addresses of alternate error recovery routine in the event the task addresses outside its assigned code. Illegal operation address is relative address of alternate illegal instruction recovery routine.
- Word 10 Contains drum increment to parameters stored by a task unsolicited operator enter for conveyence to next task. Drum increment is to logical File Code ZD.
- Word 11 Contains absolute code address assigned to current task and length of task in groups of  $a100_g$ .
- Word 12 Contains a link to chain descriptors committed by RT/Comm programs only.
- Word 13 Contains address and # of words for an outstanding RECEIVE operator.
- Word 14-17 Unassigned
- Word 20-34 Are used to control primary and secondary input/output and are explained under cooperative control.
- Word 35-66 Contains the basic set of logical file codes A-Z which the user may assign peripheral devices or random storage files. Each entry is composed of one word ordered A through Z and contains address of unit, control block describing peripheral device, list of mass storage descriptors or an expanded list of 26 file codes.

File code entry

1	1	1	26	8	17	ADDRESS	0
0	0	0					

- (1) indicates assignment to be held between tasks
- (1) indicates address is of random storage list
- (1) indicates base address of an additional set of 26 file codes

Word 67-77 contains file codes ZA-Z1 reserved for systems usage.

### 1.1.2. ACTIVITY ADDENDUM DEFINITION

An activity is established by definition of an operating task and/or activity. The function allows a dynamic declaration of parallel or asynchronous paths through the task level code. Activity addendums are used to control and register eligible control paths.

One Activity Addendum is formed upon initiation of the task. This is used to start the task. One additional activity addendum will be formed for each activity registration or fork performed by the task. These activity addendums will be used (in conjunction with assigned storage modules) by the dispatcher as switching points control running of the task and task fragments within the system. Activity addendum require 10g words.

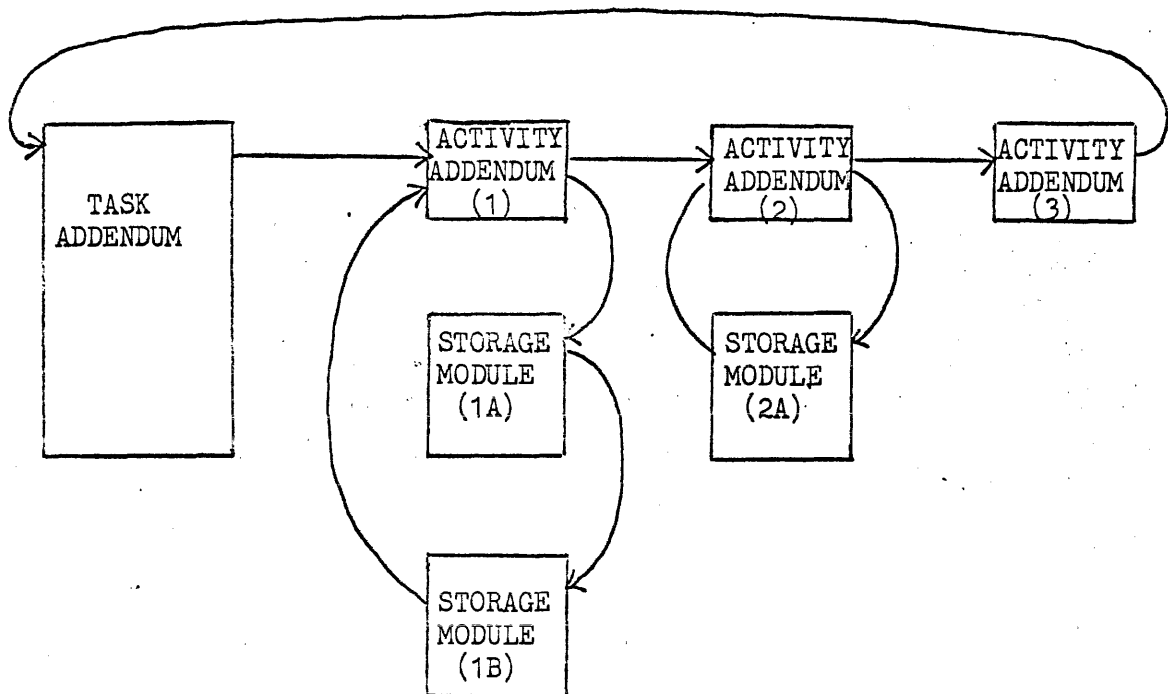
The following describes the contents of the activity addendum:

- Word 0 Queue cell for central processor control will be used to link all activity addendums to the C. P. Control Queue. Also used to queue activities to hold or delay queues.
- Word 1 Storage module link used to link together all storage modules assigned to the activity. This points to the last module in the chain which will be used for first return of control.
- Word 2 Priority number of this activity (0-36). Used to control selection of highest priority activity eligible for control from C. P. queue. Binary identify is number assigned by used for queued activities.
- Word 3 Address of the task addendum
- Word 4 Fork control. Bit 29 indicates if a JOIN has been given next 12 bits contain the number of forks this activity has made. Lower 18 bits used to link all forked addendum. If queue processing activity the lower 15 bit contains the relative starting address. If other activity the word is zeroes.
- Word 5 Linking cell for all activity addendums associated with the task.
- Word 6 Contains the upper 12 bits of IFR and the relative index register of the activity.
- Word 7 Program lock register.

ACTIVITY ADDENDUM FIG. 1-2

0	29	18	17	0	ZERO FILL	QUEUE CELL FOR C.P. CONTROL OR HOLD QUEUES
1	29	18	17	0	ZERO FILL	STORAGE MODULE LINKING CELL
2	29	15	14	0	ZERO FILL IF OTHER ACT.	
	29	15	5	4	BINARY IDENT IF QUEUE PROCESS	ZERO PRIORITY NUMBER
3	29	18	17	0	UNUSED	TASK ADDENDUM ADDR.
4	29	28	18	17	J FORK COUNTER	FORK JOIN CONTROL CELL
	29	15	14	0	UNUSED	REL. START ADDR. FOR QUE PROC
	29			0		ZERO FILL IF OTHER ACTIVITY
5	29	18	17	0	ZERO FILL	ACTIVITY ADDENDUM LINKING CELL
6	29	18	17	0	UPPER 12 BITS OF IFR	RIR
7	29			0		PLR
	29	26	15	14	11	10
10	29			0		UNUSED
11	29			0		UNUSED

The following illustrates the chaining employed for task addendum. Activity Addendums and storage modules used by the OMEGA to control a program.



The task addendum describes the task code and contains all pointers to peripheral assignment descriptions used by the task code. The presence of three activity addendums implies three control threads through the program or parts of the program have been declared by the code at object time. Each activity addendum is normally registered to the dispatcher queue as an eligible point for program control providing for parallel or asynchronous processing within a program. This is in addition to the normal multiprogramming and/or multiprocessing of tasks within the system.

The presence of two storage modules under activity addendums 1 implies the registered control thread has performed a service request to the OMEGA which has: 1) caused an additional service request by the activated OMEGA element which has called on a third element which is currently in control of the control processor, 2) the called for element was interrupted and requeued to the dispatcher, 3) the called for element has taken the activity out of the system until some disposing action has been completed for the request, e.g., I/O completion, freeing of.



#### 1.1.4 STORAGE MODULE DEFINITION

The allocated storage module is a 20 octal word core area assigned to control interrupts or service requests of activities. The storage module is always placed on an even core address so service routines (switcher, interrupt processor, handler) can take advantage of the memory overlap feature of the 494.

Entry Word	Use
0	18 bit link to previous storage modules if any or back to addendum. Use of control last in first out sequence of requests.
1	18 bit address of the addendum this request is associated with. Stored in when module is assigned and used by servicing routines to find addendum.
2	Internal function register store in at time of interrupt.
3	Relative index register of the activity interrupted.
4	Program lock register of the activity interrupted.
5 Upper	Upper 15 bits in the captured relative P a interrupt. Stored in after Store worker B is done.
5 Lower	Worker B <sup>1</sup> registers store in at interrupt by store.
6-13	Lower 15 bits of word 7 and 10 and lower 18 bits of words 11-14 contain worker registers. Additional bits may be used by service routines for additional storage.
14	Contents of A register at interrupt.
15	Contents of Q register at interrupt.
16-17	Additional storage for service routines.

ALLOCATED STORAGE MODULE

Located on Even Core Address

0	29	ZERO FILL	18	17	PUSH POP LINK	0
1	29	ZERO FILL	18	17	ACTIVITY ADDENDUM ADDR.	0
2	29	IFR				0
3	29	ZERO FILL	18	17	RIR	0
4	29	ZERO FILL	26	25	PLR UPPER BOUND	0
5	29	RELATIVE P	15	14	ZERO FILL	0
6	29		15	14	LOWER BOUND	0
7	29	ADDITIONAL	15	14	WORKER	0
10	29	STORAGE FOR	18	17	B REGISTERS	0
11	29	SERVICING ROUTINES	18	17		0
12	29		18	17		0
13	29		18	17		0
14	29	A REGISTER				0
15	29	Q REGISTER				0
16	29	ADDITIONAL STORAGE				0
17	29	ADDITIONAL STORAGE				0

Figure 1-3

## 2.0 Basic OMEGA System

The basic OMEGA system is divided into two sections, the interrupt processors which answer all hardware and software interrupts and the related interrupt support routines which assist the interrupt processors in disposing of the interrupt. The second group, the interrupt support routines, include routines such as queue control and the dispatcher.

### 2.1 OMEGA INTERRUPT PROCESSORS

The OMEGA interrupt processors for the 494 operating system shall operate on three distinct types of interrupts. These three types are a) task or task fragment generated interrupts, b) normal hardware and peripheral generated interrupts c) main frame contingency interrupts.

#### 2.1.1 Task or task fragment generated interrupts.

These generated interrupts are either requesting service from a part of the operating system or are contingencies relating only to the specific task. The service request interrupts and the relative servicing routines will operate as an extension of the requestor. Task related contingencies interrupts usually result in the queuing of the contingency interrupt processor for subsequent control.

##### Service Request Interrupts

Service requests are normally made by the use of the Exec return instruction and an associated packet or registers containing the parameters required to process the request. The only exception to this is the segment load that runs off of a guard mode interrupt. The organization of the routines which respond to the service request function is shown in figure 2-1.

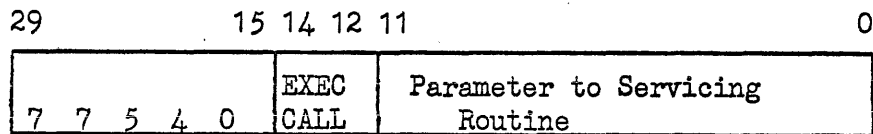
Each routine handling of a service request may itself make service requests for other functions to be preformed. Much of the communication between elements of OMEGA is preformed in this manner; some service requests therefore, are internal to the system and will not be available to the general user. This organization contributes to the modularity and open ended design and allows the system to dynamically expand when core is available and contract as core is required by the user.

### 2.1.1.1 Service Request Routine

This element of Omega is the interface by which an operating program communicates with and requests services of Omega. The requestor is reactivated only when some disposing action is complete. Result parameters are returned to indicate non-performance, normal completion or abnormal completion.

An operating program requests service by a sequence of instructions which submit a parameter packet appropriate to the request and interrupts to Omega. Since hardware guard mode is enforced against operating tasks, the special executive entry instruction is used to submit a request. This instruction causes an interrupt and includes a 15 bit field identifying the function requested.

Exec return instruction



where EXEC CALL represents the number of the resident EXEC function required to process or schedule the request and are as follows:

- 0 Task Control and Miscellaneous Service Functions.
- 1 Input/Output service request at the packet level.
- 2 Contents supervisor, schedule request for a drum based service routine.
- 3 I/O cooperative control, process primary and secondary Input/Output requests.
- 4 Core allocator, responsible for allocation and deallocation of core storage.
- 5 Common subroutine linkage, responsible for establishing linkage between a common subroutine and its caller.
- 6 Reserved for system use
- 7 Reserved for installation usage

The lower 12 bits of the Exec return instruction contain additional parameters to direct the servicing routine.

Upon detection of the Exec return interrupt the service request routine performs the following functions for all Exec returns with a non-zero operand.

- . All transitory registers (A, Q and worker B registers), IFR and the captured relative P will be stored in a storage module that contains the activities PLR and RIR. This storage module is then linked to the activity addendum through the storage module linking cell.
- . A new storage module will be acquired from the unallocated storage module chain and interrupt lockout will be released.
- . IFR, RIR and PLR will be set to the values for the Basic Exec routine responsible for handling the request and these values will be stored in the new storage module.
- . Program control is switched to the called basic Exec routine with the "A" register containing the IFR value (with the Exec call) and the "Q" register holding the Activity Addendum address. (The assigned storage module can be found from storage module link cell)

The servicing of the standard requests by the primary service functions are explained in the related individual section. Only the special service functions performed by the Basic Omega system are explained in this section.

## Special OMEGA Service Functions

Because many OMEGA routines need special functions for queueing and switching and do not have direct access to the queue tables and storage area, a special OMEGA return will be used by secondary OMEGA routines to request service from the basic OMEGA system. This return may only be used by OMEGA routines and is not allowed to worker programs.

77540 00000

with the parameters and service request number in the A and Q registers. Service request number is in A lower.

The servicing routines will operate either interrupts locked out or logically locked out.

IDLE Service request number is zero

This entry makes a direct entry to the dispatcher to select the next eligible task or task fragment for central processor control.

### Queue Control

Queueing of tasks or activities to be performed is a major function of OMEGA. Besides the requirement for queueing at the occurrence of an interrupt, many of the servicing routines will require queueing functions to return control to worker programs, queue other functions to be performed, and queue themselves if requests must be delayed. A common queue control routine will be employed to perform the functions.

Queue Control can be referenced by servicing routines by an EXEC return instruction. The call is for an immediate function to be performed and will be performed with interrupts locked out or a logical lock imposed to protect over queueing or double referencing of the queue tables.

### Central Processor Queue

The CP queue is the queue that directs the use of central processor control. Any routine that is eligible for control (and not currently in control) is listed on the ready queue. Servicing routines are required to have placed on the C. P. queue requests that have been completed. Other functions also are required to place entries on the C.P. queue (i.e., Program load to start program, etc.).

### C. P. queue with idle

This group places activities on the C.P. queue and does not require return of control. An entry to the dispatcher will be made after queue control has performed its task.

Current Activity at its priority Service request number is 1.

Queue the activity currently running at its registered priority and exit to switcher. May be used to return control to worker upon completion of a service request.

Current Activity at priority X Service request number is 2.

Same as number 1 except the priority is defined by the upper 5 bits of the Q register. Useful to displace routines for a turn through the dispatcher.

Addressed activity at its priority. Service request number is 3.

Queue the activity addendum addressed by the lower 17 bits in Q register at its registered priority. The Q register contains a 17 bit absolute addendum address to go on the ready queue. Priority will be taken from the addendum.

Addressed activity at a specified priority. Service request number is 4.

Same as number 3 except the priority is supplied as an additional parameter in the upper 5 bits in the Q register. Useful to displace routines or have them gain control at a priority other than that assigned to them.

### C. P. Queue with return of control

This group places activities on the C.P. queue but wishes return of control for subsequent processing. Control will be returned immediately following the EXEC Return after the queueing function has been performed. Subsequently an idle entry must be made to allow the queued routine to get control. Current activities cannot be queued with return of control because no queue cell link would exist for interrupt queueing.

Addressed activity at its priority. Service request number 5.

Same as number 3 except return control after queueing.

Addressed activity at a specified priority. Service request number 6.

Same as number 3 except priority is supplied as a parameter in the upper 5 bits of the Q register and control is returned after queueing.

#### Switch Functions

Switch functions are used by I/O control, contents supervisor, common subroutine linkage and RT/Comm control to switch control from itself to a routine being activated and to gain subsequent return of control.

#### Direct Switch 1 Service request number 7

Reset RIR and give control to the address specified by RIR. Q register contains a 17 bit address of the RIR to reset to.

#### Direct Switch 2 Service request number 10

Reset IFR, RIR and PLR and give control to the address specified by RIR. Q register contains a 17 bit address of the list. Where IFR RIR and PLR are stored.

#### Direct Return A Service request number 11

Reset and return control to the primary exec routine whose identity is contained in Q lower. Q contains one of the following numbers.

- "1" to return to I/O Director
- "2" to return to Contents Supervisor
- "3" Not Applicable
- "4" Not Applicable
- "5" to return to Common Subroutine linkage
- "6" Not Applicable
- "7" to return to Real Time Control

#### Direct Return B Service request number 12

Deallocate last SMOD on current activity addendum and return control to it. Used to return control to worker, etc. without going through switch routine.



POP Service request number 13

Compliment of Push. Remove from chain and queue for return of control the indicated activity addendum. Control will be returned to requestor upon completion. The Q register contains the address of the chain cell for FIFO sequence or it is the address of the entry in the chain immediately before the one the requestor wants popped.

Initiate ISI I/O Service request number is 14

This request comes from the formator with the Q register containing the address of the SMOD with all parameters in it. These parameters include channel number, queue placement etc.

Initiate ESI I/O Service Request number is 15

Link Task Addendum. Service Request Number 16

This request asks for the formed task addendum whose 17 bit absolute address is contained in the Q register to be added to the chain of task addendums. The new addendum will be linked to the task addendum currently in control and all other links will be updated.

De-link Task Addendum. Service Request Number 17

This request asks that the task addendum whose Job number is contained in the Q register be removed from the chain of all task addendums.

Switch Control Thread. Service Request Number 20

This request asks that the 17 bit Activity Addendum address in Q be switched to the task addendum address by the Job number contained in A upper, that the indicated addendum be queued for control and an idle entry made to the dispatcher.

### 2.1.1.2 Task Contingency Interrupts

Task contingency interrupts are hardware interrupts that impact only on the related task and do not prevent the rest of the system from continuing. These interrupts include:

#### Illegal Instruction

This interrupt is caused by attempting to execute a 00 or 7700 instruction code or attempting to execute a privileged instruction. Standard systems action is to abort the offending task. Tasks may, however, pre-establish an illegal instruction recovery routine. If established, the routine will be eligible for control under the offending activity addendum. Additional parameters besides transitory registers will be available to the routine.

#### Program Protection

This interrupt is caused by a program trying to read, store or jump outside its assigned area. With the exception of the informal segment call (which is explained in segmentation) the task will be aborted with information about the violation logged by the abort routine.

Timeout

Floating Point Interrupt

Test and Set Interrupt

### 2.1.2 Normal Hardware and Peripheral Interrupts

These interrupts usually result in the execution of waiting functions and generate a request for service. The servicing of the interrupts will operate at varying priorities. They may have been established from the requesting task, priorities of waiting functions, optimum peripheral usage or other system considerations. Disposal of the interrupt will usually result in the removal of an activity from a delay queue and placed on the central processor queue for subsequent control.

### 2.1.2.1 Internally Specified Index Interrupts

Control of ISI interrupts is handled by two routines. The ISI processor and the channel control block processor.

#### ISI Processor

The ISI processor is activated by the ISI interrupt and runs with interrupts Lock out. If the interrupt was an external interrupt, the interrupt word and both input and output BCRs are save (these are not saved on a monitor interrupt). The next function queued on the interrupted channel control block is then executed. Upon completion of the execution of the I/O commands, the ISI processor checks to see if the Channel Control Block processor can be run. If it cannot, because of a higher priority routine in control, the channel control block is marked as needing processing and a return to the point of interrupt is made. The CCB must then be processed in it priority to the rest of the interrupt processing before control can be returned to any worker program. If no higher priority routine is in control, such as EST processors etc., the Channel Control Block processor will be entered directly from the ISI processor after saving the interrupt point.

#### Channel Control Block Processor

The Channel Control Block processor is a routine that operates logically non-interruptable and prepares the CCB for the next execution and processes the previous function and queues interrupt analysis if necessary.

Channel control blocks are used to control input/output operation for all standard peripheral channels. One CCB is set up for each channel containing standard peripheral hardware. A table of addresses called CCB table is kept to be able to access the CCB by channel number. The CCB is shown in figure 2.1-1.

CHANNEL CONTROL BLOCK

Label Ref-Upper or Whole

Label Ref-Lower

KCCEXTINT	0	29	EXTERNAL INTERRUPT STORAGE			0	
KCCINBER	1	29	INPUT BCR AT EXT INTERRUPT			0	
KCCOUTBCR	2	29	OUTPUT BCR AT EXT INTERRUPT			0	
KCCWAIT	3	29	WAIT FOR INTERRUPT INDICATOR	15	14	TEST EXT INT INDICATOR	0
KCCITYPE	4	29	TYPE OF INTERRUPT INDICATOR	15	14	QUEUED INTERRUPT INDICATOR	0
KCCPROC	5	29	CCB PROCESS IND.	15	14	CHANNEL LOCKOUT IND	0
KCCSWIND	6	29	SWITCHING INDICATOR	15	14	5 CHANNEL NUM	0
KCCIOGCTL	7	29	28	18	17	ADDR. OF LAST I/O GROUP EXECUTED	0
	10	29	28	18	17	ADDR. OF NEXT I/O GROUP TO EXECUTE	0
KCCCEMCTL	11	29		18	17	CEM CHAIN	0
KCCINTADDR	12	29		18	17	INTERRUPT STORE ADDR.	0
KCCUCBC	13	29		18	17	UCB CHAIN	0
KCCFUNC	14	29		18	17	ADDR. OF FUNCTIONAL CHAR. OF CHANNEL	0
KCCLOCK	15	29		18	17	ADDR. OF LOGICAL LOCK TABLE	0

Figure 2.1-1

Explanation of Entries in the channel control block.

Word #

- 0 Used to temporarily store the external interrupt before it is moved to the CEM. Not used on monitor interrupts.
- 1 Contents of the Input BCR at external interrupt time.
- 2 Contents of the Output BCR at external interrupt time.
- 3 Upper half is wait for interrupt indicator and is set or cleared depending on the exit used from the CEM. Lower half contains an indication of a by-passed I/O sequence on an external interrupt.
- 4 Upper half is interrupt type indicator  
00000 = External    77777 = Monitor    40000 = Error (Time OUT, Parity errors)  
Lower is used to indicate an interrupt has occurred when the channel was logically locked out.
- 5 Lower half is channel lock out indicator and if set it means the CCB has not been updated to accept the interrupt.  
Upper half is used to indicate the channel has to be processed before OMEGA goes logically interruptable.
- 6 Switch indicator - if clear switch to analysis on high priority, if set return to interrupted activity unconditionally. Lower half contains the channel number.
- 7 Address of last I/O group executed used to control return to analysis. Bit 29, if set, indicates there was no previous function.
- 10 Address of next I/O group to execute. Used to execute next I/O sequence. Bit 29, if set, indicates there is no new function to execute.
- 11 Channel executor module chain control word.
- 12 Address to move interrupts for device handler analysis.
- 13 Address of first unit control block in the UCB list for this channel.
- 14 Address of the functional description of this channel. Used by device handlers for function and interrupt codes, etc.
- 15 Address of Logical Lock List for this subsystem.



The CCB processor upon completion of updating the CCB enters an interrupt support routine that check for other interrupts in order of priority and switches control to the appropriate processor.

#### 2.1.2.2. ESI Interrupt Processing Routines

The function of the ESI Interrupt Processor is to answer ESI interrupts, identify the interrupting ESI, re-establish activity on the I/O channel, provide an additional communication buffer if required, and perform any other functions as pre-directed by the Communication Handlers. The ESI Interrupt Processor is distinct from, and has priority over, the ISI Interrupt Processor.

The ESI processor consists of three routines:

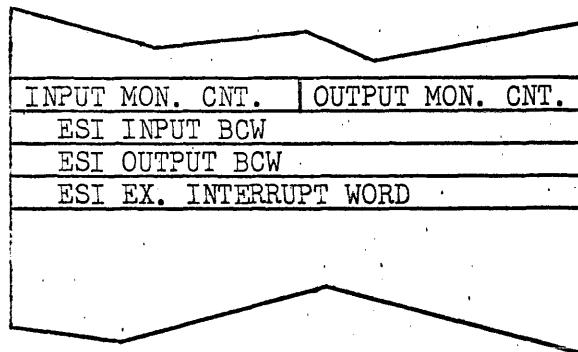
- .The ESI Interrupt Answering Routines
- .The ESI Interrupt Processing Routine
- .The ESI Buffer Chain Control Routine

## Interrupt Answering Routine

- Function** - The ESI Interrupt Answering Routine identifies terminating ESI, re-establishes activity on the interrupting channel and insures an orderly path through the ESI Interrupt Processing Routine.
- ENTRY** - Entry to the interrupt answering routine is from the three ESI interrupt entrance locations.
- EXIT** - The routine exits either to the ESI Interrupt Processing Routine or to the point of interrupt (Exit to the point of interrupt occurs only when the ESI or ISI Interrupt Processor has been interrupted).
- Operation** - The ESI Interrupt Answering Routine operates in the Executive Mode and with interrupts locked out (the result of being entered from an interrupt). Three distinct entrances are required because of the different manner in which the input monitor, output monitor, and external ESI interrupts must be handled initially.

The input and output monitor entrances must capture the terminating ESI and then re-establish channel input or output activity. If the address contained in the terminating ESI points to a previously defined buffer, the buffer is established (see Section 8.3). The external interrupt entrance need not re-establish the channel.

The routine now increments the ESI interrupt count in the CTM Control Block Table (Figure 2.1-3) and accesses the location in the table corresponding to the terminating ESI. This location contains the address of the CTM Control Block. This Control Block, explained in Section 8.2.1, contains control information for the line. The section of the Control Block shown below is used by the answering routine to store information for the ESI Interrupt Processor.



The Answering routine increments either the input or output monitor count depending on the type of interrupt. If the count was zero the contents of the terminating ESI location is stored in either of the next two words depending on type. If an external interrupt occurred, the word on the line is stored in the last word shown. The contents of the terminating ESI enables the Interrupt Processor to examine the chain links of the communication buffer to determine whether or not a new buffer must be obtained. The input and output counts are used to determine the number of interrupts on this ESI when a buffer chain has been previously set up by the handler. They are of value only when extremely small buffers have been specified or when an overload condition exists.

After storing the proper controls as described above the Answering routine switches from the Executive mode to the Worker mode, enables interrupts, and passes control to this ESI Interrupt Processing Routine. If the Interrupt Processor was already in control when the ESI interrupt occurred, control is returned at the point of interrupt.

#### ESI Interrupt Processing Routine

- Function** - The ESI Processing Routine functions to obtain or release communication buffers from the core chain, and informs the appropriate communication handler (via the QREF function) of the interrupt if previously instructed to do so by the handler.
- ENTRY** - Entry is from the ESI Interrupt Answering Routine.
- EXIT** - Exit is a return to the ESI Interrupt Answering Routine for the release of the logical lockout and for switching if necessary.
- Operation** - The ESI Interrupt Processing Routine operates as an extension of the Interrupt Answering Routine. The difference between the two routines is that the Answering Routine operates with interrupts lockout while the Processing Routine operates with interrupts enabled, but in a logically non-interruptable mode.

Upon gaining control from the answering routine the processor interrogates the indicated CTM Control Block to determine the type of interrupt. If an input monitor interrupt occurred the routine checks the Input Buffer Link of the last communication buffer to determine whether or not a new input buffer is needed.

If the link is not zero the Answering routine has already established the link as the new buffer. If the link is zero the processor obtains from the buffer chain an input buffer of the size stated in the CTM Control Block and places the BCW in the appropriate ESI address and also in the Input Buffer Link position of the terminating buffer. The processor then examines the input monitor section of the "Interrupt Control" word in the CTM Control Block to determine the next course of action. The possible alternatives are:

- . QREF the handler indicated in the CTM Control Block conveying the addresses of the CTM Control Block and the terminating buffer.
- . Do not QREF the handler. This means that the handler is scanning the buffer on a certain time interval and is not interested in the monitor interrupt.
- . Send a "Look for Sync" external function to the CTM (applicable only into synchronous CTM's. This option may be used in combination with either of the above.

Upon receiving indication of an output monitor interrupt the processor examines the Output Buffer Link of the terminating ESI and the output monitor section of the "Interrupt Control" word. The options are:

- . Return the terminating output buffer to the buffer chain (would not be used if the same message was being sent to more than one CTM).
- . QREF the handler on every output monitor interrupt conveying CTM Control Block and buffer.
- . QREF the handler only when the Output Buffer Link of the terminating output buffer is zero. Thus, if the message is broken up into a chain of buffers, the handler will be notified only when the last buffer of the chain has terminated.

Upon receiving indication of an external ESI interrupt the processor will QREF the handler indicating the CTM Control Block the interrupt status, and the contents if the input ESI address at interrupt time. Buffer chaining may be performed. As an option the "Interrupt Control" word may specify that a "Look for Sync" function be sent to the CTM at this time.

After processing an interrupt or series of interrupts (as determined by the count in the CTM Control Block) the processing routine clears the indicators in the CTM Control Block and checks the interrupt counter in the CTM Control Block Table for other ESI interrupts which may have occurred while the Processor was operating. If no other interrupts have occurred the Processor returns control to the Answering Routine for switching.

#### ESI Buffer Chain Control

The ESI Buffer Chain Control Routine may properly be considered a part of the ESI Interrupt Processing Routines since it must be accessed directly by the Interrupt Processor for fast acquisition of input buffers. The routine must also be available to the communication handlers for acquiring and releasing ESI input and output buffers. The handlers cannot enter the routine directly and, therefore, must use an EXECUTIVE RETURN instruction. In view of this dual usage of the routine, the description of this routine along with the ESI buffering scheme will be given in Section 8.0, "Communication and Real Time Control".

CTM CONTROL BLOCK TABLE

MPLX	CLT	PRIOR	INTERRUPT COUNT	
			29	18 17 0
00	00-01		PRIORITY	CTM CONTROL BLOCK
00	02-03			
	⋮			∅
00	76-77			CTM CONTROL BLOCK
01	00-01			
	⋮			
	⋮			
NN	76-77			CTM CONTROL BLOCK

FIG. 2.1-3

The CTM Control Block Table points to the core location of the CTM Control Block associated with the corresponding ESI. There is one entry in the table for every ESI in the system. If there is no CTM at that location or if the CTM has not been acquired, the lower 18 bits of that location are zero. The first word of the table is a count of the number of ESI interrupts to be processed. The remainder of the table is arranged according to ESI address. The correct location in the table is referenced by subtracting the base address, dividing by 2 (right shift one bit), and adding the table base plus one. The upper twelve bits of each table entry is an increment to another location in the table. The sequential arrangement of these increments defines the processing priority of the ESI interrupt.

### 2.1.2.3 Clock Interrupts and Clock Control

Time and clocks are controlled by the table shown in figure 21-2. The entries in this table are

#### Word Number

- 0 A 30 bit 24 hour clock maintained from the real time clock interrupts. This clock contains a count of the number of 200 microseconds intervals that have elapsed since the previous midnight or other arbitrary starting time to the time of the last Real Time clock interrupt.
  - 1 Contains the last value set into the real time clock (current time of day to 200 hrs can be calculated by the difference of the RTC and the last RTC setting plus the value in word zero). This value is set from the difference between current time and the time to activate the next entry on the list.
  - 2 Contains a chain cell for all activities linked to the time delay table. Last word of the addendum is used to hold the 24 hour time to reactivate the activity.
  - 3 Contains a chain cell to link all activities to be activated on gross time of the day clock. Clock time is in the last word of the addendum.
- Note; Chains linked to words 2 and 3 are ordered according to relative activation time.
- 4 Current date is maintained in word 4 in the form YY DDD



TIME CONTROL TABLE

0	29	30 Bit 24 Hour clock 200 us granules (Time at Last Timer Interrupt)		0
1	29	18	17	0
		Zero Fill	Last RTC Setting	
2	29		17	0
			Timing Chain Cell	
3	29		17	0
			Day Clock Time Cell	
4	29	DATE		0

Figure 2.1-2

#### 2.1.2.4 Synchronization Interrupts

### 2.1.3 Main Frame Contingency Interrupts

These interrupt routines such as parity error and power loss will operate at highest priority until recovery procedures are effected and it is determined that the system can continue to run or a system abort is necessary with its own recovery procedures.

### 2.1.3.1 Parity Error Interrupts

### 2.1.3.2 Power Loss

## 2.2 Interrupt Support Routines

Interrupt support routines are those not directly related to interrupt processing but are necessary for switching and basic control of the system.

### 2.2.1 C.P. Queue Table Definition

Two tables are required to control the C.P. queue (see figure 4). Each of these tables is 40<sub>8</sub> words. There is one entry for each possible priority (0-36) and one for Exec functions to be run during idle time (core compaction, Drum compaction, etc.). Each entry in Table A contains the 17 bit addendum address of the first task or task fragment eligible for control in the associated priority group. (These may be completed service requests, interrupted activities, or programs just loaded and ready to run.) If the contents of any entry in Table A is zero it indicates that there are no activities eligible for control in the associated priority group. Table B entries contain the 17 bit addendum address of the last task or task fragment eligible for control in the priority group indicated by the position within the table. Within the activity addendum of the task or task fragment addressed by the entry in Table A is an address linking the first addendum in any priority group to the next addendum in the chain. This linking continues until the end of the chain of addendums is reached and this last addendum address appears in the corresponding priority entry in Table B. As task or task fragments become eligible for control, the entries in Table B are used to get to the last addendum, the link is made and the entry in Table B is updated to point to the new addendum address added to the C.P. queue chain. The Switcher is used to provide the transfer of control to the first eligible task or task fragment. The Switcher inspects entries in Table A, selects the next candidate, updates the entry in Table A to the next addendum in the chain, and transfers control to the selected activity. The exec functions entered in the last table location are only selected if no task or task fragment is eligible for control.

READY QUEUE CONTROL TABLES

Priority		
0		1st Entry Priority 0
1		1st Entry Priority 1
2		If=0 No Entries this Priority
3		
4		
32		
33		
34		
36		1st Entry Priority 36
37		1st EXEC FUNC TO BE RUN IDLE DURING IDLE

0		Last Entry Priority 0
1		Last Entry Priority 1
2		
3		
4		
32		
33		
35		
36		Last Entry Priority 36
37		LAST EXEC FUNC TO BE RUN DURING IDLE

Figure 4

### 2.2.2 STORAGE MODULE CONTROL

Storage modules are used to contain register settings for a control thread (activity) upon execution of Exec return or lost control and re-enter point (LCR) due to interrupt.

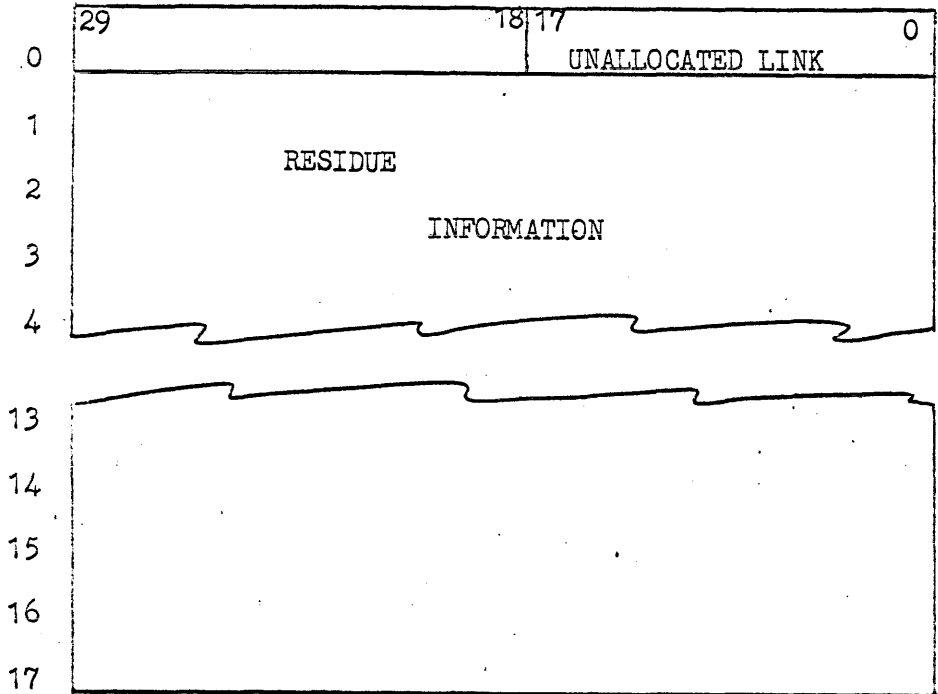
Because of the need for storage modules to run tasks and tasks fragments, some of the available core storage will be dedicated to OMEGA for storage modules. This core will be acquired from core allocation who is responsible for assigning and keeping track of all available core. The number of modules needed is dependent on the systems configuration and activities registered with OMEGA for running.

The core that OMEGA acquires will be placed in an unallocated storage module chain. Each module will have a link associated with it (see figure 1-4) that will point to the next available module in the chain. The control table for this chain is shown in figure 1-5. The first address held in this four word table is the address of the first available storage module. Each storage module then has the address of the next available module. The last module has zeros in the link indicating the end of the storage module chain.

In addition to the storage module chain, OMEGA will have one extra module. Each time that OMEGA leaves the Exec mode and allows interrupts and service requests the address of this available module will be held in one of the seventeen bit Exec B registers. If a service request or an interrupt requires the use of the module, it will be linked to the addendum and a new module will be obtained from the storage module chain. As service requests and interrupts are disposed of, the modules that were assigned will either be put back in the storage module chain or will be used as the available module OMEGA holds when giving control to the activity of non-Exec mode service routine.



UNALLOCATED STORAGE MODULE



If Unallocated link is zero, this is the last available module. Non-zero is the address of next module in chain.

Figure 1-4

STORAGE MODULE CHAIN CONTROL

	29	18	17	0	SMOD CHAIN CELL	KSMODCHAIN
KSCNASG	29	15	14	0	NUM of SMOD'S Assigned CHAIN	KSCNLEFT
KSCNOUT	29	15	14	0	NUM of SMOD'S in use	KSCMOUT
	29	15	14	0	Num of entries in Free Core	KSCEFCHAIN

Figure 1-5

Note: Last two words are used to gather statistical information about SMOD uses.

### 2.2.3 Channel Control

Due to a number of enhancements in 494 hardware operating under control of OMEGA the peripheral device handlers will not be allowed direct control of the input/output logic. The list below includes some of the reasons necessitating this change.

- 1) Common interrupt locations for all channels
- 2) Required use of channel select register
- 3) Peripheral may be attached to CPU through IOC, normal channel or both.
- 4) Non-standard device handlers
- 5) Modularity within OMEGA
- 6) Increased speed of response at occurrence of interrupt to submit next function of channel.

Without direct control of I/O the device handler will be required to communicate with the routine that will control all input/output operation. The communication will be performed through a list of Input/Output Command Words. The format and description of the command word is shown in figure X. The device handler will create the command words, function words for the specific peripheral, and buffer control words. This information will be passed to the OMEGA subroutine responsible for all I/O operation through register A, Q, and B7.

When control is assumed, B4 has the 17-bit address of the storage module wherein the registers A, Q, and B7, containing pertinent information, are stored.

STORED in SMOD for use of GEM FORMATTER:

- A Contains 17-bit address of the CHANNEL EXECUTOR MODULE previously formed, after which present GEM is inserted in chain. If A<sub>20</sub> is set, negative control will be returned immediately after queueing with no return of control upon execution.
- Q Number of command words in upper and channel number of I/O in lower.
- B7 17-bit address of the word used by Device Handler; used for priority number, request ordering number, etc.

B7+1 INTERRUPT STORE ADDRESS

B7+2 First command word in list.

Format of EXEC RETURN instruction is 77540 0000

## INPUT OUTPUT COMMAND WORD FORMAT

	1	2	3	4	5	6	7	
29	27	26	25	24	23	22	18	17
DEF			C	W	I	T	COUNT	ADDR. OF FUNCTION OR BCW LIST
								0

### Explanation of Fields

1) DEF Bits 29-27--Definition Field 3 bits.

- 000 Send terminate for channel logic.
- 001 Not used.
- 010 Activate buffer with monitor.
- 011 Activate buffer without monitor.
- 100 Send external function word to peripheral.
- 101 Read time and send external function put time in Input BCR of interrupt store.
- 110
- 111

2) C Bit 26 Chain indicator 1 bit

- 1 Chain this command word with next command word
- 0 Do not chain this command word

3) W Bit 25 Wait indicator 1 bit  
(Used only if chain bit is not set)

- 1 Wait for interrupt
- 0 Do not wait for interrupt - none expected.

4) I Bit 24 Input/Output indicator 1 bit

- 1 Output from CPU to peripheral
- 0 Input from peripheral to CPU

5) T Bit 23 Test for external interrupt indicator 1 bit

- 1 Test for external interrupt before executing I/O
  - 0 No test for interrupt.
- If the test bit is set a test for external interrupt will be performed before executing the I/O defined by this command word group. If the previous interrupt was an external interrupt bypass the execution of this I/O command word group.

6) COUNT

Bits 22-18 Count field 5 bits. Five bit count of the number of entries in the list specified by the list address.

7) ADDR

Bits 17-0 Address of the list 18 bits. Absolute address of the list associated with the command word. The lists may be functions words for peripheral devices or buffer control words. Not used with terminate.

### 2.2.3.1 CHANNEL EXECUTOR MODULE

The Channel Executor Module (CEM) is a variable length core storage area used to queue and execute request through the hardware function executor. The core storage is dynamically acquired from the core allocator. The CEM is constructed by the FORMATTER routine from the command word list and other information specified in A, A, and B7. This module is then queued to the appropriate CCB for subsequent execution of the input-output commands.

Explanation of entries in figure 5

Word number

- |             |   |
|-------------|---|
| 0           | Contains a 17 bit linking address to chain all requests to the CCB.   |
| 1           | Available for device handler use, e.g. Priority number, request ordering number.  |
| 2           | INTERRUPT STORE ADDRESS - where external interrupt and the contents of the BCR registers are stored at interrupt time.  |
| 3 bit 29    | Must be set zero to test positive to distinguish end of module.   |
| 3 bits 0-14 | Number of words in the following I/O Group. Used to control execution. Includes I/O instructions, EXIT Jump, and associated data words.   |
| 4,5         | Variable number of input-output instructions to be executed. All I/O instructions must be modified by index register B7. Address portion is an increment from the number of words entry to the associated I/O Data word, i.e. If word 10 is associated with I/O instruction at word 4, the increment would be 00005. B7 would contain the address of the # words entry. |
| 6           | Exit instruction. This is a jump instruction to leave the CCB. The exit jump is set up depending on the type of processing required.  |

Word Number

- 9,10 Data words associated with previous input-output instruction, i.e. Function word, BCW, etc.
- 11-14 Another group of I/O instruction data words and associated control information as explained above.
- x+2 Second word in CEM-variable placement.  $2^{29}$  is set to 1 to test negative for end. Bits 0-16 contain a 17 bit addendum address to activate upon completion of the I/O functions.
- x+3 Last word in CEM contains the count of the number of words to release back to free core and the address of the CEM. Use to release the core upon completion of the I/O sequence.

CHANNEL EXECUTOR MODULE (494)

0	29	18	17	CCB LINK	0
1	29	DEVICE HANDLER USE			0
2	29	INTERRUPT STORE ADDRESS			0
3	29	15	14	No. of words in group	0
4	29	17	15	14	0
	I/O INSTRUCTION	7		∅ ∅ ∅ ∅ X <sub>1</sub>	
5	29	17	15	14	0
	I/O INSTRUCTION	7		∅ ∅ ∅ ∅ X <sub>2</sub>	
6	29	EXIT JUMP			0
7	29	DATA WORD ASSOCIATED WITH I/O			0
10	29	DATA WORD ASSOCIATED WITH I/O			0
11	29	15	14	No. of words in group	0
12	29	17	15	14	0
	I/O INSTRUCTION	7		∅ ∅ ∅ ∅ X <sub>3</sub>	
13	29	EXIT JUMP			0
14	29	DATA WORD ASSOCIATED WITH I/O			0
X					
X+1	29	DATA WORD ASSOCIATED WITH I/O			0
X+2	29	28	18	17	0
	1	ACTIVITY ADDENDUM ADDR.			
X+3	29	18	17		0
	WORD COUNT	CEM ADDRESS			

Group A  
(example)  
X<sub>1</sub> = 4

X<sub>2</sub> = 5

Group B  
(example)

X<sub>3</sub> = 3



## THE CHANNEL EXECUTOR MODULE FORMATTER

The command words are formatted in a list - and necessary data and information concerning the necessary operations are stored in the registers. On occurrence of the exec return instruction the registers are stored in a storage module and the address of the module is furnished in B4.

The first step is to get core allocated for the formation of the module. A scan is performed on the command word list checking the number of data words associated with each command word and the occurrence of a chain indicator to determine the total number of words necessary. The routine then puts the number of locations necessary into Q and the number of the variable length chain into A and makes an exec return to get the needed core allocated.

Upon return of control, A will have the address of the first location of the allocation. The routine then stores data into the module and begins the formation of the module. The indicator bits in the command words are tested to determine whether a terminate, an activate buffer, or a send external function instruction is specified. The instruction is set up and the address and count of the data words is stored in the lower 23 bits of the word until it can be processed later. Command words are examined and I/O commands set up until the end of the first subgroup of I/O commands and the necessity of a wait or no wait test. A jump is set up to either the wait or no wait location for use during execution. The routine maintains a count of the number of I/O instructions to determine the increment to be stored in the lower of each I/O instruction.

After the jump has been established, the routine retrieves the address and count of data words from each I/O instruction, masks in the necessary B7 indicator, inserts the increment, and links the data to its associated I/O instruction. It continues to format the subgroups until the supply of command words becomes exhausted.

It then sets a one bit in  $2^{29}$  in the first location after the last data word to signal the end of instructions and stores the word count and CEM address in the last word. The routine then makes a check to see if the command word routine wants return of control. If it does the activity addendum address is set to zero. If not, the activity addendum address is retrieved from the SMOD and stored in the next to last address. An exec return gives control to the CEM initiator where the module is queued or initiated depending on the condition of the CEM chain.

### 3.0 Input/Output Control

#### 3.1 General Description

The general sequence of events required to perform an input/output request are described and illustrated as follows:

- . An operating activity performs an EXEC return instruction specifying a function code and with B7 set to the address, relative to lower lock limit of requestor, of the packet describing the request.
- . Upon receiving the EXEC return interrupt a storage module is allocated and control passed to the I/O Director. The I/O Director retrieves the file code from the packet and locates the cell in task addendum for the code.
- . If the request is for random access storage, the assignment cell contains the address of the storage list from which the logical drum address is mapped to the channel address. Once channel address is formed it is checked for end-of-file, lock list options are performed, and the address of the unit control block retrieved. If the request was for a unit assignment, the file code cell contains the address of the Unit Control Block (UCB) directly.
- . Upon determining UCB the I/O Director retrieves free core to use as work area and switches control to indicated device handler. Device handler forms the requests into work storage, determines where on the channel queue the request should be placed to optimize channel usage and passes control to channel queue control which enters the request on the chain. The request will subsequently be processed by the Hardware Function Executor (See Section 2.2.3).
- . The Hardware Function Executor gains control from an I/O interrupt which activates the following sequence of events: 1) capture interrupted location and store contents of channel; 2) execute next function of the channel; 3) queue storage module for completed request which will eventually reactivate device handler; 4) ready next queue request for execution; 5) return control or queue interrupted program as dictated by channel option.
- . Upon completion of the function by the Hardware Function Executor, the device handler's analysis phase is automatically schedule and activated. The status word is analyzed and reformatted, and control returned to I/O Director. The I/O Director releases free core obtained for request, clears any indicated lock list, and requests the return of control to the requesting activity.

### 3.1.1 Random Access Storage

When random access storage had been assigned to a file code the list is required to contain descriptors of the file.

Pointer from task add. for file code	Word 1	DRUM ADDRESS		Access storage from last assignment
	2	CH	# OF WORDS	
	3	DRUM ADDRESS		Describe one continuous area
	4	CH	# OF WORDS	
	5	LINK TO UCB		
	ADDITIONAL 3 WORD DESCRIPTORS OF CONTINUOUS AREAS			
	N	1	LINK TO UCB	

Drum address is relative to beginning of channel, # of words are the number of continuous cells from drum address. "Ch" is the channel number of subsystem, and "Link to UCB" is the address of the Unit Control Block describing the device handler.

Words 1 - 2 Describe random access storage assigned but not required by the request. This area will be used to satisfy any expansions of the file.

Words 3 - 5 Describe a continuous area of mass storage assigned to the file. Additional entries may be present to describe areas of mass storage which are logical but not physically adjacent to one another. Last descriptor indicated by 2<sup>29</sup>th set to (1).

### 3.1.2 Unit Control Block (UCB)

The UCB contains hardware and control information used by the I/O Director and the device handler.

Word	0	HANDLER RIR
	1	UCB LINK
	2	P-TYPE   CH/CH/UNIT
	3	CCB ADDRESS
	4	LENGTH OF UCB   # OF REQUESTS
	5	CORE MOD. SIZE   TSET LOCATION
	6	PUSH/POP QUEUE
	7	
	.	
	.	
	N	

Word 0 - contains RIR setting of device handler and is the implied starting address.

1 - contains link to other UCB's on the I/O channel.

2 - contains peripheral type and channel and unit number.

3 - address of the Channel Control Block (CCB).

4 - number of words contained in the UCB and number of I/O requests currently queued to the CCB for this unit.

5 - minimum core module obtained by the I/O Director for the handler and "Test and Set" location used by the I/O Director to lock out the unit.

6 - PUSH/POP link for queing requests to the unit.

7-N- variable control information used by the handlers (see Section 3.3 for individual handler usage).

# Sequence of an I/O request

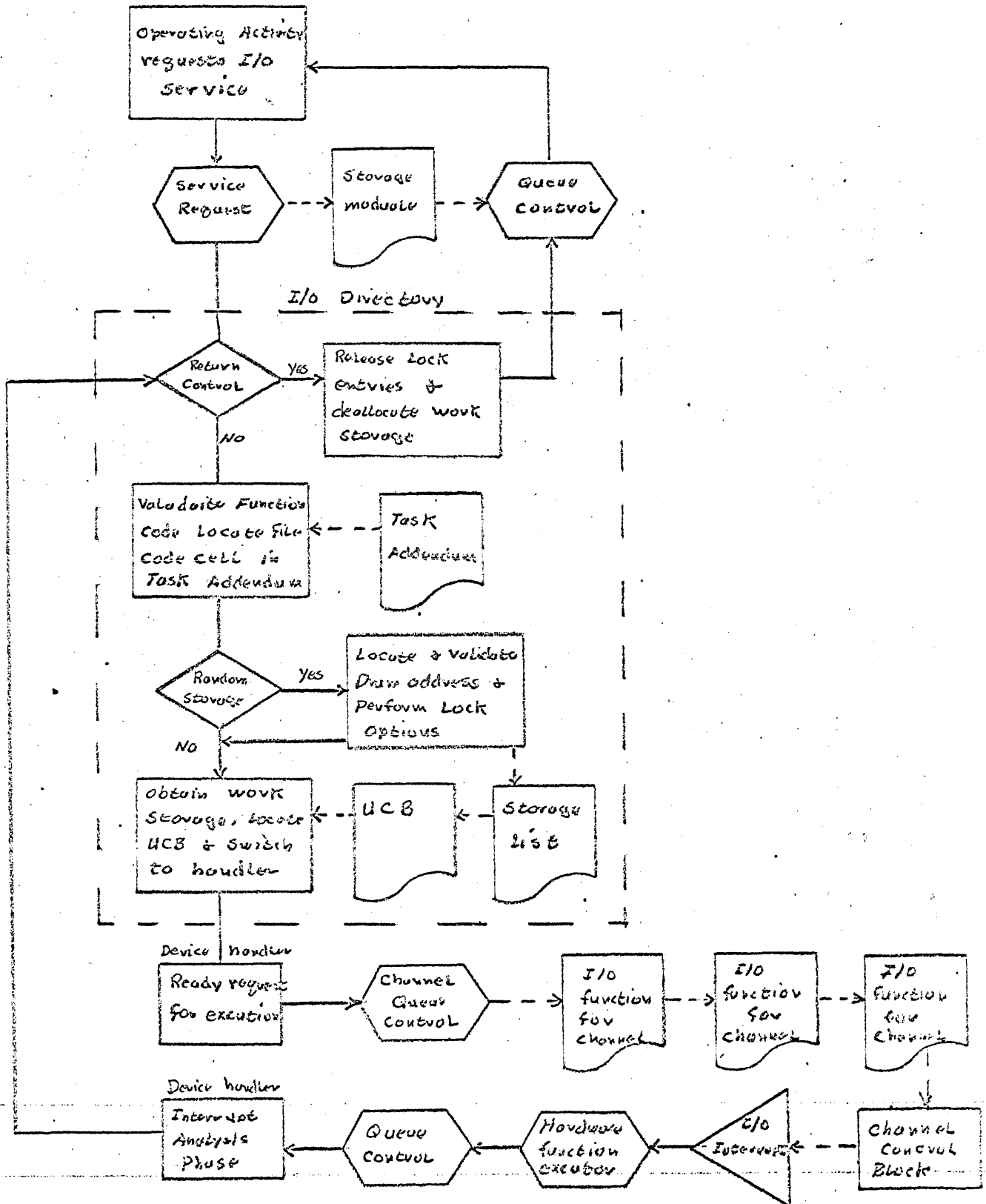


Figure 3-1

## 3.2 I/O Director

The I/O Director consists of three distinct routines: a pre-processing routine for the device handlers, a console handling routine, and a logical lock routine for mass storage.

### 3.2.1 Preprocessing Function

**Purpose:** To perform functions which are common to all handlers and coordinate and control the flow of I/O requests to the handlers.

**Entry:** One entry is from a worker program through an Exec call to the Service Request Routine. Another entry is from the Device Handler at the completion of an I/O request. Entry parameters are:

- "A" contains the Activity Addendum address

- "Q" contains the lower of IFR at interrupt time (function code). If Q is equal to zero the entry is from a handler; if Q is not equal to zero the entry is from a worker program.

- When entry is from a worker program the address (relative to PLR lower of the worker) of the associated parameter packet if required, is contained in the B7 position of the last Storage Module linked to the Activity Addendum.

- When entry is from a handler the address of the associated Unit Control Block is contained in B5.

**Exit:** One exit is to the Device Handlers for execution of an I/O request. Parameters are:

- B4 contains the address of the Storage Module associated with the request.

- B5 contains the address of the appropriate UCB.

- Within the Storage Module indicated by B4 other parameters are stored; the lower of the IFR position contains the function code; the B7 position contains the absolute address of the parameter packet; the word following the Q register position contains the address of the core area obtained for handler use.

Another exit is to the worker program at completion of the I/O request. Parameters indicating the status of the request have been placed in the

"A" and "Q" positions of the Storage Module. These values will be contained in the A and Q registers when the worker regains control.

#### Operation: Request Initiation Phase

Upon gaining control from a worker program, the I/O Director loads B4 with the address of the Storage Module contained in the Activity Addendum. This parameter is used both by the director and handlers. The function code identifying the operation to be performed is then interrogated to determine the location of the file code (may be in B7 or in the parameter packet, see Section 3.5.2).

After obtaining the file code the Task Addendum location corresponding to the file code is accessed to determine the validity of the file code. If the file code is invalid a "No Assignment" (40000000010) status code is placed in the A-register position of the S-mod and control is returned to the worker program.

A valid file code will yield the address of either a Unit Control Block or a Random Access Storage List. This value is then entered into B5 to be used by both the director and the handlers.

If the file code pointed to a Random Access List and the function code indicates rewind, rewind with interlock, write end-of-file, or erase operation, control is returned to the worker with a normal completion status code in the A-register.

If a buffer(s) is required for the operation, the buffer limits are compared to the PLR value that was in effect when the I/O request was made (PLR value in the S-mod). When the buffer is found to lie, in whole or in part, outside the PLR setting, the request is aborted and control is returned to the worker program with an "Incorrect Parameter" (4000000002) status code in the A-register.

At this point a branch is made if all buffers specified pass the above test; when the file to be referenced is found to be a random access file, the director checks the logical lock lists (Section 3.2.3) if applicable, and exits directly to the Mass Storage Handler. When the file code points to a unit oriented device further processing is required by the director.

The I/O Director then, for unit oriented devices, calculates the size of the core module required by the handler to set up the I/O commands. The size is based on the number of buffers contained in the request packet (list). If a list type packet (see Section 3.5.2) is not involved a "fixed" size is used (this "fixed" size is variable by handler; the value is held in the UCB). The size of the area to be obtained is then stored in the S-mod (upper of the B3 position).

Next a check is made to determine whether or not the unit is currently busy. This check is made using the Test and Set instruction on a location within the UCB. If the unit is busy the requesting activity is taken out of control and queued through a location in the UCB. Requests (activities) queued in this manner will be executed in a FIFO sequence when the unit becomes available.

If the "Test and Set" passes, a core module of the size determined above is obtained from the free core chain. The address of this area is then stored in the S-mod (both the beginning address and size must be retained in order to release the area subsequent to completion of the request). A direct switch to the handler indicated in the UCB is then executed.

#### Request Return Phase

After completion of the I/O requests, the handlers return control to the I/O Director. Again a distinction is made between random access and unit type devices. Upon return from a random access handler, the director checks the logical lock lists, if applicable, and returns to the worker program. However, upon return from a unit type handler the director must perform certain functions before returning control to the worker.

The director checks the unit queue to determine if any requests have been queued while the unit was busy. If not, the director releases the core obtained for the handler and returns to the worker program.

If any requests have been queued to the unit the director examines the core requirement of the next request to be executed to determine whether or not the core area of the completed request will satisfy the area required by the next request. If so, the area is transferred to the next request; if not, the area is released. The next request is then removed from the unit queue and placed on the ready queue and control returned to the worker program.



### 3.2.2 Console Handler

The Console Handler consists of two routines. One routine is a part of the I/O Director and functions to queue output messages to the console. The other routine functions to unpack and initiate output, and to accept and assemble input from the console. The two routines both access certain control locations.

#### 3.2.2.1 Console Queue Routine

##### Purpose:

The Console Queue Routine accepts message blocks, assigns Delay Numbers, if required, and queues the message block to the console handling routine. The queue routine is independent of type of console.

##### Entry:

The Console Queue Routine is entered from the Console Control Routine (see Section 6.0 ). Entry parameters include the address of the message block and the function code. The absolute address of the message block is contained in the B7 position of the allocated S-mod. The function code, contained in IFR lower, determines the type of operation.

##### Exit:

The routine exits to the Console Control Routine when the message has been queued or when the Delay has been answered.

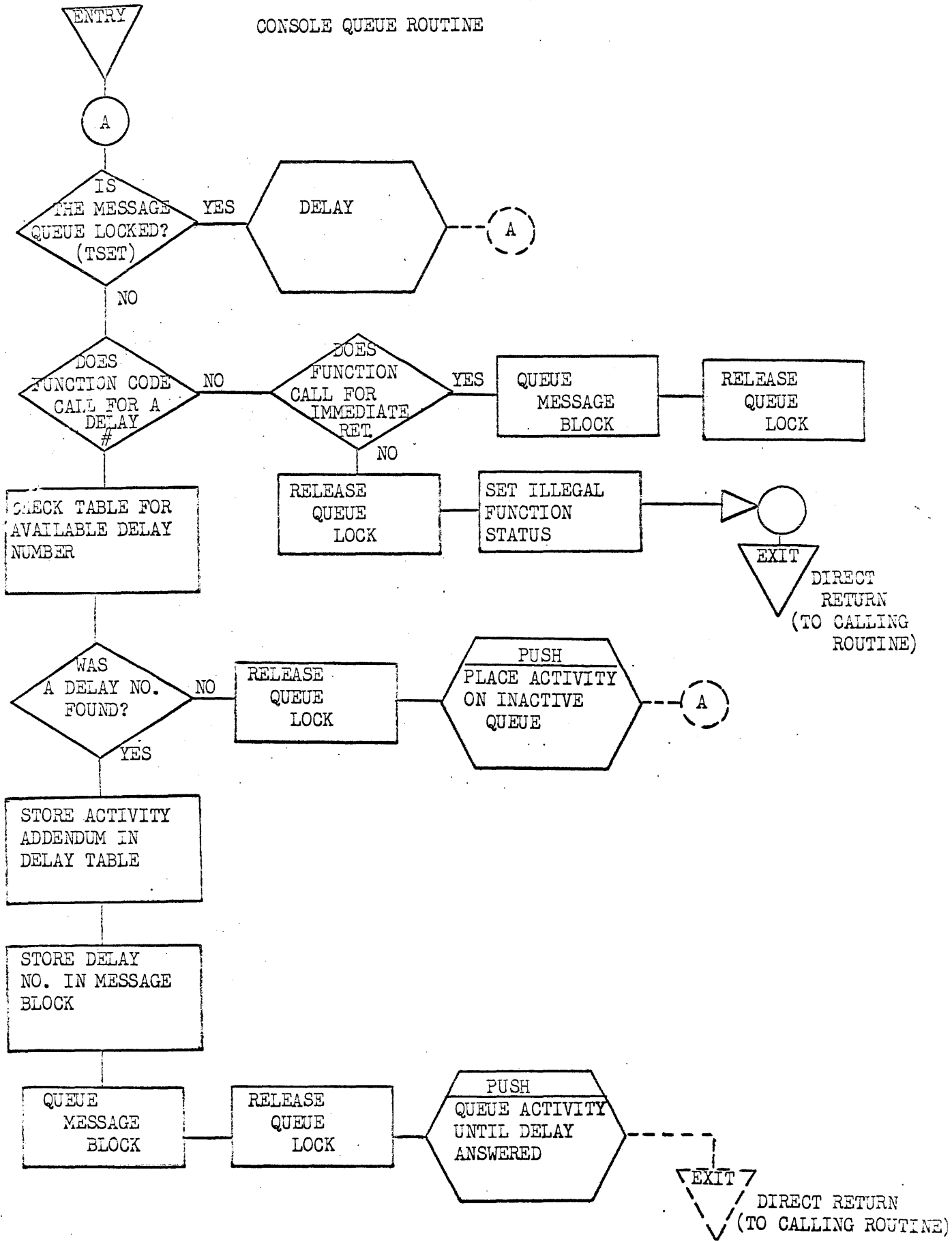
##### Operation:

The routine first tests the function code for validity; if invalid, control is returned with an "incorrect parameter" (40000002) status in the A-register. The parameter packet is shown below:

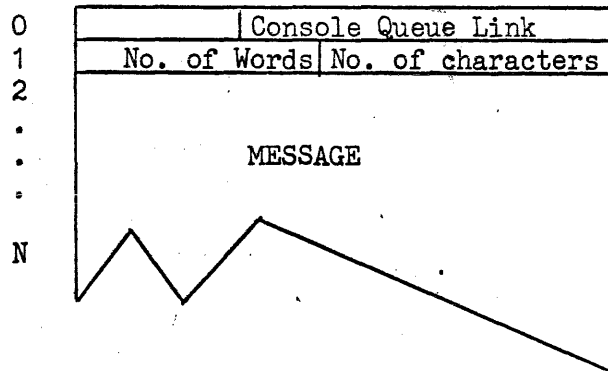
ENT*B7	V0
EXRN	V1

where: V<sub>0</sub> is the absolute address of the message  
V<sub>1</sub> is the function code:  
10101 - queue message and return control  
10102 - assign delay number and queue message. Control is returned when the delay is answered.

CONSOLE QUEUE ROUTINE



The message block indicated in B7 must have the following format:



The above message is formed by the Console Control Routine in response to a literal in the requesting program. Word 0 is a chain cell used for linking the message to the console queue. The lower of word 1 contains the number of characters in the message. The upper of word 1 contains the number of words in this message area (this value need not have any connection with the number of characters, it is used to release the message area to free core when the complete message has been unpacked). Words 2-N contain the output message.

If the function code indicates "queue and return" the message is queued to the console output queue and control is returned to the Console Control Routine at the location following the packet.

If the function code indicates that a delay number should be assigned, the routine chooses an unused number from the Delay Table shown on the following page and places this number in bits  $2^{24}$  -  $2^{29}$  of word 1 of the message block. The activity addendum address is placed in the Delay Table, and the message block is then queued to the console output queue. The activity is then placed on an inactive queue until this delay is answered.

The Delay Table is used to associate a delay number with a particular activity and to indicate those numbers currently in use.

## DELAY TABLE

Delay No.	01	ACTIVITY ADDENDUM
	02	0
	03	ACTIVITY ADDENDUM
	.	
	.	
	.	
	N	

The Delay No. is implied by position in the table. When the number is in use the associated activity addendum address appears in the corresponding table position. When the number is unassigned its table location is zero.

### 3.2.2.2 Console Handling Routine

#### Purpose

The Console Handling Routine is the interface between the Console Control Routine and the hardware. The routine unpacks and initiates output to the console, and accepts and packs input from the console.

#### Entry and Exit

Entry to the routine is from various points within OMEGA when output is inactive on the console channel. Exit is made to the point of entry.

#### Output Operation

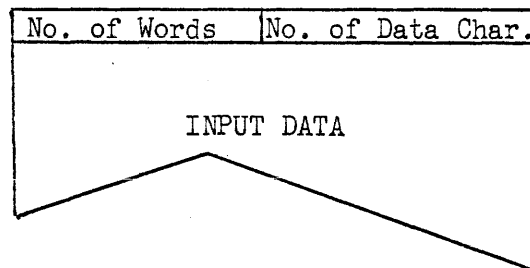
Upon receiving control the routine examines the console output queue. If the queue is empty the routine exits. If the queue is not empty the following operations are performed:

- The first request (FIFO) is removed from the queue.
- The message is unpacked and placed in the routine's output buffer.
- If a delay number is indicated the number is added to the output buffer in the form "DXX".

- The output buffer control word is set up and output initiated on the console channel.
- A "control thread" is established so that the message area may be released to the free core chain.
- Control is returned to the Console Control Routine, or, if a delay number is assigned, the activity is queued through the PUSH\$ operator.

### Input Operation

When output is not active the handler will accept input from the console. Answers to messages with delay numbers are restricted to ten characters exclusive of the delay number. The first ten characters entered will be returned in the AQ-register (the delay answer is intended primarily for positive/negative type response). All other entries will be assembled in free core and passed to the Console Control Routine for interpretation. The routine ultimately receiving the message is responsible for releasing the message area to the free core chain. The input message will have the following format when given to the Console Control Routine:



The "No. of Words" is the length of the message area to be released to the core chain.

Certain conventions and control characters have been established for console input. These are delineated below:

- An input message must be initiated by a "carriage return" (Fielddata code 04). Any character entered before the carriage return will be ignored. The carriage return will cause two line feeds and a carriage return to be sent to the console.
- Each character will be sent to the console printer as it is entered. If an input character does not appear on the console printer, it has not been recognized by the handler.

- The maximum number of characters per input message is 32 decimal.
- The Fieldata code 77 (⤴) is used as an erase code. Entering this code will cause the last character entered to be deleted from the message. Two erase codes in succession will cause the last two characters to be deleted. Three erase codes in succession will cause the entire message to be deleted.
- The Fieldata code 57 (Ⓢ) is used as a stop code; it signifies the end of an input message.
- The carriage return (04), line feed (03), erase code (77), and master space (00) if entered, will not be placed in the message buffer and will not count against the maximum message size. All other characters entered will become part of the input message.
- Characters placed in the input message will be left justified as they are entered. Unused character positions of the last word (following the "stop code") will be cleared to master spaces (00).
- The "initiate input" character (carriage return) may be entered during a console output operation. It will be recognized as soon as the output operation has been completed.

### 3.2.3 Logical Lock of Mass Storage

Logical Lock is an element which may be collected as part of the I/O Director. Logical Lock pertains only to mass storage files. If a lock function is received for a unit device, the operation will be performed as if given without lock. When Logical Lock is included in the director, mass storage may be accessed both with and without lock functions.

Read, write, and search requests may be given both with and without lock. Functions without lock are provided so that time is not spent searching the lists when the lock condition is unimportant. The following interpretations are placed on the functions:

- Read - The lock list is not checked. The read is performed regardless of whether or not the area is locked. The area read is not placed on the lock list.
- Read Lock - The lock list is checked. If any part of the area to be read is locked the request is delayed until the area has been released. When the area is not locked the read is executed and the area is placed on the lock list.
- Write - The list is not checked. The write is executed and no release is made. If the area is locked it remains locked.
- Write Release - The lock list is checked. If any part of the area being written is within a locked area, that complete entry (or linked entries) is removed from the list. When an area is to be released without actually writing into it, a Write Release may be given with the address within the locked area and the number of words equal to zero.
- Search - The list is not checked. If a find is made the area is read regardless of lock condition.
- Search Lock - The list is checked when and if a find is made. If the area is locked the request is delayed until the area has been released. When unlocked, a read is performed at the find address and the area read is placed on the lock list. If a find is not made the area is not entered on the list.

#### Operation

When the I/O Director detects a request for a mass storage file, the function code is tested to determine if it is a lock function. A "read lock" function will cause the I/O Director to search the lock list. If any part of the area to be read falls within an area contained on the list, the activity is queued until the area has been released. When the area to be

read is not found on the lock list the area is then placed on the list and control given to the mass storage handler for execution. The area will be locked regardless of the results of the read operation.

A "write release" function is given by the I/O Director directly to the mass storage handler. When the director receives control from the handler at completion of the write, the lock list is interrogated. If any part of the area written lies within a locked area, that area is removed from the list. At this time the queue of activities inactive because of a "read lock" request is checked to determine if any of these requests may now be executed. If so, the activity is removed from the inactive queue and placed on the ready queue.

### Lock Lists

A lock list is maintained for each mass storage subsystem. The lists consist of four word entries chained together. Normally there will be one entry per locked area. If, however, the area to be read covers two or more noncontiguous areas of the file, multiple entries must be made. The four word entry is shown below.

Word	0		17	QUEUE LINK
	1	BEG. ADDRESS OF LOCKED AREA		
	2	LINK	15	NO. OF WORDS
	3	JOB NO.	15	ACTIVITY I.D. (B6)

Word 0 - Chain cell linking all entries on the lock list.

Word 1 - Beginning address of locked area. The address is a subsystem logical address (physical address of drum subsystems). The lists are maintained in order of ascending address.

Word 2 - "No. of words" is the size of the locked area for this entry.

"Link" is a number assigned to identify all multiple entries which are the result of locking non-contiguous file areas. If the link is zero this is not a multiple entry.



Word 3 - "JOB No." of activity locking this area. Used to delete entries which may be present when the task terminates or is aborted.

Activity I.D. is the identity of the activity locking this area. Activity I.D. is contained in B6 when the request is made. This identity is used to delete any entries which may be on the list when the activity terminates.

### 3.3 Device Handlers

#### 3.3.0 General Description

Device Handlers are divided into two broad categories -- Unit Device Handlers and Mass Storage Device Handlers. The main distinction between the two types is the hardware address calculation required in the Mass Storage Handlers. Magnetic tape, card, printer, and 1004 handlers comprise the Unit Device Handlers. Fh-432, FH-880 Drums, and FASTRAND I are classified as the Mass Storage Handlers. All Device Handlers are interruptable and re-entrant. Each handler has an Initiation Phase and an Interrupt Phase.

##### Initiation Phase

The Device Handlers are entered in the Initiation Phase from the I/O Director. The addresses of the Storage Module and The Unit Control Block are contained in B4 and B5 respectively. When a parameter error is detected, the Initiation Phase exits to the I/O Director with the appropriate status code in the A-register. If any I/O instructions are to be executed in response to the packet, this phase exits to the Hardware Function Executor.

##### Interrupt Phase

After the Hardware Function Executor has initiated the functions or series of functions and all interrupts have occurred for a particular I/O Module, control is returned, at the activity priority, to the interrupt phase of the Device Handler. The hardware status code and the buffer control registers are stored, for external interrupts, by the Hardware Function Executor. Exit is made to the I/O Director to list the activity on the ready queue or to the Executive to call a recovery routine from the system mass storage.

##### Unit Device Handlers

In the Initiation Phase, the Unit Device Handlers will check the function code and buffer control word to determine if they are appropriate to the particular device. If the parameters are found to be correct, the Device Handler then proceeds to set up the I/O Module with the information required by the Hardware Function Executor (the format of the I/O Module is described in Section 2.2.3 ). After setting up the I/O Module, the handler scans the channel initiation queue to determine the most advantageous position for the request. Exit is made to the Hardware Function Executor with an indication of where the module should be queued.

Upon regaining control in the Interrupt Phase, the Unit Device Handlers examine the hardware status word(s) placed in the I/O Module. If the operation has been successfully completed, the status code is placed in the A-register, any supplementary information is placed in the Q-register, and exit is made to the I/O Director.

Functions that could not be performed (Servo rewinding, etc.) are reinitiated by setting up the I/O Module and again placing the module with the Hardware Function Executor. If error recovery is necessary, the appropriate recovery routine is called from mass storage by the Device Handler and operated as described in section 3.4.1.

#### Mass Storage Handlers

In addition to the above procedures the mass storage handlers must calculate the physical address of the request from the logical file address contained in the packet. The logical file area referenced may cover non-contiguous areas of mass storage unit(s); in this case the handler must generate a separate function and buffer for each physical area.

### 3.3.1 Magnetic Tape Handler

- Purpose:** To transform a magnetic tape I/O request into the designated tape operation, analyze the external interrupt(s) which occur from this operation and notify the I/O Director of the results.
- Entrance:** Entrance to the Magnetic Tape Handler is made through the I/O Director. Parameters are conveyed through B registers and the Storage Module set up at I/O request time. Parameters upon entry are:
- B4 - Address of the Storage Module set up at I/O request time.
  - B5 - Address of the Unit Control Block of the magnetic tape unit marked for an I/O operation.
  - Word 16g of the Storage Module - Address of free core to be used by the Magnetic Tape Handler when setting up command words for executing the designated I/O.
- Exit:** Exits are made by the Magnetic Tape Handler to the Formatter for execution of the designated I/O, and the I/O Director to report the results of the executed I/O.
- To Formatter:** Parameters are conveyed in the A, Q and B7 registers as specified by the Formatter.
- To I/O Director:** Parameters are placed in the Storage Module positions as follows:
- B7 - Contains the octal number of the last data block correctly passed over, backward or forward, by the Magnetic Tape Handler if numbered data blocks have been requested by the user.
  - A - The code designating the status of the I/O operation performed. See chart and notes on pages 6 and 7.
  - Q - The number of words correctly read or written in the I/O operation. See charts and notes on pages 6 and 7.

### 3.3.1.1. Description

All magnetic tape I/O requests will be serviced by one Magnetic Tape Handler. Functions applicable to magnetic tape will be serviced identically on all subsystems with the exception of READB on Uniservo 3C/4C, which is serviced as a backspace block.

If designated through the ASSIGN statement, an octal number will be placed on each data block written numbering sequentially with load point designated as zero. The number of the last data block successfully passed over during the I/O operation is returned to the user in the B7 register. An out of sequence status appears in the A register as  $\text{00011*000XX}$  (XX = I/O status). The current block count for each unit is maintained by the Magnetic Tape Handler in the appropriate Unit Control Block.

After executing an I/O packet, the Magnetic Tape Handler notifies the I/O Director of the success or failure of the operation. No repositioning or error recovery is attempted by the Magnetic Tape Handler, but is accomplished by a separate Error Recovery Routine (see Section 3.4.1).

### 3.3.1.2. Operational Mode

All information influencing the way magnetic tapes are to be written or read is stored or maintained in the Unit Control Block for that particular unit. This information is put there through the use of the ASSIGN statement and maintained by the Magnetic Tape Handler when necessary. Two words, KUCB1 and KUCB2, words 7<sub>8</sub> and 10<sub>8</sub> respectively are used to hold this information.

#### KUCB1

A bit is set in this word to designate numbered data blocks to be written, the "Noise Constant" to be used on this particular unit when applicable is stored here and the initialization constant specifying density, parity, etc., to use when writing and reading on this particular unit, is contained in this word.

#### Numbered Data Blocks

Bit  $2^{15}=1$  of KUCB1 designates each data block to be written with an octal number making up the first word of the block. These data blocks will be numbered sequentially with load point designated as zero.

This number is written with a GWRITE operation and read with an SREAD operation from the Magnetic Tape Handler, hence, the number will not be reflected in the data used. The number of the last block correctly passed over, reading or writing, backward or forward, is returned to the user in the B7 register. An out-of-

sequence status is indicated with an 11 in A register upper. REWIND or REWINDI result in the B7 register being returned as zero.

#### Noise Constant

Bits  $2^0 - 2^1$  are used to hold the "Noise Constant" for a specific unit when applicable (Uniservo 3C, 4C, 6C, 8C). All blocks read or moved over causing an error status, the length of which is less than or equal to the "Noise Constant" will be considered as foreign material on tape and disregarded.

#### Initialization Constant

This constant specifies the density, parity, etc., which should be used when reading or writing tapes on this particular unit. This constant is only applicable to Uniservo 3C, 4C, 6C, 8C subsystems. Values of this constant are listed below.

If numbered data blocks and a "Noise Constant" are specified, they will be included within the constant.

Binary 200 ppi	03000*10000
Binary Coded Decimal 200 ppi	03200*50000
Binary Coded Decimal Translate 200 ppi	03210*54000
Binary 556 ppi	02000*20000
Binary Coded Decimal 556 ppi	02200*60000
Binary Coded Decimal Translate 556 ppi	02210*64000
Binary 800 ppi	02020*30000
Binary Coded Decimal 800 ppi	02220*70000
Binary Coded Decimal Translate 800 ppi	02230*74000
Binary 800 ppi 9 Channel (Uniservo 6C, 8C only)	00000*32000

#### KUCB2

The upper half of this word is used to maintain the octal number of the block of tape just passed over on this particular unit when numbered data blocks are designated. This number is maintained by the Magnetic Tape Handler.

### 3.3.1.3. Function Execution

#### Applicable Functions

The resulting operation of I/O packets marked for execution by the Magnetic Tape Handler is described below.

- READ - One block of data, not to exceed the word count, will be read in the forward direction into the area designated by the buffer base. A word count of zero will result in a move forward, without data transfer, of one block.
- WRITE - One block of data, designated by the word count and buffer base will be written on tape. A word count of zero will result in the return of an illegal parameter status.
- READB - One block of data, not to exceed the word count, will be read in the backward direction into the area designated by the buffer base. A word count of zero will result in a move backward, without data transfer, of one block.
- WRTEOF - The appropriate hardware end-of-file mark will be recorded on tape.
- REWIND - Tape will be positioned at load point.
- REWINDI - Tape will be positioned at load point with the designated unit in an interlock condition.
- ERASE - A fixed area of tape will be erased.
- SREAD - One block of data will be scatter-read into the areas dictated by the word counts and buffer bases designated in the LIST packet. Word counts of zero will be ignored.
- GWRITE - One block of data will be gather-written into the areas dictated by the word counts and buffer bases designated in the LIST packet. Word counts of zero will be ignored.
- MREAD - Sequential blocks of data, dictated by the word counts and buffer bases of the LIST packet, will be read or passed over. Word counts of zero will result in blocks of data being moved over without data transfer.

## Method of Execution

Upon entry, the Magnetic Tape Handler retrieves the function code and I/O packet address placed in the Storage Module. Through examination of the I/O packet the function code, it can be determined how many command words, buffer control words, and buffers are needed to execute the designated I/O. These parameters are then formed in the free core assigned, the address of which is contained in word 16<sub>3</sub> of the Storage Module.

After setting up the free core with parameters for I/O execution, an examination is then made of the Unit Control Block to determine which function is needed for the I/O operation. This function is dictated by the type of subsystem the unit is on. After this function is formed, it is placed in the free core and an Exec Return is made transferring control to the Formatter for I/O execution.

Upon re-entry, the I/O has been executed and external interrupt(s), have been stored. After examination of these interrupt(s), the Magnetic Tape Handler determines whether the I/O has been successfully completed. If so, the appropriate status is placed in the A register position, the number of words transferred, if any, in the Q register position and the number of the last data block passed over in the B7 position, if applicable, of the Storage Module. If errors have occurred during I/O execution, the Magnetic Tape Error Recovery routine is called for completion of the I/O. Control is then returned to the I/O Director through an Exec Return.



FUNCTION	(See notes for AX, QY explanation)					
	STATUS	NORMAL	FRAME COUNT	INCORRECT PARAMETER	*UNRECOVERABLE ERROR	END-OF-FILE
READ	A0,Q1	A1,Q1	A2,Q2	A3,Q0	A4,Q0	
WRITE	A0,Q1		A2,Q2	A3,Q0		A5,Q1
READB	A0,Q1	A1,Q1	A2,Q2	A3,Q0	A4,Q0	A5,Q1
WRTEOF	A0,Q0		A2,Q2	A3,Q0		A5,Q0
REWIND	A0,Q0		A2,Q2	A3,Q0		
REWINDI	A0,Q0		A2,Q2	A3,Q0		
ERASE	A0,Q0		A2,Q2	A3,Q0		A5,Q0
SREAD	A0,Q1	A1,Q1	A2,Q2	A3,Q0	A4,Q0	
GWRITE	A0,Q1		A2,Q2	A3,Q0		A5,Q1
MREAD	A0,Q1	A1,Q1	A2,Q2	A3,Q1	A4,Q1	

A Register value of the Storage Module

A0 - 0000\*0000

A1 - 0000\*000X  
X = Magnitude of Frame Count

A2 - 4000\*0002

A3 - 4000\*0003

A4 - 0000\*0004

A5 - 0000\*0005

Q Register value of the Storage Module

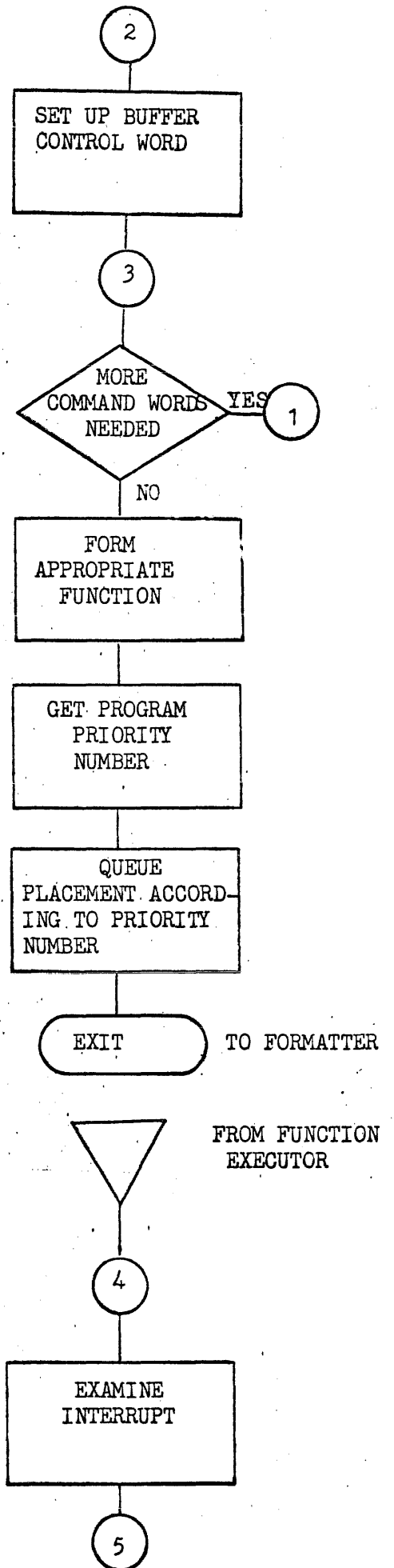
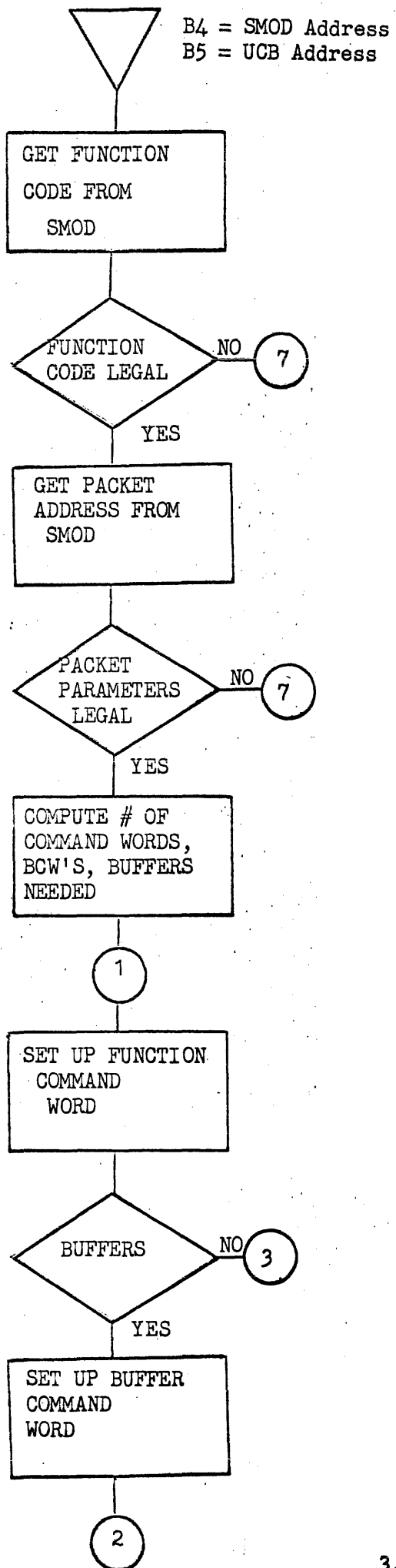
Q0 - 0000\*0000

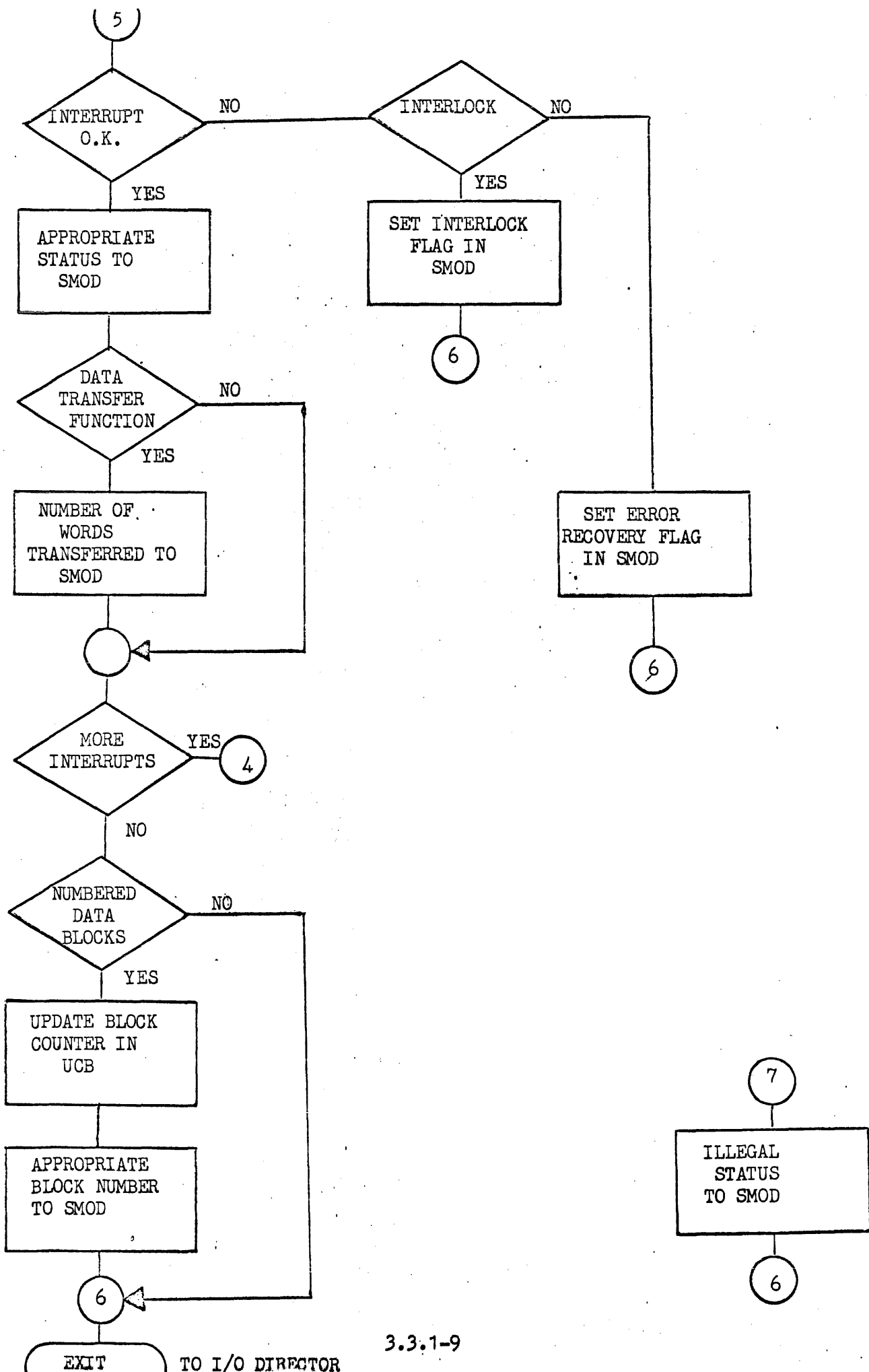
Q1 - 0000\*XXXXX  
XXXXX = number of words transferred

Q2 - YYYYY\*YYYYY

Y... = meaningless information

- \* Tape is positioned beyond the block in error
- \*\* The last block of the series read having an abnormal Frame Count will be indicated.
- \*\*\* Tape will be positioned beyond the last block indicated to be read.  
The number of words successfully read of the blocks indicated will be contained in the Q Register.
- \*\*\*\* Tape will be positioned beyond the End-of-File marker. The number of words read before the End-of-File marker was encountered will be indicated in the Q Register.





### 3.3.2 High Speed Printer Handler

**PURPOSE:** To verify the packet information and use it to form High Speed Printer Command words. After the Commands have been executed the Handler will analyze the interrupts and notify the I/O Director and/or the User of these results.

**ENTRANCE:** B4 = Address of Storage Module  
B5 = Address of Unit Control Block  
B7 of SMOD = Address of Packet  
SMOD+17 = Address of Core Module

**EXIT TO FORMATTER:** As specified by the FORMATTER

**EXIT TO I/O DIRECTOR:**

A register of SMOD	=	0000000000	-	Successful Completion
" " " "	=	4000000001	-	Inappropriate Function
" " " "	=	4000000003	-	Unrecognizable Interrupt Status.

The Unrecognized Interrupt Status will be in the Q register of SMOD.

This handler will control printing and spacing commands on both the 0751 printer and the new 1100-1600 line per minute printer.

Form control is achieved by specifying the top and bottom margins and the number of printable lines per page (the sum of these three quantities must be equal to the physical page length). This specification is made in the ASSIGN statement and the values are placed in the Unit Control Block of the printer. Printing will not occur within the margins -- whenever the handler senses that, with the spacing given in the packet, the print line will fall within either the top or bottom margin, the paper is spaced to the next printable line and the print executed there. Spacing within the printable area, however, is the responsibility of the activity. The physical line position of the printer is maintained by the handler in the Unit Control Block. This line counter is cleared whenever a new printer ASSIGN statement is processed. At this time, it is assumed that the paper is positioned at the first physical line (at the fold). At the completion of each print job a "Home Paper" command should be given in order to position the paper at the first physical line in anticipation of a new ASSIGN statement.

The WRITE function is the only function recognized by the printer handler; all others will be returned as inappropriate. The "Number of Words" in the printer packet is used as the number of lines to be spaced -- the handler will output a twenty-seven word buffer (one print line) beginning with the buffer base specified in the packet. One 132 character line

will be printed for each request (the line may be less than 132 characters if the "77" stop code is detected within the buffer). The number of lines to space before printing is contained in the packet. If the number of lines to space is equal to zero, printing will occur on the same line as that of the previous request. When the "number of words" in the printer packet is found to be 40000<sub>8</sub> ( $2^{14}$  set) a skip to the first physical line of the next page will be performed, "Home Paper" - no printing will occur. A print request received when the paper is positioned at the first physical line will be performed at the first logical line (the physical line minus the top margin) and packet spacing will be ignored.

The handler will place each request received at the end of the channel queue.

No form control may be specified by listing the number of printable lines with the value of zero in the ASSIGN statement. If this is done, all subsequent spacing and printing will be performed as directed by the WRITE orders. The top and bottom margins will then be the responsibility of the user.

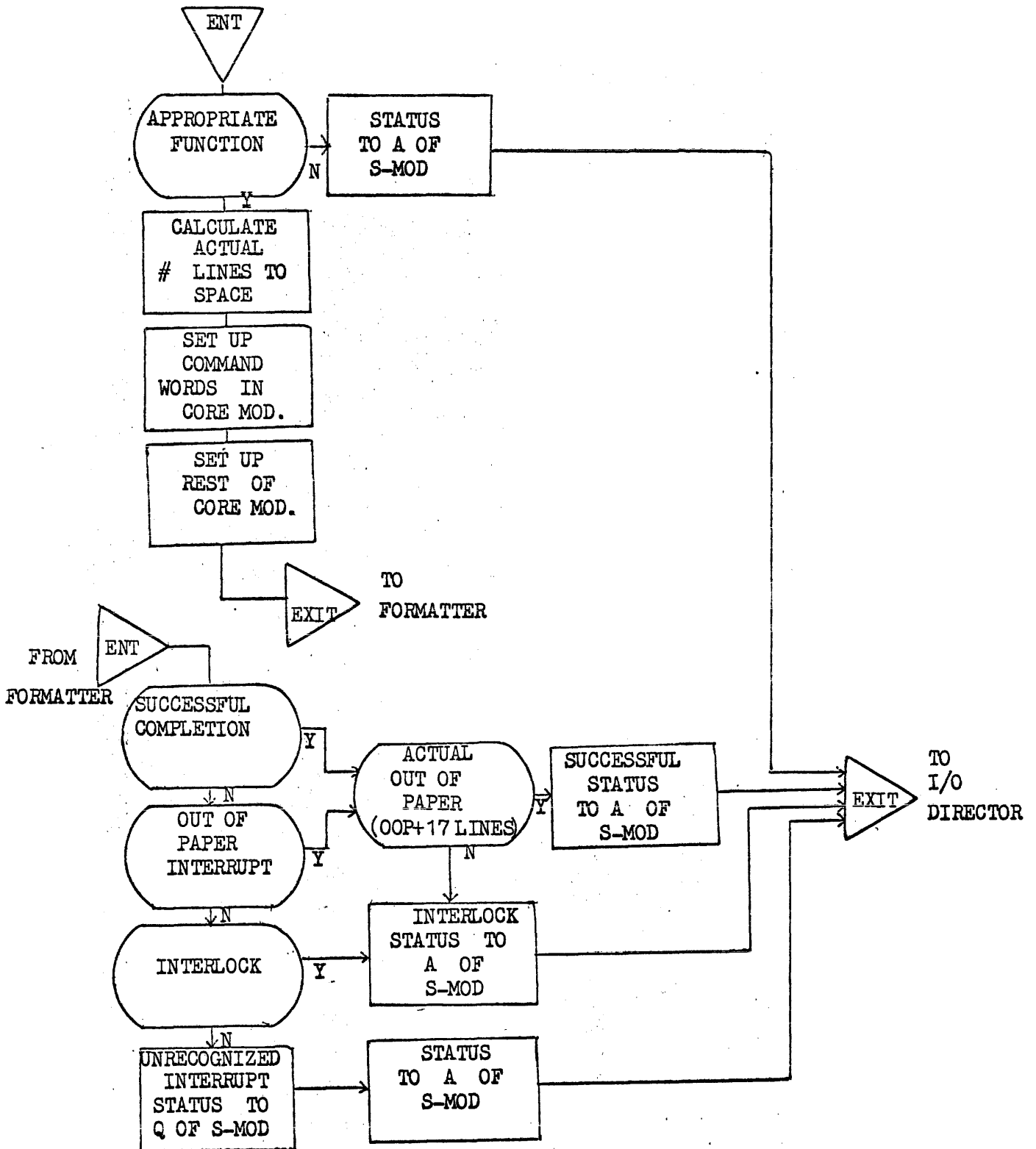
When the "Out of Paper" indication is detected, the handler if under form control will allow printing to the bottom of the page, then an interlock condition indication will be returned. The new forms should be physically positioned at the "Home Paper" position (first physical line). Thus when control is returned after the interlock (out of paper) has been satisfied, the paper will be spaced to the first printable line and the line will be printed.

If "Out of Paper" occurs and form control had not been specified, the printing will continue until it is possible that a line had been printed below the paper. An interlock condition will then be indicated. The new forms should be mounted with the fold in the same physical position as the previous. Answering the interlock will cause the same line to be reprinted on the new form without spacing since the spacing was performed earlier.

UNIT CONTROL BLOCK

0	HANDLER RIR		
1	UCB LINK		
2	P-TYPE	CH/CH/UNIT	
3		LINK TO CCB	
4	LENGTH OF UCB	# OF REG.	KUCBNO
5	MIN C.M. SIZE	TEST & SET LOG	KUCBCM
6		UNIT INACTIVE CELL	KUCBQ
7	UPPER MAR.	LOWER MAR.	KUCB1
10	LINE POS.	PRINTABLE LINES/PAGE	KUCB2
11		OUT OF PAPER	KUCB3

The processing of the ASSIGN statement will place the upper margin, lower margin and the number of printable lines per page in the proper positions of the Unit Control Block. The Line Position and Out of Paper locations will be cleared to zeroes.





CORE MODULE

0	For Handler Use
1	Interrupt Status Store Address
2	Terminate Command
3	XF Command
4	Buffer Command
5	XF Command
6	Buffer Command
7	Fastfeed Buffer Control Word
10	XF Control Word
11	Buffer Control Word
12	Store for Interrupt Status
13	Store for Input Buffer
14	Store for Output Buffer
15	Store for Interrupt Status
16	Store for Input Buffer
17	Store for Output Buffer

HSP CHARACTER CODES  
(63-character set)

Octal. Sequence

00	@	40	)
01	[	41	-
02	]	42	+
03	#	43	<
04	Δ	44	=
05	Space	45	>
06	A	46	&
07	B	47	\$
10	C	50	*
11	D	51	(
12	E	52	%
13	F	53	:
14	G	54	?
15	H	55	!
16	I	56	,
17	J	57	\
20	K	60	0
21	L	61	1
22	M	62	2
23	N	63	3
24	O	64	4
25	P	65	5
26	Q	66	6
27	R	67	7
30	S	70	8
31	T	71	9
32	U	72	'
33	V	73	;
34	W	74	/
35	X	75	.
36	Y	76	□
37	Z	77	≠

When selection is made for the 62-character set, code 77 becomes a stop code; as such it is non-printable.

**Note:** In order to accomplish the proper spacing capabilities provided by the printer subroutine, the 62-character set must be selected. Selection is controlled by a manually operated switch located on the printer control unit.

### 3.3.3 Card Reader Handler

**PURPOSE:** To verify the packet information and use it to form Card Read command words. After the Commands have been executed the Handler will analyze the interrupts and notify the I/O Director and/or the User of these results.

**ENTRANCE:** B4 = Address of Storage Module  
B5 = Address of Unit Control Block  
B7 of S-MOD = Address of Packet  
S-MOD+17 = Address of Core Module

**EXIT TO FORMATTER:** As specified by the FORMATTER

**EXIT TO I/O DIRECTOR:**

A register of S-MOD	=	0000000000	- Successful Completion
" "	" "	4000000001	- Inappropriate Function
" "	" "	4000000002	- Buffer too small
" "	" "	4000000003	- Unrecognizable Interrupt Status. The Unrecognized Interrupt Status will be in the Q register of S-MOD.

The card reader handler will recognize the READ and MREAD (function codes 01 and 26) functions; all others are inappropriate.

The READ function will transfer the data from one card to the computer.

The MREAD function will transfer multiple card images into the core buffers as specified by the LIST operator. Each buffer must be large enough to contain the data from one card.

The mode of a card file is specified by the ASSIGN statement. There are two possible modes: TRANSLATE and BINARY. When the assign statement is processed, the appropriate mode will be set and the card memory will be cleared. Thus if the card read mode is changed in the middle of a deck, three blank cards should be inserted at this point to prevent the loss of card data.

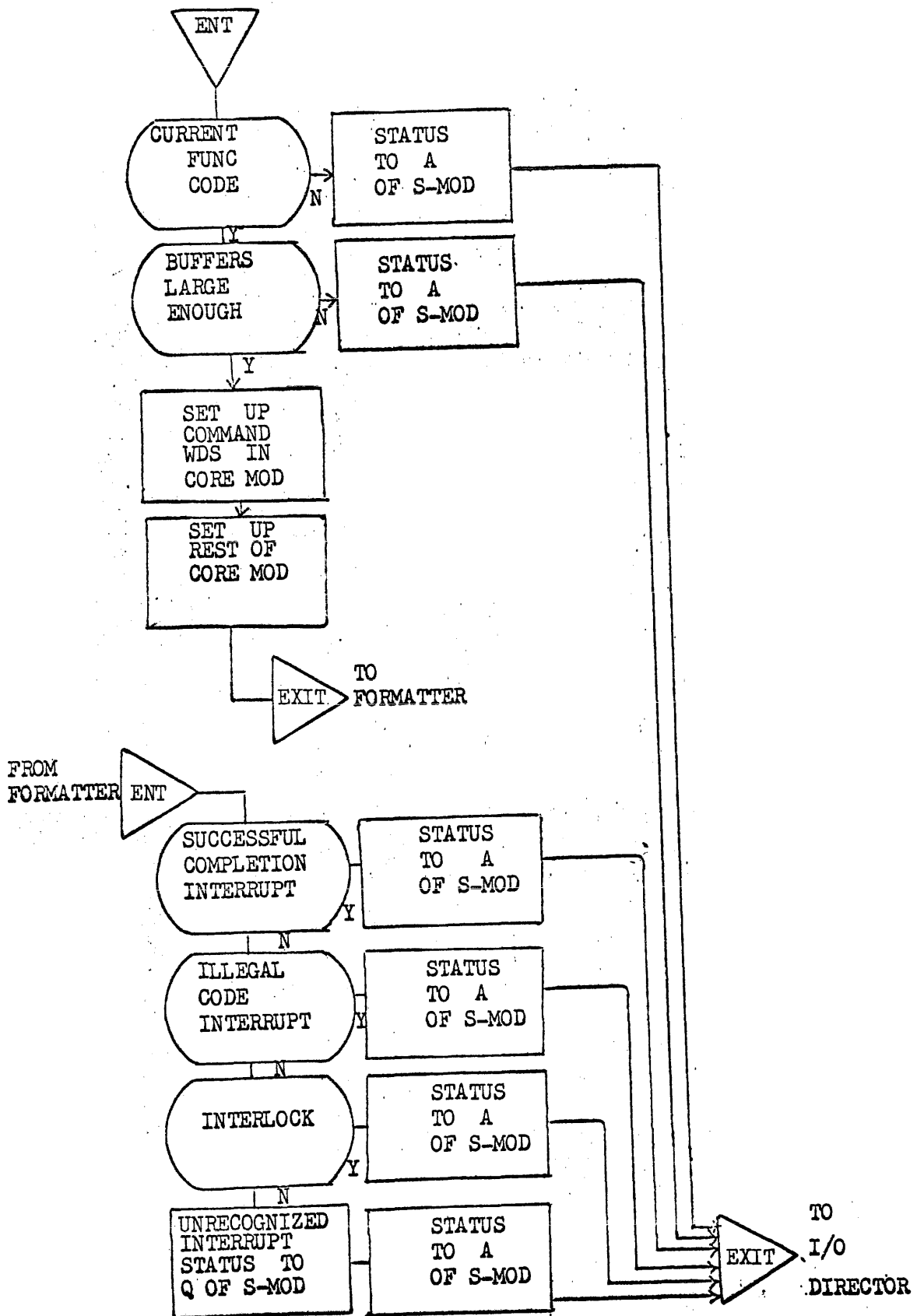
The TRANSLATE mode will cause each subsequent card that is fed to be translated from Hollerith code to a six-bit "internal" code. This "internal" code is defined by a conversion matrix within the card reader. Normally, Fieldata code is used. In this mode, one card occupies 16 computer words. The buffer specified by the read packet must be greater than or equal to 16 words, less than 16 words is an illegal parameter.

The BINARY mode will cause each subsequent card that is fed to be read in the column binary mode. Two and one-half card columns represent one 30-bit computer word. A minimum buffer of 32 words are required, less than this is an illegal parameter.

UNIT CONTROL BLOCK

0	HANDLER	RIR	
1	UCB LINK		
2	P-TYPE	CH/CH/UNIT	
3		LINK TO CGB	
4	LENGTH OF UCB	# OF REQ	KUCBNO
5	MIN C.M. SIZE	TEST & SET LOC.	KUCBCM
6		UNIT INACTIVE Q CELL	KUCBQ
7		CARD MODE 020 OR 040	KUCB1

The processing of the ASSIGN statement will place the mode indicator (020 or 040) in Word 7 of the UCB. The card memory will also be cleared at this time.



CORE MODULE

READ

0	2
1	
2	
3	
4	
5	
6	
7	

FOR HANDLER USE  
 Interrupt Status Store Adr.  
 BUF CMD WD  
 XF CMD WD  
 XF  
 BUFFER CONTROL WD  
 Interrupt Status Store  
 Buffer Status Store

M READ

0	
1	
2	
3	
7	
10	
11	
12	
13	
14	
15	
16	
17	
20	
21	
22	
23	
24	
25	

FOR HANDLER USE  
 Interrupt Status Store Adr.  
 1st BUF UMD WD  
 1st XF CMD WD  
 2nd BUF MCD WD  
 2nd XF CMD WD  
 3rd BUF CMD WD  
 3rd XF CMD WD  
 XF  
 1st BUF CW  
 2nd BUF CW  
 3rd BUF CW  
 1st Int. Status Store  
 1st BUFIN Status Store  
 1st BUFOUT Status Store  
 2nd INT Status Store  
 2nd BUFIN Status Store  
 2nd BUFOUT Status Store  
 3rd INT STATUS STORE  
 3rd BUFIN STATUS STORE  
 3rd BUFOUT STATUS STORE

$$X = 3Y + 4$$

X - Number Locations Required  
 for Core Module

Y - Number Wds in Packet

### 3.3.4 Card Punch Handler

**PURPOSE:** To verify the packet information and use it to form Card Punch Command words. After the Commands have been executed the Handler will analyze the interrupts and notify the I/O Director and/or the User of these results.

**ENTRANCE:** B4 = Address of Storage Module  
B5 = Address of Unit Control Block  
B7 of S-MOD = Address of Packet  
S-MOD + 17 = Address of Core Module

**EXIT TO FORMATTER:** As specified by the FORMATTER

**EXIT TO I/O DIRECTOR:**

A register of S-MOD =	000000000	- Successful Completion
" " " "	= 400000001	- Inappropriate Function
" " " "	= 400000002	- Buffer too small
" " " "	= 400000003	- Unrecognizable Interrupt Status. The Unrecognizable Interrupt Status will be in the Q register of S-MOD.

The card punch handler will recognize the WRITE function (01): all others are inappropriate. The WRITE function will transfer data from the computer to the card punch unit and cause it to be punched into a card.

The mode of a card file is specified by the ASSIGN statement. There are two possible modes: TRANSLATE and BINARY. The desired mode will be set as the ASSIGN statement is processed.

The TRANSLATE mode will cause each subsequent card that is punched to be in the Hollerith code. In this mode, one card occupies 16 computer words. The buffer specified by the write packet must be greater than or equal to 16 words: less than 16 words is an illegal parameter.

The BINARY mode will cause each subsequent card that is punched to be punched in the column binary mode. Two and one-half card columns represent one 30-bit computer word. A minimum buffer of 32 words are required; less than this is an illegal parameter.

UNIT CONTROL BLOCK

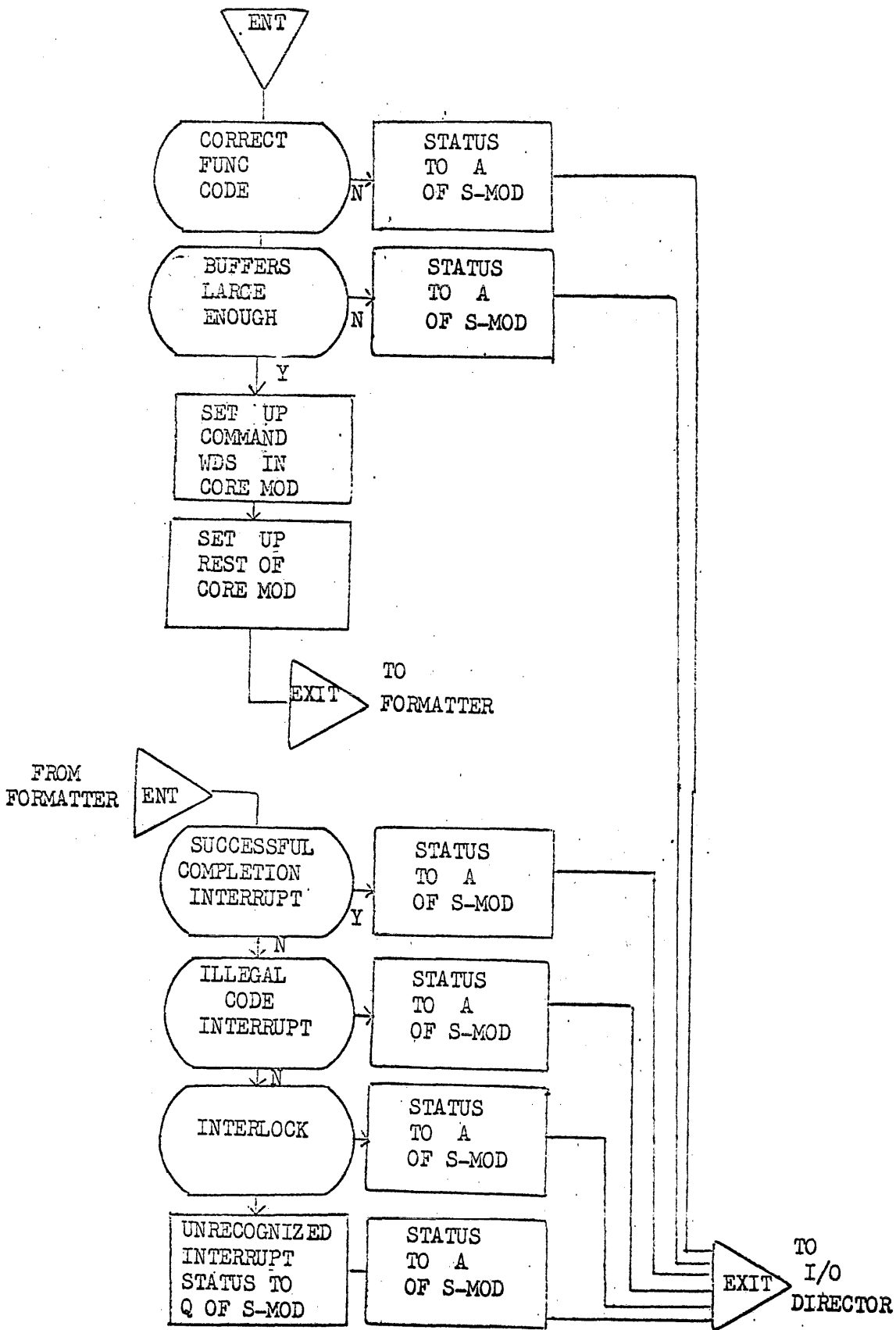
0	HANDLER RIR		
1	UCB LINK		
2	P-TYPE	CH CH/UNIT	
3		LINK TO CCB	
4	LENGTH OF UCB	# OF REQ	KUCBNO
5	MIN C.M. SIZE	TEST & SET LOC	KUCBCM
6		UNIT INACTIVE Q CELL	KUCBQ
7		CARD MODE 020 OR 040	KUCB1

The processing of the ASSIGN statement will place the mode indicator (020 or 040) in word 7 of the UCB.

CORE MODULE

0	HANDLER USE
1	INTERRUPT STATUS STORE ADDRESS
2	XF COMMAND WD
3	BUFFER COMMAND WD
4	BUFFER C.W. WORD
5	XF CONTROL WD
6	INTERRUPT STATUS STR
7	BUF. IN STAT STR
10	BUF. OUT STAT STR





### 3.3.5 Omega Mass Storage Handler

**PURPOSE:** To interface the mass storage subsystem(s) with the Omega executive and programs running in its environment.

**ENTRANCE:** EXEC Return 77540 100FC  
B4 = Address of Storage Module  
B5 = Address of Random Access Storage List  
B7 = Address of Packet

#### FC $\phi$ 1 READ

Transfer to core the data found on mass storage as defined by the manipulation of the Logical Address given in the Random Access Storage List (RASL), the Logical Increment (LI) as given in the packet, and the logical to hardware address conversion parameters given in the Unit Control Block (UCB). The number of words to be read is given in the lower 15 bits of the packet, and is limited by the length of file (RASL) and the LI. The buffer base is defined by the sum of lower PLR register and the second packet word.

Upon return of control report the status of the function in the A register (A) and the number of words read in the Q register (Q).

#### FC $\phi$ 2 WRITE

Transfer data from core to mass storage. (See FC  $\phi$ 1)

#### FC $\phi$ 3 BLOCKR

(See FC  $\phi$ 1) If an End-of-Block (EOB) external interrupt (EI =  $\phi$ 4) is received report the contents of the lower 24 bits of the Overflow Word\*\* in the upper 24 bits of the A register at return of control. The number of words read is reported in Q.

\*This paper is specifically generated for the FH880 drum handler. When the details of the Fastrand - FH880 handler are defined, this paper will be corrected to show changes of philosophy as related to both subsystems.

\*\*An Overflow word is the contents of a drum storage location following an End-of-Block sentinel (a location containing the octal word 7777777777). Bits 2<sup>29</sup> through 2<sup>24</sup> of the Overflow Word must be zeros. The contents of the Overflow Word is logically added to 04000 00000 and reported as an external interrupt.

## FC 04 BLOCKS

Search for the FH880 address containing the data word given in the fourth word of the packet. Transfer no data. Report the address of this mass storage location in terms of a logical increment to the first address of the file (Q). If an EOB sentinel is detected before a find is made, report an Unsuccessful Search status (06), and the contents of the Overflow word in A.

## FC 24 SREAD

(See FC 01) Transfer consecutive data locations from a single mass storage file to one or more core areas.

## FC 25 GWRITE

(See FC 02) Transfer data from one or more core areas to one continuous area of mass storage file.

## FC 26 MREAD

(See FC 01) Transfer one or more data areas of a single mass storage file to one or more core areas.

## Exit

The status codes reported upon exit are:

000000000 - Normal completion

XXXXXXXXX0 - This is the normal completion response to the BLOCKR function. The upper 24 bits of A contain the lower 24 bits of the Overflow Word.

4000000001 - Inappropriate function - the function code is not applicable to the subsystem. (SEARCHT issued to an FH880 subsystem, etc.)

4000000002 - Incorrect parameter - This status indicates that the data transfer is outside of the program lock limits. Under this condition this status code is generated by the I/O Director.

This status code is generated by the handler if:

- The Logical Increment specified in the packet is negative.
- An MREAD packet does not contain a Logical Increment (Bit  $2^{29}$  of the first word of the LIST = 1)

- The peripheral type code as given in the UBC 4.
- A negative hardware address is generated. This condition will occur if the LA as given in the RASL is less than the first LA of the unit.
- The packet and file or so constructed as to cause the generation of 33 or more consecutive buffer control words.

4000000003 - Unrecoverable Error - This code indicates that the function could not be completed without error interrupt. The handler will issue the function a total of four times before returning this status.

4000000004 - End-of-File - This code indicates that the function requested a transfer of data from (or to) an area of mass storage outside of the file. Reading or writing in the last address of a file will not cause the generation of an  $\emptyset 4$  status. An  $\emptyset 4$  response to a write function indicates that no data was transferred.

~~XXXXXXXX~~06 - Unsuccessful Search - At the handler level, this response can only be received in response to a BLOCKS function. It indicates that an EOB interrupt was received before a find was made. The lower 24 bits of the Overflow Word are contained in A.

The following table illustrates all legal functions at the handler level, and the possible responses to these functions. The presence of a number at the intersection of a function code and status code indicate that the particular status may be received for the prescribed function. The number at the intersection is applicable to the notes which follow. The conditions which caused the generation of a particular status have been defined.

Function Code	Status (A/Q) (See Notes)						
	∅∅ Normal	XXXXXXXX∅∅ Normal	∅1 Inappropriate Function	∅2 Incorrect Parameter	∅3 Unrecoverable Error	∅4 End-of-File	XXXXXXXX∅6 Unsuccessful Search
∅1 READ	1		2	2	2	1	
∅2 WRITE	1		2	2	2	3	
∅3 BLOCKR	4	1	2	2	2	4	
07 BLOCKS	5		2	2	2	2	2
24 SREAD	1		2	2	2	1	
25 GWRITE	1		2	2	2	3	
26 MREAD	1		2	2	2	1	

Key to Q register values

1. Q = the number of data words written or read.
2. Q = unknown
3. See #1 above. No data will be transferred, and Q = ∅.
4. See #1 above. An end of block sentinel was not read.
5. Q = the logical increment as related to the base address of the file.

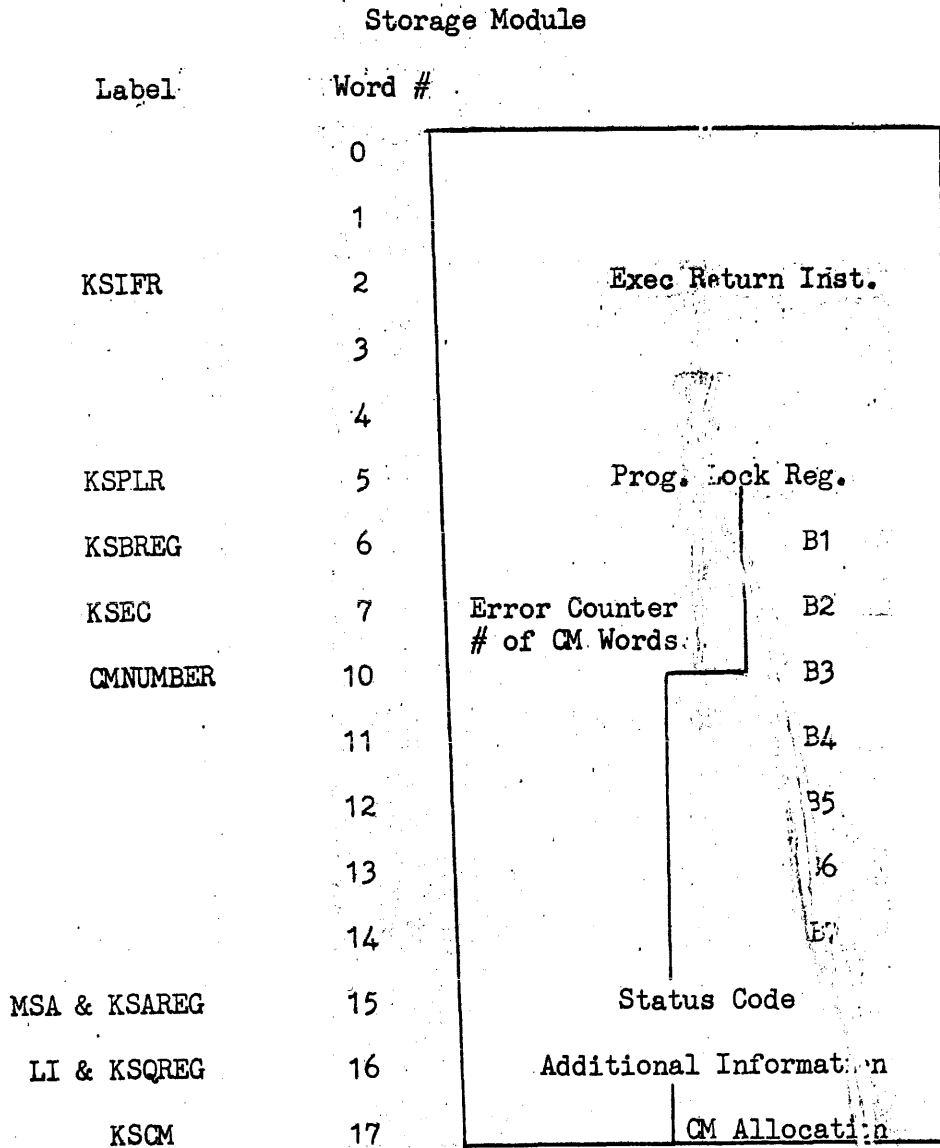
The handler will allow a program to reference from 1 to 32768 words of mass storage whether the subsystem be an FH880 or Fastrand. Packets which contain more than one set of parameters (SREAD, GWRITE, \$MREAD) may specify 32K data transfers in each parameter set. When communicating with a Fastrand subsystem it is very desirable to limit I/O buffers and logical increments to multiples of 33 words. This rule is particularly important on WRITE functions, for the handler must insure the integrity of the entire sector.

The handler will allow a program to read an entire file with one packet submission, providing the above restriction is not violated, regardless of the number of times that the file is split. A split file is a storage area which is contained in more than one continuous mass storage area.

The handler will maintain a table indicating the angular position of each drum unit under its control. Via this table it will attempt to queue each Command Word List in such a manner as to minimize latency. From this table, and the hardware function generated, the handler will compute the Start-Stop\* time of each Command Word List and cause it to be queued for execution at the earliest possible time providing its execution does not usurp a time interval allocated to a previously queued function. The RASL will aid the algorithm by unique entry for each subsystem unit containing parts of a file. The handler will account for files contained on two cylinders of a Fastrand unit.

\*Start time is that time at which a function must be issued. Stop time is the time when the subsystem completes the function and becomes available for another function.

Tables used by the handler and labels used to reference the tables.



Bit 2<sup>28</sup> of word 13 = 1 indicates an interlock status.

Core Module (Preliminary)

Twenty-two words of "free" core are requested upon entry the handler. The area is used in the preliminary of the packet to determine the number of address needed to generate the Command Word List(s). If the number of words needed does not exceed 22 the same area is used for CWL construction.

B1T	0		Temp B1 storage
B3T	1		Temp B3 storage
	2		
MSA1	3	Sum of mass storage add. in file	
LIBCW	4	LI + BCW	
B5I	5	Initial B5 value	
	6		
WORDS	7	Misc.	
	8		
	9		
	10		
B5T	11	B5 at MSA - LI	
SPLIT	12	# of CWL's	# of BCW's
	13		
	14		



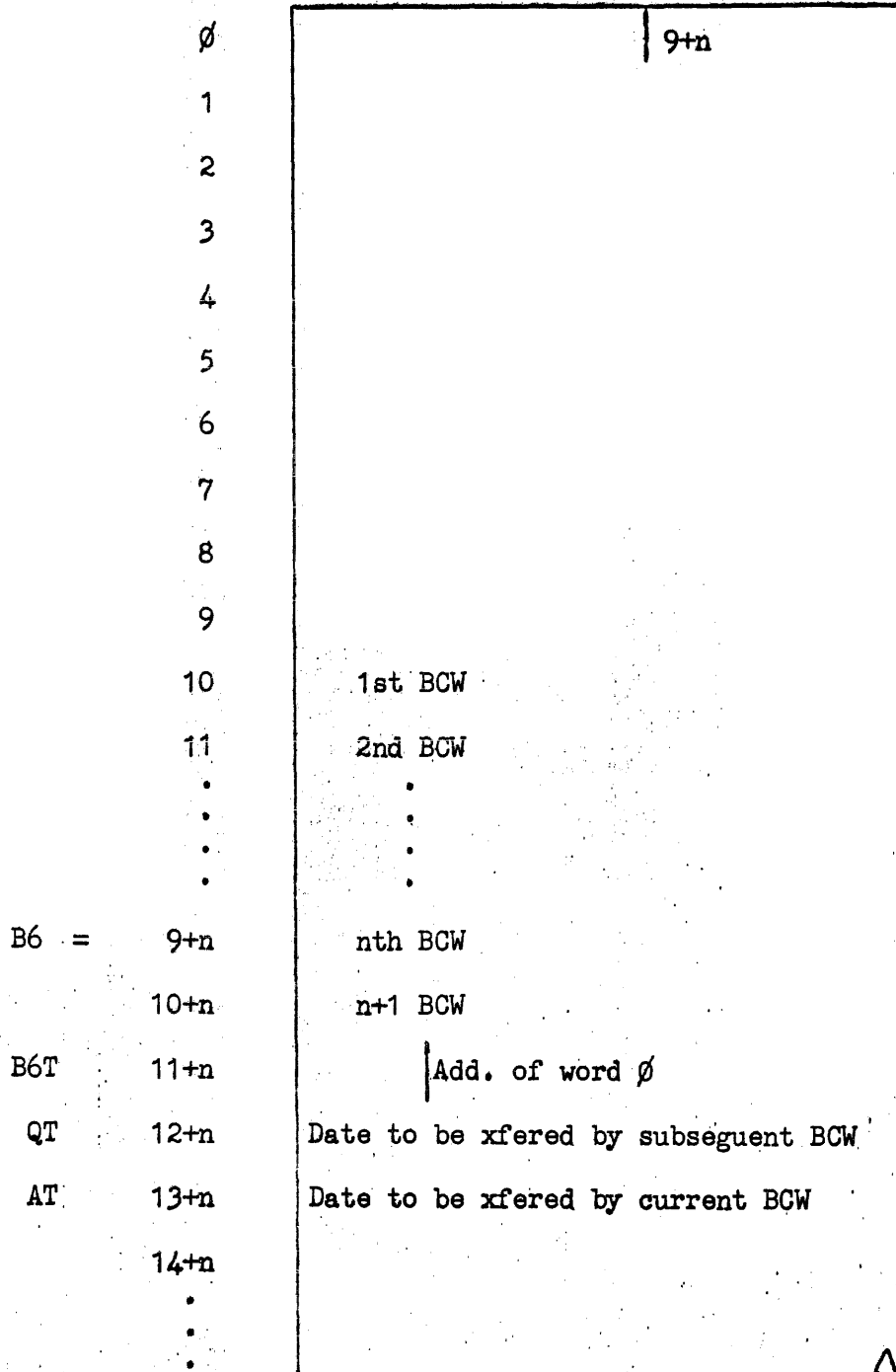
Core Module (Containing Command Words)  
 (B6 = add. of word 0)

	0		Add. of "present" BCW
	1		EI store add.
CWL	2	First Command word	
	3	2nd	
	4	3rd	
B5I	5		Initial Value of B5
STATUS	6		EOF indicator (4 or 0)
WORDS	7	(Q) at Formatter return	
POSTCM	8	P-TYPE (0,33D)	POST CM address
EF	9		EF
BCW	10	1st BCW	
	11	2nd BCW	
B6I	12		Add. of word 0
QTI	13	Temp storage of Q	
ATI	14	Temp storage of A	
	15		
	16		
	17		
	18		
	19		
	20		
	21		

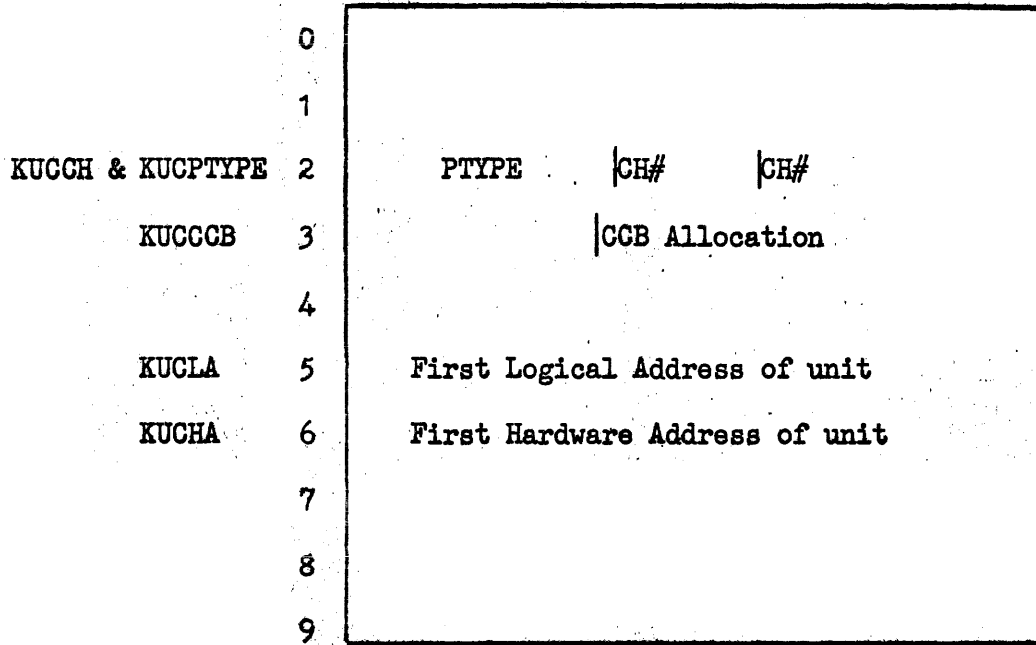
Core Module (at BCW generation)

During the generation of a Command Word List B6 equals one of two values:

- (1) it equals the add of the first word of the Core Module, (2) it equals the add of the BCW being generated. Under condition #1 the above labels are used. Under conditions #2 the following labels are used.



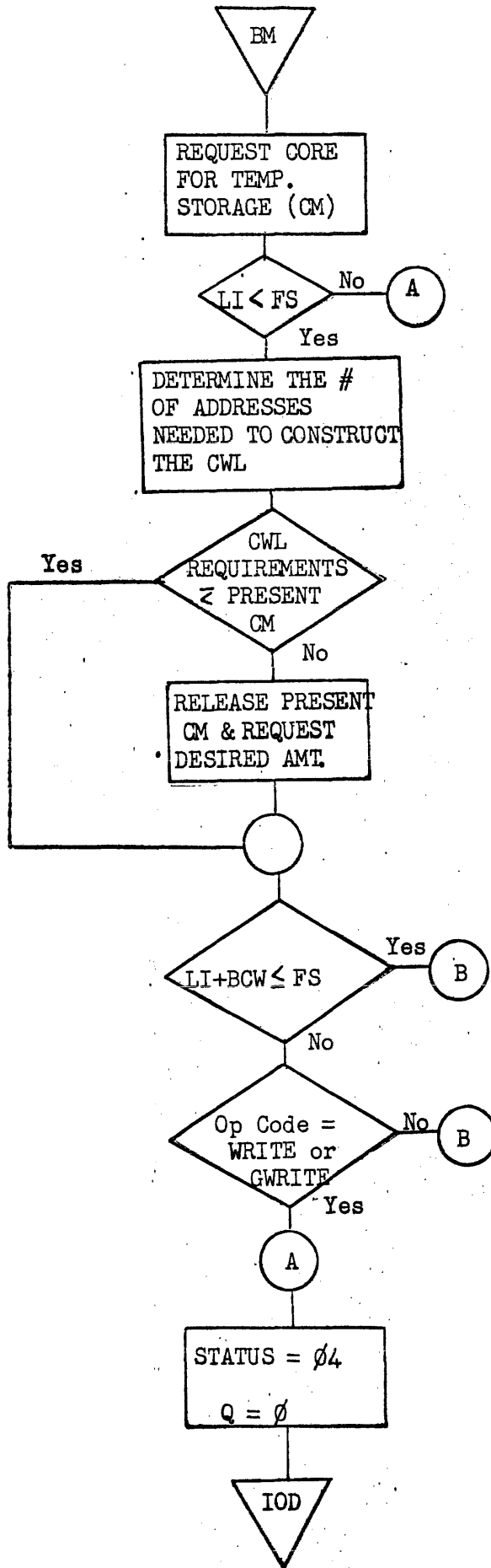
Unit Control Block

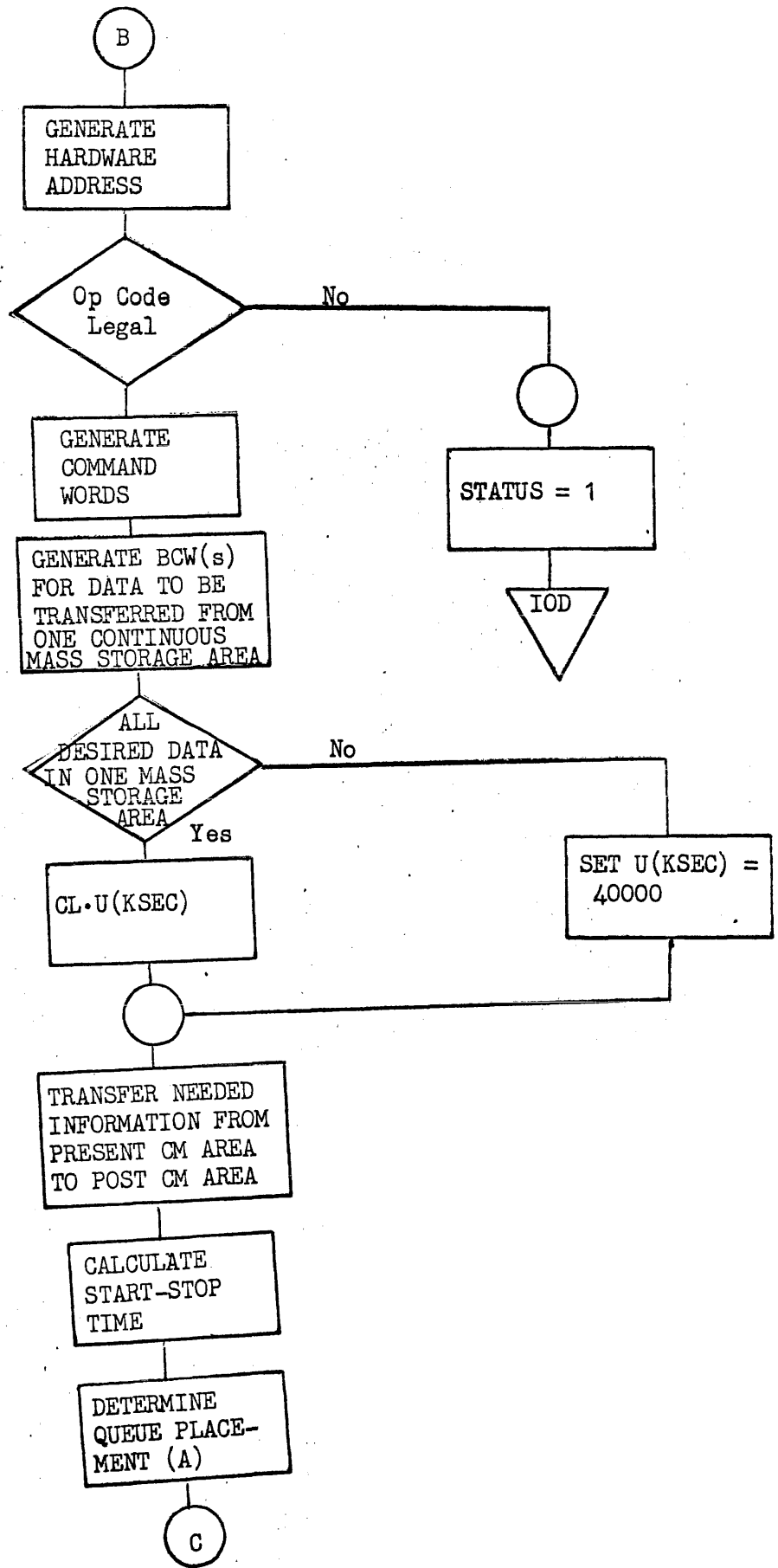


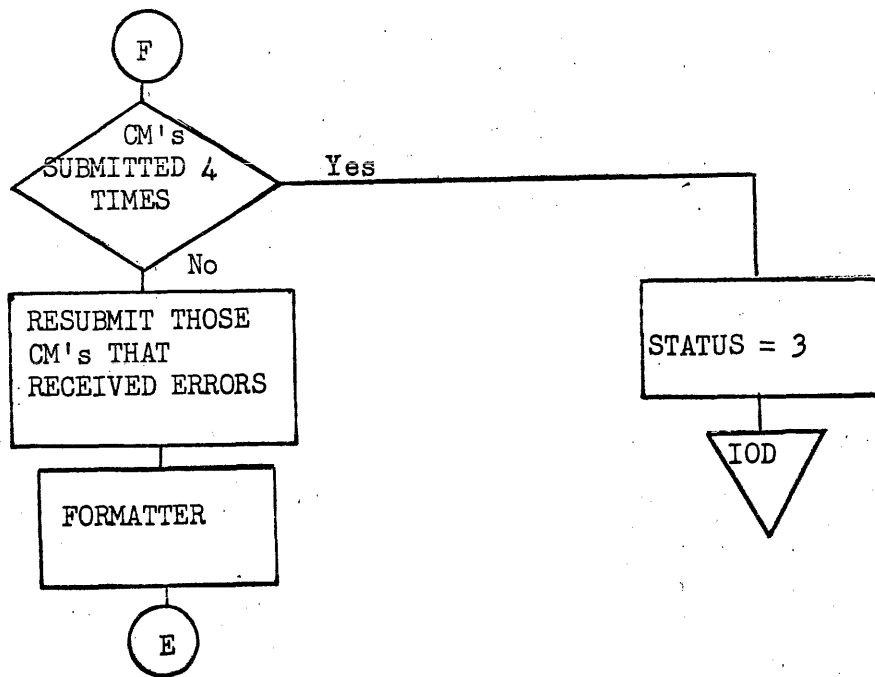
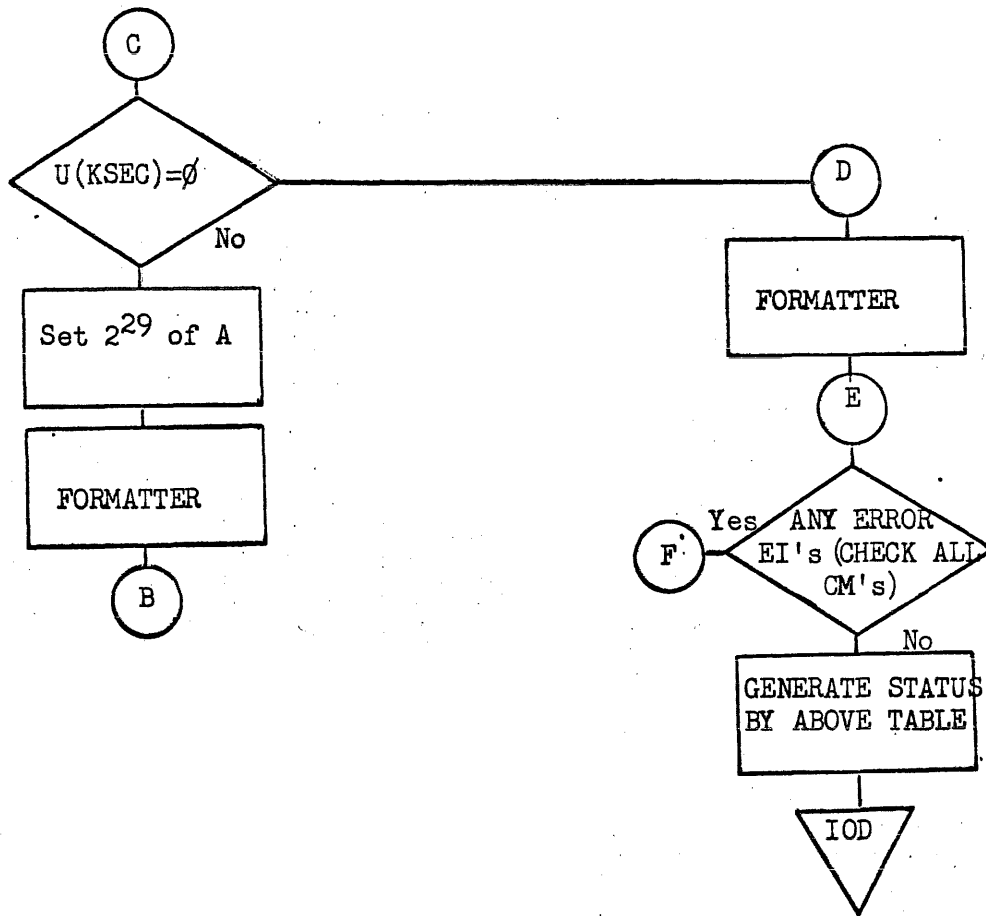
Mass Storage Handler

(FH880 Drum)

- Notes:
- LI - Logical Increment as given in a packet.
  - RASL - Random Access Storage List.
  - IOD - I/O Director
  - BCW - Number of data words to be transferred.
  - CWL - Command Word List(s).
  - SMOD - Storage Module
  - CM - Core Module (contains CWL)
  - FS - File Size as stated in the number of 30 bit words.







### 3.3.6 1004 Subsystem Handler

#### 3.3.6.1. Introduction

This document outlines the specifications for the U-494 OMEGA On-Line 1004 Handler and includes utilization of U-1004 Option 86.

#### 3.3.6.2. U-1004 Option 86

The following three (3) features are included in Option 86:

- A. External Interrupt - This feature enables timely and accurate status reporting by the U-1004 to the U-494 Central Processor(s) in the multi-programmed and/or multi-processor environment. The feature permits absolute control during "error" conditions that can occur in the U-1004 and enables the necessary synchronization for operations in this type system.
- B. Stop-Disable - This feature enables the U-1004 to maintain program control during a card reader, card punch, or printer error/fault conditions, while disabling the "stop" these conditions normally cause. The operator can recover the error condition and processing can continue.
- C. Punch Select - This feature enables automatic recovery from "punch-check" errors. The card following a mispunched card is selected along with the card in error. Recovery is programmed from the resultant error signal. No operator intervention is required and the correct card sequence is maintained in the output stacker.

#### 3.3.6.3. Operations

The operations of the U-1004 handlers will be compatible with the HSP and the Card Reader/Card Punch handlers insofar as the user is concerned.

##### A. Printer Operations

This operation duplicates that of the HSP operation. Form control is specified by the user through the OMEGA ASG\$ operator, wherein the user specifies the number of lines in the top and bottom margins and the number of printable lines per page, i.e.,  $TM+EM+PL$ =Total number physical lines per page. The absence of these parameters in the ASG\$ operator will indicate "No Form Control" and the handler will consequently relinquish form control to the user.



When form control is specified in the marginal and printable line information will be maintained in the Unit Control Block (UCB). A "current" line counter is also maintained in the UCB. The introduction of a new ASC\$ operator will cause the UCB to be reset for form control.

The OMEGA WRITE\$ operator is the only function acceptable for printer operations on the on-line 1004. Any other operator usage will result in an Inappropriate Function Status.

A maximum of 132<sub>10</sub> characters will be printed per line. The "Stop" code (77) will be honored.

Spacing with printing will be specified by the user within the OMEGA WRITE\$ operator (# of Words). Spacing on the U-1004 printer will occur before printing of the line. A space count of zero (Ø) will be honored, thus enabling over-printing. The maximum number of lines that may be spaced within one command is 63<sub>10</sub>=77<sub>8</sub>. Where the space count exceeds the number of printable lines remaining on a page will be interpreted by the handler as advancement of the form to the first logical print line on the next page.

Where form control has been specified by the user, the following operation will be honored by the handler. The user may "HOME" form (page eject) by setting the OMEGA WRITE\$ operator (# of words) to 40000<sub>8</sub>. The handler will initiate spacing to the first physical line (fold) of the next page.

The handler will also honor a "SPACE ONLY" form request from the user (with or without form control). The user may "SPACE" form by setting the OMEGA WRITE\$ operator (# of words) to 200SS<sub>8</sub>, where SS is spacing <63<sub>10</sub>=77<sub>8</sub>. No printing will occur during execution of this request on the U-1004. NOTE: where form control has been specified a "SPACE" instruction which carries into or beyond either margin of the current page will be interpreted by the handler as advancement of the form to the first logical print line on the next page and an appropriate spacing instruction will be issued in this case. If the user is desirous of spacing to the middle of the next page, for example, he must request a "HOME" form and then request spacing to the middle of the page along with the print line to be printed at that point, or must break the "SPACING" into two "SPACE ONLY" requests, the first of which carries into a margin.

The handler will convert all space counts from octal to decimal to XS-3, or 90 column code for presentation to the U-1004.

The handler will convert all print data from Fieldata to XS-3, or 90 column code for presentation to the U-1004.

## B. Card Operations

### 1) Card Reader Operations

Cards will be read in either 80 or 90 column format and in

either XS-3 or binary code. The user will specify which code he desires to be read through the OMEGA ASG\$ operator. The UCB will contain the format of the U-1004, i.e., whether it is a 80 or 90 column model. A mode indicator will be set in the UCB to key the handler to the "Translate" or "No translate" mode for XS-3 or binary code respectively. In the "Translate" mode the card image will be converted from XS-3 or 90 column code to Fielddata Code before presentation to the user. In the "No translate" mode the card image is presented exactly as received.

Single image card reads are possible through the OMEGA READ\$ operator. Multiple image card reads are possible through the OMEGA MREAD\$LIST operator. Any other operator usage will result in an Inappropriate Function Status.

## 2) Card Punch Operations

Card images to be punched will encompass the same features as explained in the Card Reader operations above, except that "Multiple" punch operations are not available in OMEGA. 80 or 90 column format and XS-3 or binary code will be processed by the handler. Data to be outputted to the U-1004 Card Punch will be translated from Fielddata Code to XS-3 or 90 column code, or translation will be bypassed in case of binary code.

Single image card punch operations are possible only by use of the OMEGA WRITE\$ operator. Any other operator usage will result in an Inappropriate Function Status.

### 3.3.6.4. Buffer Operations

#### A. Printer Operations

The Print Handler will be assigned a  $28_{10}$  word buffer as an integral part of its core module by the I/O Director. A maximum of  $132_{10}$  data characters will be moved from the user's buffer to the print handler buffer for output to the on-line 1004 printer. Detection of the "Stop" code (77) will abort the data move at that point. Thus a user need not necessarily provide a  $132_{10}$  character buffer.

#### B. Card Reader Operations

The user will furnish the input buffer(s) either as an integral part of his program or which he has acquired from an OMEGA core chain. The following MINIMUM size buffers are required:

##### 1) Single Image Card Reads

- a. 80 Col, Translate Mode must be  $\geq 16_{10}$
- b. 80 Col, No-translate mode must be  $\geq 32_{10}$
- c. 90 Col, Translate Mode must be  $\geq 18_{10}$
- d. 90 Col, No-translate Mode must be  $\geq 18_{10}$

##### 2) Multiple Image Card Reads

- a. 80 Col, Translate Mode must be  $\geq n(16_{10})$
- b. 80 Col, No-translate Mode must be  $\geq n(32_{10})$
- c. 90 Col, Translate Mode must be  $\geq n(18_{10})$
- d. 90 Col, No-translate Mode must be  $\geq n(18_{10})$

where n = the number of card images when using the MREAD\$LIST operator. In this multiple read packet n buffers are specified each of which must be  $\geq 16$ (or 18).

One big buffer which must accomodate all card images is not required!

### C. Card Punch Operations

The Punch Handler will be assigned a  $33_{10}$  word buffer as an integral part of its core module by the I/O Director. A maximum of  $80_{10}$  or  $90_{10}$  characters will be moved from the user's buffer to the punch handler buffer for output to the on-line 1004 card punch. The size of the user buffer is entirely optional by specifying number of words in the OMEGA WRITE\$ operator. The handler will move only the number of words specified and will space-fill any remaining area in the output buffer.

NOTE: Inconsistencies in buffer sizes by the user will in all instances result in an Inappropriate Buffer Status.

#### 3.3.6.5. U-494 Status Codes

The following status codes will be provided by the Handlers to the OMEGA I/O Director as a result of interrupt/error analysis:

RA of SMOD -  $\emptyset$  = Successful Completion  
RA of SMOD = 4000000001 = Inappropriate Function  
RA of SMOD = 4000000002 = Inappropriate Buffer Size

#### 3.3.6.6. U-494 Instructions to On-line U-1004

Instruction formats are based on the following scheme in the low order five bits of the instruction word:

$2^0 = 1 =$  Read  
 $2^1 = 1 =$  Punch  
 $2^2 = 1 =$  Print  
 $2^3 = 1 =$  No-translate;  $\emptyset =$  Translate  
 $2^4 = 1 =$  90 Column;  $\emptyset =$  80 Column  
 $2^5 = 1 =$  Space

The following constitute valid instructions acceptable to the on-line U-1004.

##### A. 80 Column

<u>OCTAL</u>	<u>DESCRIPTION</u>
40000 00001	Transfer card read image (XS3) to U-494
40000 00011	Transfer card read image (CI) to U-494

<u>OCTAL</u>	<u>DESCRIPTION</u>
40000 00002	Transfer card punch image (XS3) to U-1104
40000 00012	Transfer card punch image (CI) to U-1004
4XXXX 00004	Transfer Print Line to U-1104; spacing XXXX (XS3) decimal equivalent) before printing.
4XXXX 00044	Transfer space count XXXX (XS3 decimal equivalent) to U-1004 and space printer XXXX lines.

B. 90 Column

<u>OCTAL</u>	<u>DESCRIPTION</u>
40000 00021	Transfer card read image to U-494
40000 00022	Transfer card punch image to U-1004
4XXXX 00024	Transfer print line image to U-1004; space XXXX (90 Col decimal equivalent) lines before printing.
4XXXX 00044	Transfer space count XXXX (90 Col decimal equivalent) to U-1004; and space printer XXXX lines.

3.3.67. External Interrupts From U-1004 to U-494

<u>OCTAL</u>	<u>DESCRIPTION</u>
6XXXX 000FC	U-1004 cannot perform the requested function. 2 <sup>28</sup> of the instruction word received is set. and the instruction word is returned to the U-494 by the U-1004. This is termed a "reject interrupt".
4XXXX 00044	Space only instruction accepted and executed by U-1004.

3.3.6.8. U-1004 Operations

The U-494 will control the sequence of data flow and information as to what must be done to this data. The U-1004 will control its own peripheral devices, i.e., card reader, line printer, card punch, and the sequence of physical input/output events. The interface between the two systems, therefore, is one strictly to control the transfer of data and instructions from one to the other.

The U-1004 will effect data transfers immediately in most all cases. If it is unable to perform a requested function, an External Interrupt will be sent to the U-494, indicating that the function cannot be performed (reject interrupt). The U-1004 will then come to an orderly halt to allow operator correction of the problem, e.g., out of paper, card hopper(s) empty, etc.

The U-494 handler, upon analysis, of the external interrupt, will notify the I/O Director of the "Interlock" condition. This will cause a console typeout and suspension of the U-1004 channel operations. When the problem has been corrected, the U-494 operator will respond, via the console typewriter, that the problem is corrected. The U-1004 channel will then be reactivated and the I/O Director will resubmit the same request to the handler and U-1004 operations will resume from where they were discontinued. An illegal function code will cause the 1004 to send a reject interrupt.

The following operations will be performed by the U-1004:

- A. Space and Print - the OMEGA on-line 1004 handler will activate a 28<sub>10</sub> word output buffer. The U-1004 will accept the first word of this buffer, decode it and, if no previous print line is waiting, execute the form spacing specified. The following 27<sub>10</sub> words will then be accepted into working storage. If a print line is waiting in print storage, the new print line will be transferred into working storage (a card will be read if desired) before the waiting line is printed. 90 column code will be printed when operating in the 90 column mode.
- B. Space - The U-1004 will receive a one word instruction from the U-494. If no previous print line is waiting to be printed, the spacing of the form will be executed. An external interrupt will then be sent to the U-494, acknowledging the execution of the specified form control. If a print line is waiting to be printed, the line will be printed (a card will be read if desired) and the form control (space) instruction executed. As above, an external interrupt will now be sent to the U-494.
- C. Read - the U-1004 will receive a one word instruction, decode it and transfer the card image to the U-494. The U-1004 will then determine whether or not an I/O function must occur to have another card image available for the U-494. If so, another card will be read and the U-1004 will await receipt of the next instruction. The instruction will define the code to be read, i.e., XS-3, Code image, 90 column code.
- D. Punch - the OMEGA on-line 1004 handler will activate an output buffer of adequate size to accomodate XS-3, code image or 90 column code. The U-1004 will accept the first word of the buffer (instruction word decode it and wait for the punch to complete a prior task, if any, and then accept the remaining punch data words. The U-1004 will then start to punch the data and will await receipt of the next instruction. The instruction word will define type of code for the U-1004.

1004 PRINTER UNIT CONTROL BLOCK

WORD

0		RIR SETTING OF HANDLER
1		UCB LINK
2	PERIPHERAL TYPE	CHANNEL / UNIT
3		LINKING ADDRESS TO CCB
4	LENGTH OF UCB (11 <sub>g</sub> )	NO. REQUESTS TO CCB
5	MIN. SIZE OF CORE MODULE	INTERRUPT TSET
6		REQUEST QUEUE (HEAD OF CHAIN)
7	MODE INDICATOR	
10	UPPER MARGIN	LOWER MARGIN
11	CURRENT PRINT POSITION	# OF PRINTABLE LINES/PAGE

NOTES: WORD 7 -  $2^{29} = 1 = 90$  COLUMN  
 $L(\text{WORD } 11) = \emptyset = \text{NO FORM CONTROL}$

1004 CARD READER UNIT CONTROL BLOCK

WORD

∅		RIR SETTING OF HANDLER
1		UCB LINK
2	PERIPHERAL TYPE	CHANNEL/ UNIT
3		LINKING ADDRESS TO CCB
4	LENGTH OF UCB (10g)	NO. OF REQUESTS QUEUED TO CCB
5	MIN. SIZE OF CORE MODULE	INTERRUPT TSET
6		REQUEST QUEUE (HEAD OF CHAIN)
7	MODE INDICATOR	MULTIPLE READ COUNTER

NOTES: WORD 7 -  $2^{29} = 1 = 90$  COLUMN  
 $2^{28} = 1 =$  NO TRANSLATE



1004 CARD PUNCH UNIT CONTROL BLOCK

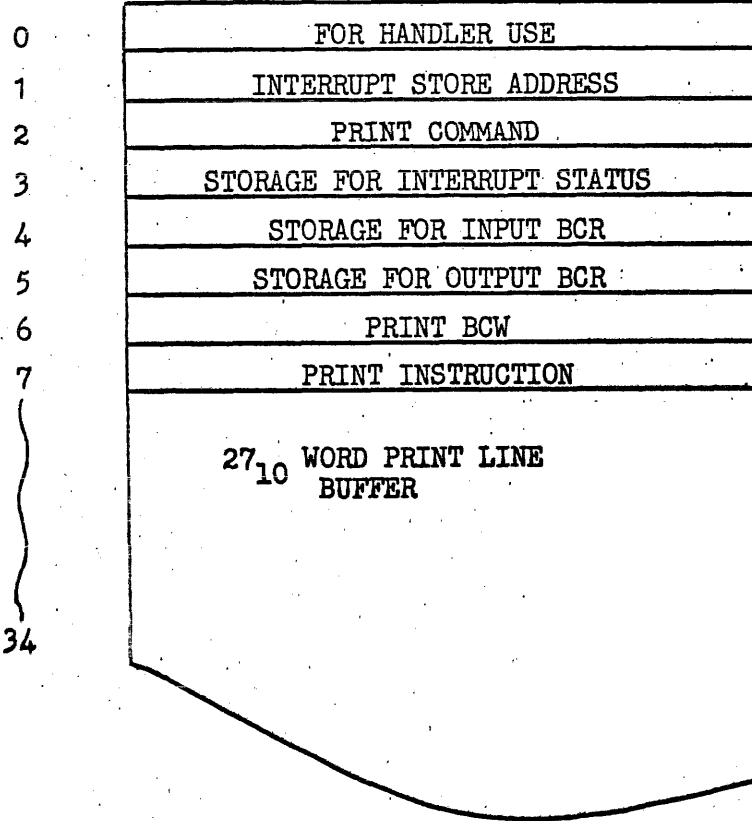
WORD

0		RIR SETTING OF HANDLER
1		UCB LINK
2	PERIPHERAL TYPE	CHANNEL / UNIT
3		LINKING ADDRESS TO CCB
4	LENGTH OF UCB (10g)	NO. OF REQUESTS QUEUED TO CCB
5	MIN. SIZE OF CORE MODULE	INTERRUPT TSET
6		REQUEST QUEUE (HEAD OF CHAIN)
7	MODE INDICATOR	

NOTES: WORD 7 - 229 = 1 = 90 COLUMN  
228 = 1 = NO TRANSLATE

1004 PRINTER CORE MODULE

WORD



1004 CARD READER CORE MODULE

WORD

0	FOR HANDLER USE
1	INTERRUPT STORE ADDRESS
2	READ COMMAND
3	INPUT BUFFER COMMAND
4	STORAGE FOR INTERRUPT STATUS
5	STORAGE FOR INPUT BCR
6	STORAGE FOR OUTPUT BCR
7	READ BCW
10	READ INSTRUCTION
11	INPUT BCW
12	MULTIPLE READ LIST POINTER

1004 CARD PUNCH CORE MODULE

WORD

∅  
1  
2  
3  
4  
5  
6  
7  
47

∅	FOR HANDLER USE
1	INTERRUPT STORE ADDRESS
2	PUNCH COMMAND
3	STORAGE FOR INTERRUPT STATUS
4	STORAGE FOR INPUT BCR
5	STORAGE FOR OUTPUT BCR
6	PUNCH BCW
7	33 <sub>10</sub> WORD PUNCH BUFFER
47	

NOTE: BUFFER ACCOMODATES EITHER 80 OR 90 COLUMN U-1004, OR BOTH. 1st WORD OF BUFFER CONTAINS PUNCH INSTRUCTION.

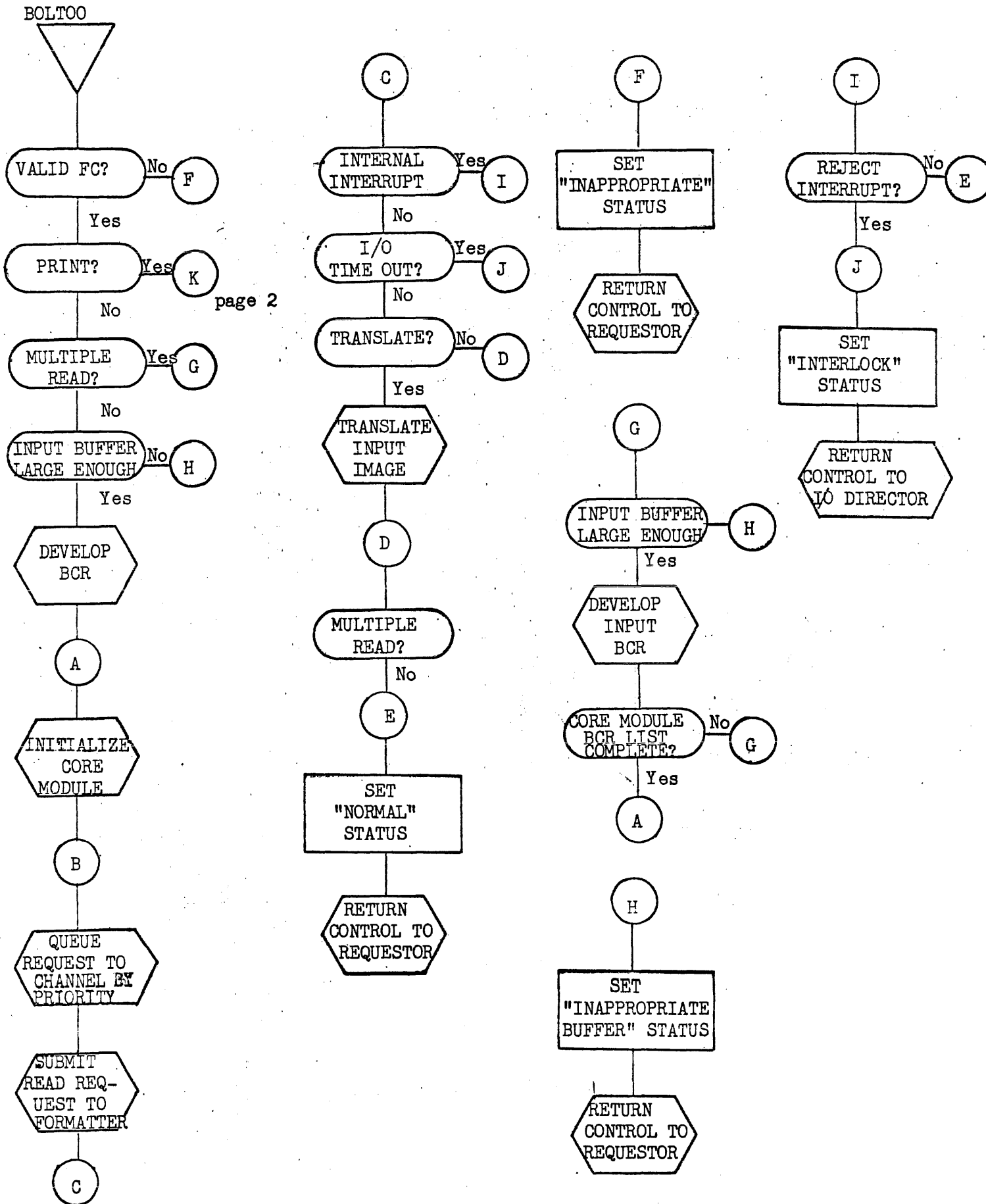
3.3.6.9. I/O STATUS CONDITIONS

REGISTER A AND/OR REGISTER B6 STATUS OP CODE	∅∅ NORMAL	∅1 INAPPROPRIATE FUNCTION	∅2 INAPPROPRIATE BUFFER	INTERLOCK
80 Column				
READ XS3 - ∅1	1	2	3	4
READ BINARY - 11	1	2	3	4
PUNCH XS3 - ∅2	1	2		4
PUNCH BINARY - 12	1	2		4
PRINT - ∅4	1	2		4
SPACE - 44	1	2		4
90 Column				
READ 21	1	2	3	4
PUNCH 22	1	2	3	4
PRINT 24	1	2		4
SPACE 44	1	2		4

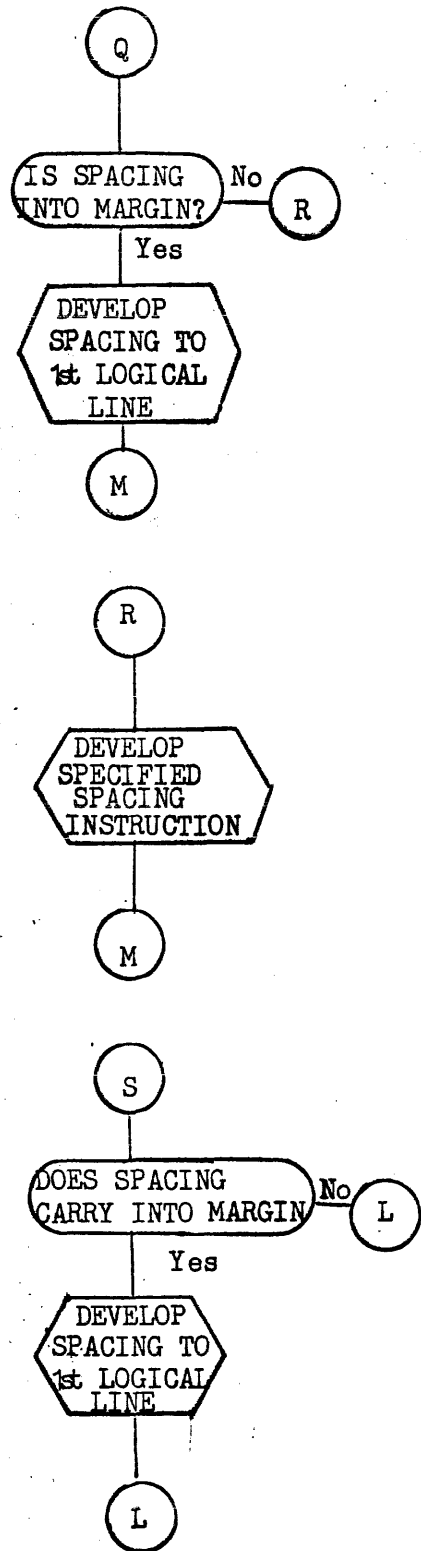
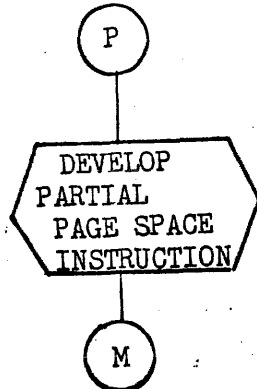
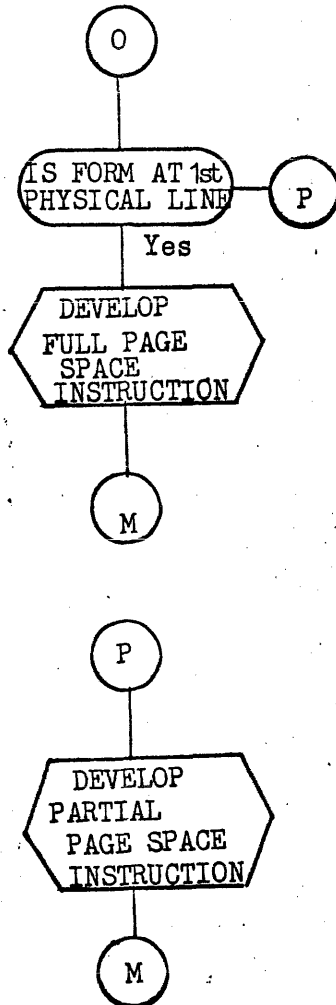
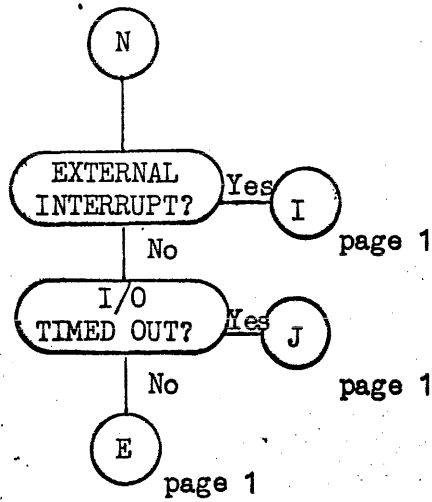
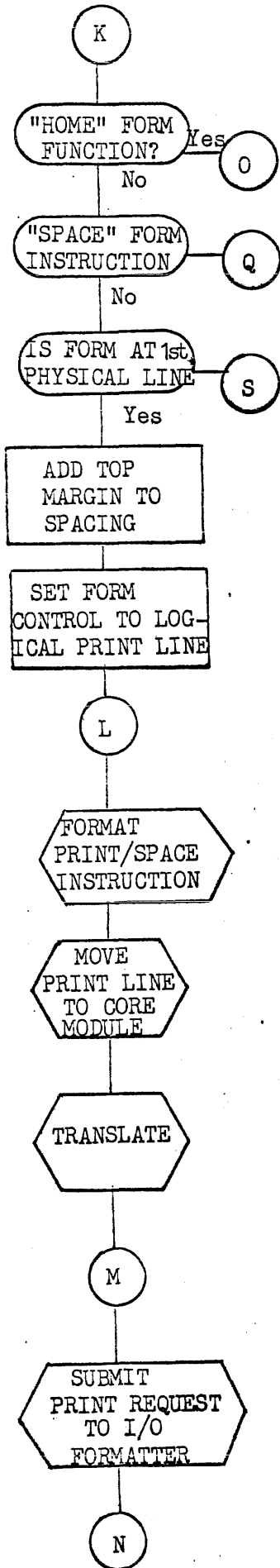
Key to Register Values Submitted to I/O Director

1. Normal Response. RA = ∅
2. Requestor has submitted invalid Function Code. RA = 4000000001
3. Requestor has specified Input Buffer size less than minimum required size for type image to be read. RA = 4000000002
4. The requested function cannot be executed by U-1004 at this time due to "out of paper", "empty card hopper", etc.

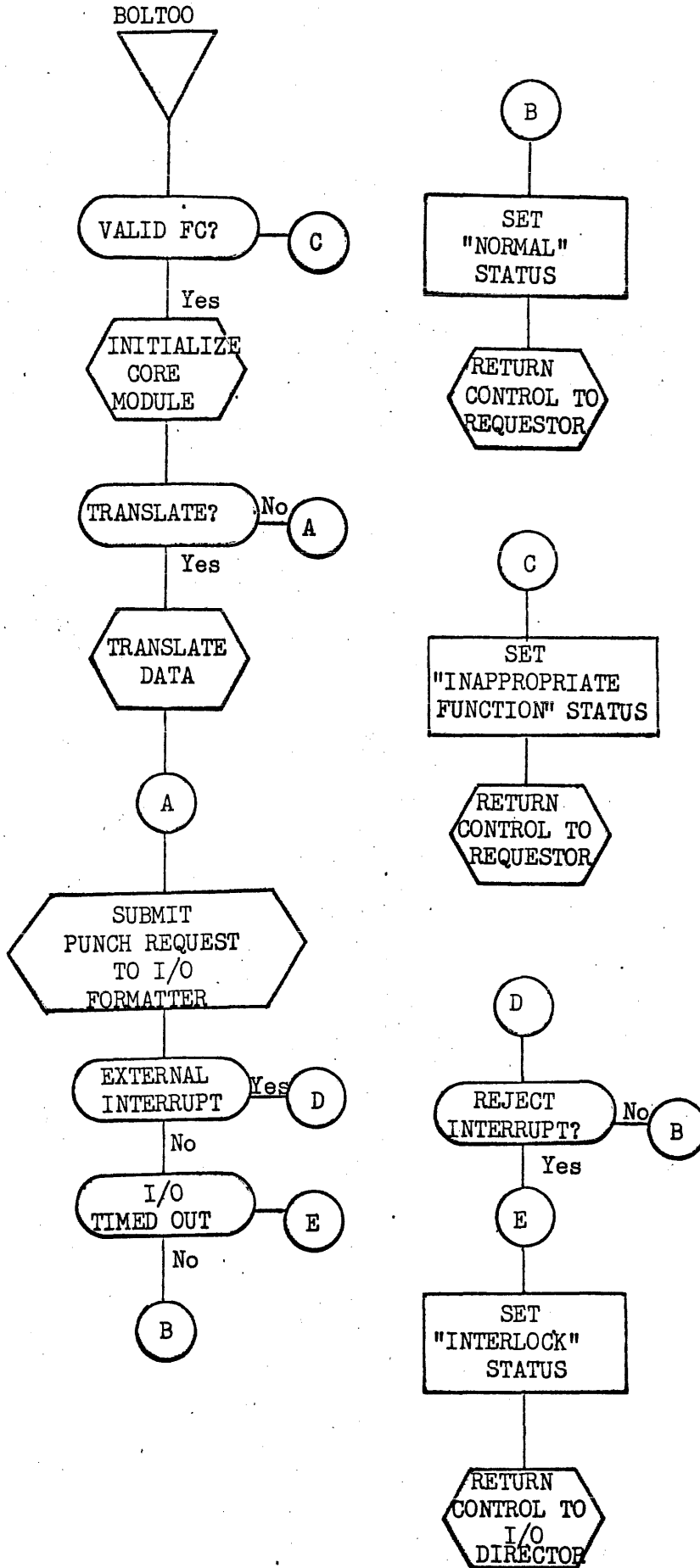
READ/PRINT HANDLER



page 2



PUNCH HANDLER





### 3.4 Auxiliary Routines

In order to conserve the amount of core memory required by OMEGA, certain auxiliary I/O routines are drum stored. These routines are loaded into core when they are needed and remain as long as they are active. Included in this group are the following: Error Recovery Routines, Interlock Routine, Search Routines, and Peripheral Initialization Routines.

#### 3.4.1 Error Recovery Routines

Error Recovery Routines, such as magnetic tape recovery involving a change of tape direction, are re-entrant routines listed with the Content Supervisor.

##### Entry

The appropriate Error Recovery Routine is entered from the Device Handler by a call to the Content Supervisor identifying the routine to be loaded. The call is shown below:

EXRN	V <sub>0</sub>
------	----------------

where V<sub>0</sub> is the mnemonic name of the appropriate Error Recovery Routine (this name will be equated to the correct Secondary Exec Library No. and function code).

When this call is made a Storage Module is allocated and linked to the Activity Addendum. Error Recovery receives control with the absolute address of the allocated S-mod in B4. The handler may set up parameters in the registers which may then be accessed by Error Recovery through the S-mod. The Storage Module assigned when the activity issued the I/O request is the next S-mod on the chain and its address may be found in the PUSH/POP Link location of the S-mod indicated by B4.

##### Exit

After performing its functions the Error Recovery Routine returns control to the Device Handler. Control is returned through the Content Supervisor by the packet shown below:

ENT*B1	0 0 0 0 0
DRET1\$	0 0 0 0 2

Control is returned to the handler at the location following the call. Registers are restored to their value at the time Error Recovery was called unless they are changed in the S-mod by Error Recovery (the success or failure of the recovery attempt may be indicated in the A and/or Q register position of the S-mod).

## Operation

Error Recovery examines the hardware status code(s) to determine the proper recovery procedure. Error Recovery may then obtain an area of free core to set up the recovery functions, or it may resubmit the original module which was set up by the Device Handler, or it may do both. For example, the magnetic tape recovery routine may obtain free core to set up the repositioning functions and then submit the original function. Error Recovery Routines submit functions directly to the Formatter as do the Device Handlers.

### 3.4.2 Interlock Routine

The Interlock Routine, as the Error Recovery Routines, is a re-entrant Secondary Executive Routine listed with the Content Supervisor. Its purpose is to provide a common routine which will handle interlock type out messages for the Device Handlers.

#### Entry

The Interlock Routine is entered by the Device Handlers through the call to the Content Supervisor shown below:

EXRN*INLK
-----------

where "INLK" is the mnemonic name of the Interlock Routine. This name will be equated to the Secondary Exec Library number and function code of the Interlock Routine.

Before calling the Interlock Routine, the Device Handler must set B5 equal to the absolute address of the Unit Control Block of the interlocked device. This parameter is used to identify the channel and unit of the device.

#### Exit

The Interlock Routine returns to the Device Handler at the location following the call when the operator has answered the type out. Return is made through the following packet:

ENT*B1	0 0 0 0 0
DRET1\$	0 0 0 0 2

All registers will be restored to their value at the time of the call with the exception of the A-register which will contain a status code:

0 0 0 0 0\*0 0 0 0 0 - the interlock condition has been corrected.

7 7 7 7 7\*4 0 0 1 1 - the interlock condition cannot be corrected.

### Operation

The Interlock Routine obtains an area of free core and builds the interlock message in this area. The channel and unit contained in the Unit Control Block, identified by B5, are placed in the message area. The message is then given to the I/O Director for assigning a response number and placing on the console output queue (see Section 3.2.2). The output message will take the following form:

"INTERLOCK CXX UYY DZZ"

where XX is the octal channel number  
YY is the octal unit number  
ZZ is the response (delay) number

When the operator has responded control is returned to the Interlock Routine with the input message in the A-register. The routine then stores the appropriate status code in the A-register position of the S-mod (address in B4) and transfers control, through the Content Supervisor, to the Device Handler.

The operator response to the interlock message is of the following form:

DZZ  $\square$   $\left\{ \begin{array}{l} Y \\ N \end{array} \right.$   $\textcircled{S}$

where ZZ is the response number of the interlock type out.

Y is entered if the interlock has been corrected. The handler will resubmit the function to the Formatter.

N is entered if the interlock cannot be corrected. The handler will return an error status to the requesting activity.

### 3.4.3 Search Routine

The Search Routine is a re-entrant Secondary Executive Routine listed with the Content Supervisor. Its purpose is to provide a software simulation of those hardware search functions which may unduly monopolize a peripheral subsystem for extended periods of time.

#### Entry

The Search Routine is entered by a call to the Content Supervisor generated in response to certain search mnemonics. These mnemonics generate standard I/O packets (See Section 3.5.2) with the call directed to the Content Supervisor instead of the I/O Director. The following mnemonics will cause entry to the Search Routine:

- SEARCH\$ - generalized search, tape or mass storage.
- SEARCHL\$ - same as above with logical lock of mass storage.
- BLOCKSL\$ - block search with logical lock (drum subsystems only)
- SEARCHTL\$ - search track with logical lock (FASTRAND subsystems only)
- SEARCHPL\$ - search position with logical lock (FASTRAND subsystems only).

The BLOCKS\$, SEARCHT\$, and SEARCHP\$ operators (as above only without logical lock) are hardware search functions. Their counterparts with logical lock must be software searches since the find address is not indicated by the hardware on search read operations. The area is locked only if a find is made.

#### Exit

At the completion of its operations the Search Routine places status information in the A and Q registers of the Storage Module associated with the operation (address in B4 at entry) and exits to the calling routine through the Content Supervisor with the following packet:

ENT*B1	0 0 0 0 0
DRET1\$	0 0 0 0 2

Status information returned by the Search Routine is given in Section 3.5.3.

## Operation

The Search Routine issues Read functions to the I/O Director in the normal manner. The parameter packet, including buffer, is the search packet. For tape searches the first word of the buffer is compared to the search identifier; for mass storage searches each word of the buffer is compared to the search identifier. Repetitive reads are given by the Search Routine until either a find is made or an error condition occurs.

When a find is made on a mass storage search another Read (with or without lock, as appropriate) is initiated beginning with the find address. The actual functions performed for the various search operators are given below.

- SEARCH\$ - READ\$ functions are given. When a find is made control is returned (magnetic tape), or another READ\$ is given beginning at the find address (mass storage).
- SEARCHL\$ - READ\$ functions are given. When a find is made a READL\$ is given beginning at the find address.
- BLOCKSL\$ - BLOCKR\$ functions are given. When a find is made a BLOCKRL\$ is given beginning at the find address.
- SEARCHTL\$ - READ\$ functions are issued. When a find is made a READL\$ is given beginning at the find address.
- SEARCHPL\$ - READ\$ functions are issued. When a find is made a READL\$ is given beginning at the find address.

### 3.4.4 Peripheral Initialization

Peripheral Initialization Routines are drum stored routines called by Facility Assignment to assist in setting up the Unit Control Block and to perform any functions required for the initialization of the peripheral device. The Initialization Routine to be used with each Device Handler is indicated in the Handler Description (See Section 6.6).

#### Entry and Exit

The Initialization Routines are entered from and exit to Facility Assignment via the Return Jump instruction (Facility Assignment is a non re-entrant routine).

#### Operation

As an ASSIGN statement is processed by Facility Assignment a Unit Control Block will be partially formed. The remainder of the UCB is to be set up by the initialization code for this peripheral type.

#### Facility Assignment will:

1. Secure core for the UCB, the number of words will be specified in the Unit Descriptor.
2. Set up the first 7 words of the UCB which are similar for all types of devices.
3. Will load the initialization code which was specified with the handler of this peripheral.
4. Form a buffer containing Assign and Unit Descriptor information.
5. Enter the initialization coding by a Return Jump instruction.

#### Initialization Entrance Parameters:

- B1 = 15 bit address of a buffer which contains Assign information and the Unit Descriptor.
- B4 = Storage Module address. This value must be retained.
- B5 = 17 bit address of the Unit Control Block which is being formed.

Initialization will interpret the option and estimate information from the Assign statement in conjunction with the Unit Descriptor to form the UCB word 7 and on up. Any required initialization functions such as clear card memory should be performed at this time.

The operator may be notified of any abnormal conditions by forming a LOG statement in a buffer obtained from free core. The buffer with the LOG statement will be passed to Facility Assignment as the Initialization exits. Exit is by a JP to the L(RJP entrance location) with appropriate exit parameters.

Upon receiving an error status from the initialization routine facility assignment will submit the message generated by the initialization routine to the LOG routine for processing. Control will be returned to the requestor of the assignment from the LOG routine with the appropriate status of the assignment.

#### Initialization Exit Parameters

REG A = 00000 00000 Successfully completed

REG A = 77777 40000 Unsuccessful - Reject ASSIGN request

REG A = 77777 40001 Unsuccessful - Perform a LOG operation (Console typeout to operator) and release LOG buffer back to FREE CORE.

B7 = Absolute memory address of the LOG statement which is to be typed out.

Q = LOG message length - number of words.

ASSIGN AND UNIT DESCRIPTOR BUFFER

B1 POINTS  
TO HERE

0	OPTIONS	
1	ESTIMATE (1st WORD)	
2	ESTIMATE (2nd WORD)	
3	ESTIMATE (3rd WORD)	
4	P-TYPE	LENGTH OF U.D.
5	CH/UNIT	CH/UNIT
6	LENGTH OF U.C.B.	
7	DEVICE DEPENDENT	
10	INFORMATION	
11		

FROM THE  
ASSIGN STATEMENT

UNIT DESCRIPTOR  
AS LISTED IN THE  
FACILITY MAP

Options - Master bits set to 1 to represent the alphabetic character options contained in the ASSIGN statement.  
 $2^{29} = A$ ,  $2^{28} = B$ , - - - - -  $2^4 = Z$

Estimate - Binary representation of the estimate parameter of the Assign statement.

The estimate field from the Assign statement may consist of up to three parts. ( , 1 / 2 / 3 , )  
 Each part will occupy one word in the estimate storage set up for the initialization routine. Any part which is not present will cause all 7—7's to be placed in the appropriate word in the estimate storage.



An additional "Initialization Routine" is operated as a segment of the Facility Release Routine. The purpose of this initialization is to "clean up" the device in anticipation of the next assignment.

The segment is called when Facility Release detects  $2^{29}$  set in the UCB word containing the peripheral type code. This bit is set when the UCB is formed depending on a control bit in the Handler Description.

The action taken by this "Initialization Routine" is dependent upon peripheral type as follows:

- Standard Card Punch - Punch three blank cards so that the last card punched by the user is error checked and an automatic run-out is achieved.
- 1004 Card Punch - Punch two blank cards so that the last card punched by the user is error checked and an automatic run-out is achieved.
- High Speed Printer - Space paper to the first physical line of the next page. This is done only if a page length has been specified and the paper is not already at the Home Paper position.
- 1004 Printer - Same as High Speed Printer. The Home Paper function will force out the last print line which is held in 1004 memory.

## 3.5 User Interface

### 3.5.1 File Codes

File codes represent the basic mechanism by which an operating activity communicates I/O requests to the system. At time of activation by control card or through internal request, the user expresses the type of unit or random access file to be assigned to the file code. Once established the user presents the file code with each I/O packet submitted to the system and thereby making the object code relatively independent of device.

Each task addendum is provided with a basic set of 25 file codes, A through Y, to which the user may assign peripheral devices or random storage files. In cases where these are insufficient, the user may specify that a designated file code is to be fragmented into an additional 26 codes which would contain the same characteristics as the original set.

Example: If the file code B is fragmented, the new set would be referred to as BA, BB . . . BZ

File code Z of the basic set is reserved for systems references and is fragmented as such:

<u>File Code</u>	<u>Reserved Use</u>
ZA	Unit record primary input
ZB	Unit record primary output
ZC	Unit record secondary output
ZD	Cooperative Library
ZE	Systems Library
ZF	Job Library

The only routines eligible for referencing these file codes are those contained in the systems library.

### 3.5.2 I/O Requests

OMEGA I/O requests are generated in response to mnemonics consisting of an operator and a specification list. The operator defines a specific executive call and function code. A parameter packet is formed from the specification list of the request. The general format is given below.

Operator\$ASpecification, Specification . . .

where the specification list could contain any following dependent upon operation or device

- V<sub>0</sub> - File Code - is an alphabetic code identifying the logical unit referenced.
- V<sub>1</sub> - # of words - specifies the length of the data buffer. Length for any one buffer is limited to 4K.
- V<sub>2</sub> - Buffer base - contains address of buffer relative to lower lock setting of activity.
- V<sub>3</sub> - Logical address - increment from base of file normally used with random access storage references; but may be present with sequential files to effect device independence.
- V<sub>4</sub> - Search word - 10 octal character number giving search identifier to be used in a search operation.

The following is a summary of the basic set of I/O operator recognized by the system, their function codes, specifications and applicable devices. This is followed by a more detailed description of the specific operator and parameter packet.

3.5.2-2

Function Code	Random Access Operators	V-Operands Used	Notes	UNISERVO Operators	V-Operands Used	Notes
01	READ\$	0,1,2,3		READ\$	0,1,2	
02	WRITE\$	0,1,2,3		WRITE\$	0,1,2	
03	BLOCKR\$	0,1,2,3	1			
04	SEARCHT\$	0,1,2,3,4	2			
05	SEARCHP\$	0,1,2,3,4	2			
06	SEARCH\$	0,1,2,3,4	3	SEARCH\$	0,1,2,4	3
07	BLOCKS\$	0,1,2,3,4	1			
10	READB\$	0,1,2	4	READB\$	0,1,2	6
11	READL\$	0,1,2,3				
12	WRITER\$	0,1,2,3				
13	BLOCKRL\$	0,1,2,3	1			
14	SEARCHTL\$	0,1,2,3,4	2 & 3			
15	SEARCHPL\$	0,1,2,3,4	2 & 3			
16	SEARCHL\$	0,1,2,3,4	3			
17	BLOCKSL\$	0,1,2,3,4	1 & 3			
20	WRTEOF\$	0	4	WRTEOF\$	0	
21	REWIND\$	0	4	REWIND\$	0	
22	REWINDI\$	0	4	REWINDI\$	0	
23	ERASE\$	0	4	ERASE\$	0	
24	SREAD\$	LIST	5	SREAD\$	LIST	5
25	GWRITE\$	LIST	5	GWRITE\$	LIST	5
26	MREAD\$	LIST	5	MREAD\$	LIST	5

- Note 1 FH-880 Drum only  
 2 FASTRAND only  
 3 Directed to the Content Supervisor  
 4 Function returned as "Successful completion"

- Note 5 V-Operand is a tag defining a list packet  
 6 Backspace Block on IIIC/IVC

Summary of Unit Record Equipment

Equipment Type	Function Code	Operator	V-Operands Used
Card and Paper Tape	01	READ\$	0,1,2
	02	WRITE\$	0,1,2
	26	MREAD\$	Tag
High Speed Printer	02	WRITE\$	0,1,2,3*

\* Specifies number of lines to space before print

General I/O Operators:

- READ - directs the system to transfer data from specified peripheral device to indicated core buffer. If peripheral device is random access storage number of words transferred will be that specified as buffer length. If UNISERVO or unit record device is assigned number words transferred will be that specified as buffer length or when end of block or record is reached. The READ operator is as follows:

READ\$ Δfile code, # of words, buffer base, logical address

Packet

EBJP*B7	N
FILE CODE	# OF WORDS
	BUFFER BASE
	LOGICAL ADDRESS
N EXRN	1 0 0 0 1

Executive Entry instruction

- WRITE - data from specified buffer is transcribed to storage. The user must maintain a logical address, i.e., is the sum of words written. This address will be used in the event the file is allocated to random access storage.

An end-of-file error status indicates that the end of allocated area has been reached. An additional area, which is logically continuously addressable, may be added on request.

The function may be specified by:

WRITE\$ ΔFile code, # of words, Buffer base, Logical address

Packet

EBJP*B7	N
FILE CODE	# OF WORDS
	BUFFER BASE
	LOGICAL ADDRESS
N EXRN	1 0 0 0 2

- READL - Read lock performs the same function as READ with the addition that an entry for the accessed area is made on a lock-list. This lock will prevent any other READL function within this area until released. Conflicting requests are requeued until they can be performed.

The function may be requested by:

READL\$ Δfile code, # of words, buffer base, logical address

The specification list and packet generated are identical to READ except that the executive entry is generated as EXRN\*10011.

- WRITER - the write release function is the counterpart of READL in that an area on the lock-list is released by the function within the area. If no write function is to be performed a release is effected by specifying an address within the lock area with number of words specified as zero. Otherwise, WRITE release performs the same function as WRITE.

The specification list and packet generated are identical to WRITE except that the executive entry instruction is generated as EXRN\*10012. The function may be specified by:

WRITER\$ Δfile code, # of words, buffer base, logical address

- SEARCH - this function compares the first word of each block against the specified identifier. Reading is initiated when a match is made to include the search identifier within the buffer. If end of the file is detected a no find status word is returned. If mass storage the find address is returned in the A-register.

OMEGA simulates the search function through use of repetitive reads to prevent undue tie-up of the synchronizer. Since the buffer for this function becomes the blocking factor for random storage; in the contingency of allocation to tape the buffer must be equal to or greater than the buffer used in writing the file. For the contingency of allocation to FAstrand, the buffer might also be a multiple of 33 words to improve efficiency. A locate function on tape can be effected by specifying # of words as zero. The SEARCH operator is as follows:

SEARCH\$ Δfile code, # of words, buffer base, logical address, searchword

Packet

EBP*B7	N
FILE CODE	# OF WORDS
	BUFFER BASE
	LOGICAL ADDRESS
	SEARCH IDENTIFIER
N EXRN	2 0 1 0 6

Executive Entry

- SEARCHL - search lock performs the same function as SEARCH with the addition that an entry for the accessed area is made on the lock-list when a successful search occurs.

The specification list and packet generated are identical to SEARCH except that the executive entry instruction is generated as EXRN\*20116. The function may be specified by:

SEARCHL\$Δfile code, # of words, buffer base, logical address, search word

- REWIND - position file to load point. This function is ignored if random access was allocated. The programmer must, therefore, reset the logical address to zero to provide for the even of random storage allocation. Operator is as follows:

REWIND\$Δfile code

Packet

ENT * B7	FILE CODE				
EXRN	1	0	0	2	1

- REWINDI - rewind with interlock performs the same function as REWIND except that the tape is interlocked for operator intervention. This function is ignored if random access was allocated. Operator is as follows:

REWINDI\$Δfile code

Packet

ENT *B7	FILE CODE				
EXRN	1	0	0	2	2

- WRTEOF - write end-of-file transcribes a point recognized as EOF when encountered by a read. When this operator is used to mark the end of recorded tape there should be as many EOF marks as there are standby buffers or block reads in a MREAD operator. This function is ignored if random access storage is allocated. The operator is as follows:

WRTEOF\$Δfile code

Packet

ENT * B7	FILE CODE				
EXRN	1	0	0	2	0

- ERASE - the erase function results in erasure of a fixed area of tape. This function is ignored if random access storage is allocated. The operator is as follows:

ERASE\$Δfile code

Packet

ENT * B7	FILE CODE				
EXRN	1	0	0	2	3

- READB - read backward function moves tape in the backward direction and the data enters the buffer in the order encountered in movement of the tape.

The specification list and packet generated are identical to READ except that the executive entry is generated as EXRN\*10011. This function is ignored if random access storage is allocated. The function may be specified by the operator:

READB\$Δfile code, # of words, buffer base

- FH Drum - specific allocation requests for FH432, FH880, etc., may be made. The functions applicable to a random file are augmented by the following operators.
- BLOCKR - the block read function is a READ function that is terminated by the end-of-block sentinel (all one's).

The specification list and packet generated are identical to READ except that the executive entry instruction is generated as EXRN\*10003. The function may be specified by:

BLOCKR\$Δfile code, # of words, buffer base, logical address

- BLOCKRL - block read lock performs the same function as BLOCKR with the addition that the lock-list mechanism is employed.

The specification list and packet generated are identical to READ except that the executive entry instruction is generated as EXRN\*10013. The function may be specified by:

BLOCKRL\$Δfile code, # of words, buffer base, logical address

- BLOCKS - the block search function is a hardware function which compares consecutive words on drum against the specified identifier. Reading is initiated when a match is made to include the search identifier within the buffer. The function is terminated by the end-of-block sentinel (all one's).

The specification list and packet generated are identical to SEARCH except that the executive entry instruction is generated as EXRN\*10007. The function may be specified by:

BLOCKS\$Δfile code, # of words, buffer base, logical address, search word

- BLOCKSL - block search lock performs the same function as blocks with the addition that the lock list as described with READL is employed.

The specification list and packet generated are identical to SEARCH except that the executive entry instruction is generated as EXRN\*20117. The function may be specified by:

BLOCKSL\$Δfile code, # of words, buffer base, logical address, search word



- FASTRAND - specific allocation requests for types of FASTRAND storage may be made. The functions applicable to a random file are augmented by the following operators:
- SEARCHT - the search track function is a hardware function which compares the first word of consecutive FASTRAND sectors against the specified identifier. Reading is initiated when a match is made to include the search identifier within the buffer. The function is terminated by the end-of-track interrupt. A locate function can be effected by specifying # word as zero.

The specification list and packet generated are identical to SEARCH except that the executive entry is generated as EXRN\*10004. The function may be specified by:

SEARCHT\$Δfile code, # of words, buffer base, logical address,  
search word

- SEARCHTL - search track lock performs the same function as SEARCHT with the addition that the lock-list mechanism is employed.

The specification list and packet generated are identical to SEARCH except that the executive entry is generated as EXRN\*20114. The function may be specified by:

SEARCHTL\$Δfile code, \$ of words, buffer base, logical address,  
search word

- SEARCHP - the search position function is a hardware function which compares the first word of consecutive FASTRAND sectors against the specified identifier. Reading is initiated when a match is made to include the search identifier within the buffer. The function is terminated by the end-of-position interrupt. A locate function can be effected by specifying # of words as zero.

The specification list and packet generated are identical to SEARCH except that the executive entry is generated as EXRN\*10005. The function may be specified by:

SEARCHP\$Δfile code, # of words, buffer base, logical address,  
search word

- SEARCHPL - search position lock performs the same function as SEARCHP with the addition that the lock-list mechanism is employed.

The specification list and packet generated are identical to SEARCH except that the executive entry is generated as EXRN\*20115. The function may be specified by:

SEARCHPL\$Δfile code, # of words, buffer base, logical address,  
search word

Multiple Read/Write Service Requests - the following operators describe special purpose functions where data is to be transcribed from or to non-continuous areas of core, random access storage, or tape. Each function is composed of two operators; one to specify the operation to be performed and the second supplies the parameters required to perform the operation.

- SREAD - scatter read requests the system to transcribe data from continuous area of random access storage or a single tape block into one or more non-continuous core buffers as specified by buffer words in LIST operator.

SREAD\$ Δ label of LIST operator

Packet

ENT * B7	LABEL
EXRN	1 0 0 2 4

- GWRITE - Gather write requests the system to transcribe data from one or more non-continuous core buffers on to one continuous area of random access storage or a tape block as specified by the LIST operator.

GWRITE\$ Δ label of LIST operator

Packet

ENT * B7	LABEL
EXRN	1 0 0 2 5

- MREAD - multiple read requests the system to transfer data from one or more non-continuous areas of random access storage or multiple tape blocks into one or more core buffers as described by the LIST operator.

MREAD\$ Δ label of LIST operator

Packet

EXT * B7	LABEL
EXRN	1 0 0 2 6

List Operators

I/O requests may be generated with remote parameter packets in lieu of the in-line packet. In this case the specification list is generated by the LIST operator. When this operator is used the I/O request specifies the label of the LIST operator. Three LIST operators are provided.

## LIST\$

The LIST\$ operator may be used in the conjunction with the following I/O operators: READ\$, WRITE\$, READL\$, WRITER\$, BLOCKR\$, BLOCKRL\$, SEARCH\$, SEARCHL\$, BLOCKS\$, BLOCKSL\$, SEARCHT\$, SEARCHTL\$, SEARCHP\$, and SEARCHPL\$. The LIST\$ operator is as follows:

LABEL → LIST\$ Δ file code, # of words, buffer base, logical address, search I.D.

The packet generated is as follows:

LABEL	FILE CODE	# OF WORDS
		BUFFER BASE
	LOGICAL ADDRESS	
	SEARCH IDENTIFIER	

Note: If Search I.D. is present, logical address must be present.

Associated with the above LIST\$ operator, a Read request would appear as follows:

READ\$ Δ LABEL

The generated call would then be:

ENT * B7	LABEL				
EXRN	1	0	0	0	1

## LISTA\$

The LISTA\$ operator is used in conjunction with the following I/O operators: SREAD\$, GWRITE\$, and MREAD\$ on unit devices (i.e. magnetic tape and card reader). The LISTA\$ operator is shown below.

LABEL → LISTA\$ Δ file code, # of buffers, logical address, # words/base, # words/base, etc.

# of buffers - the number of buffers described as # words/base in this operator.

Logical address - must be present (zero for magnetic tape and card operation).

# of words/base - describe each core buffer relative to programs lower lock setting. The number of buffers is not limited.

The packet generated is:

LABEL →

FILE CODE	# OF BUFFERS
LOGICAL ADDRESS	
	# OF WORDS
	BUFFER BASE
	# WORDS
	BUFFER BASE

### LISTB\$

The LISTB\$ operator is used only when executing the MREAD\$ operator on random access storage. The LISTB\$ operator is shown below.

LABEL → LISTB\$Δ file code, # of buffers, # words/base/logical address, # words/base/locigal address, etc.

The parameter packet generated is as follows:

FILE CODE	# OF BUFFERS
	# WORDS
	BUFFER BASE
LOGICAL ADDRESS	
	# WORDS
	BUFFER BASE
LOGICAL ADDRESS	

### 3.5.3 Status Codes

The status of each input/output operation is indicated, upon completion, in the A-register when control is returned to the worker program. Supplementary information may also be contained in the "A" and "Q" register. Upon successful completion, the "A" register is set positive on normal operations and may contain the logical address of a find when a search of random access storage was specified. The Q register contains # of words transferred.

All abnormal status conditions are signaled by 2<sup>29</sup>th of "A" register set to (1) with a six bit in the lower of indicating type of condition. Frame count errors on magnetic tape will be indicated by 2<sup>29</sup> of the A register = 0 and 2<sup>0-25</sup> equal to the magnitude of the frame count error.

- Inappropriate function (01) - The function code is not applicable to the file (e.g. READ on a printer file or SEARCHP on a tape file).
- Incorrect Parameter (02) - Buffer outside program lock limits, or illegal function code.
- Unrecoverable Error (03) - The requested function cannot be completed. Recovery procedures have been attempted and have proved unsuccessful. Parity errors, sequence errors, etc. fall in this category.
- End-of-file (04) - An end-of-file mark has been detected on magnetic tape or the end of allocated area has been reached on a random file read.
- End-of-tape (05) - A write operation has been successfully completed beyond the tape limit mark, or load point has been encountered during a READB operation.
- Unsuccessful Search (06) - The search identifier could not be located within the specified area of the file.
- Illegal character (07) - An illegal combination of punches (any combination of holes in rows 1, 2, 3, 4, 5, 6, 7 or 9) has been detected in a card column (translate mode only). The illegal character will appear as a 00 code in the buffer.
- No assignment (10) - No assignment mode for the referenced file code.
- Interlock (11) - Operator has indicated that an interlock condition cannot be corrected.

#### 4.0 Core Allocation

Descriptions of available core storage are held in chains utilizing free storage described to maintain its links. Normally the only requestors eligible to reference core chains are Omega and RT/comm control.

Three classes of chains are maintained by the system. Chain 1 holds free-storage, all non-committed core, and is used for program allocation, forming of other chains and to accommodate dedicated chain overflow. The second class are chain numbers two through five and are reserved for exec usage. Chain two contains unused storage modules, chain three activity addendums and chains 3-5 are unassigned.

The third group of chains are declared by and dedicated to the RT/comm control program; used to allocate core storage for worker programs, buffers or data-pass areas. Normally chain declaration is made during the initialization of the RT/comm control program committing an optimum area to the function. The system, upon dedication, will allocate a continuous area of core to the request. Expansion of a particular chain will be performed dynamically from free storage by the system upon sensing or unsatisfied request. All expansions will be returned to the general pool upon their release by the user. During peak periods of processing the RT/comm program may establish an additional chain at the expense of lower priority programs to accommodate overflow.

Chain declaration - each chain declared by the user must be assigned a number greater than 5 used to perform all references. The user has the option of declaring the committed area into two types of chain regulated mainly by intended usage. The first is a fixed module chain where each link is a given size; providing faster request/release mechanism as well as one less parameter at usage time. The second type provides for variable requests a parameter supplied at request time; giving the user more flexibility and promoting less wastage.

All allocations, requests or releases must be made in multiples of 2 words

FCHAIN\$ V0, V1, V2

where V0 - chain # assigned 5 - N

V1 - contains # of words declared if variable chain or # of words in each module if fixed chain.

V2 - # of modules if fixed chain 0 if variable chain.

Packet	ENT * B7	V0
	ENT * Q	V1
	ENT * A	V2
	7 7 3 4 0	2 0 2 1 4

Upon return of control A register will contain status of request 2<sup>29th</sup> set indicates core not available.

Chain release - used by RT/comm control to deallocate a previously declared chain. Caution must be used not to release chain before all expansions have been returned to the general storage pool.

RCHAIN\$ VO

where VO - # assigned to the chain by FCHAIN operator

Packet	ENT * B7	VO
	7 7 3 4 0	2 0 2 1 5

Memory request - from a previously established chain, a given number of words or a module in the fixed chain case.

MEMADD\$ VO, V1

where VO - # assigned to chain by FCHAIN operator

V1 - number of words requested when a variable chain is being referenced.

packet	ENT * B7	VO
	ENT * Q	V1
	7 7 3 4 0	4 0 0 0 1

Control is returned following packet with address of requested core in the "A" register relative to lower lock setting of requestor. If core not available in the specified chain or free storage, 2<sup>29</sup>th of the "A" register will be set. "Q" register contains # of words or modules remaining in the chain.

Memory release - requests the release of described core storage to the indicated chain.

MEMREL\$ VO, V1, V2

where VO - # assigned to chain

V1 - base address of core relative to lower lock of submitter

V2 - number of words being released applicable only to variable chains

Packet	ENT * B7	VO
	ENT * A	V1
	ENT * Q	V2
	7 7 3 4 0	4 0 0 0 2

Core Chains - are linked from the task addendum of the user to accomodate multiple RT/comm programs. For each declared chain the following description is formed to control usage.

Word	0	Chain #	Type of chain	← Pointer from addendum
	1		Link to next descriptor	
	2	Current #oflinks	Size of link	
	3		Base of core assignment	
	4		End of core assignment	
	5		Chain address	

Word 0 - contains chain number assigned by FCHAIN operator and type of chain 0 - indicates variable, 1 - indicates fixed.

Word 1 - link to next chain descriptor 0-0 indicates end.

Word 2 - contains number of modules free when chain is fixed. This value is returned to the user (in Q register) upon each request, thereby, providing a mechanism to sense depletion for time critical usage. Size of link is used for fixed chains when retrieving a module from free storage to accommodate overflow. If chain is variable, word 2 contains number of words currently available and is returned in Q register for each request.

Word 3-4 describes the core area committed to the chain.

Word 5 - address of first link in chain.

Chain links - each link in a chain describes its area and points the next link.

Word 0	Address of next link
1	# of words

Word 0 - is a 17 bit address relative to base of machine of next link in chain.

Word 1 - applicable to variable chains only and contains # of words in link.



## 5.0 Task Control Functions

Entry: Miscellaneous functions are entered from a table at label AMISC. A jump is performed using the lower four bits of the EXEC call to determine the requested function. Illegal parameters are referred to a common error routine. Miscellaneous functions currently available for use as follows:

### PUSH

A storage module has been formed and linked to current activity by EXRN routine. The PUSH mnemonic requests current activity to be linked to chain specified by  $V\emptyset$  by priority in ascending order. No control is returned until a complementary POP is given. This function is normally used by I/O control, content supervisor, etc. to queue and inactivate activity until requested service can be performed.

PUSH\$  $\Delta V\emptyset$

Packet	ENT*B7	$V\emptyset$
	EXRN	$\emptyset \emptyset \emptyset \emptyset 1$

Where  $V\emptyset$  is the base core address in absolute of linking cell of chain on which current activity is to be linked.

### QREF

Queue activity reference - mechanism for linking requests to activities previously defined by REGQ. A storage module is formed by this request and the parameters in registers A, Q, B1-B6 at the time of reference are stored in the SMOD for activation at execution. If this is the first QREF associated with the registered activity the activity is placed on the queue. If not the first it is linked to activity addendum for later execution of the registered activity. In any case, control is returned to the requestor.

QREF\$  $\Delta V\emptyset$

Packet	ENT*B7	$V\emptyset$
	EXRN	$\emptyset \emptyset \emptyset \emptyset 2$

Where  $V\emptyset$  is the binary identity of the activity referenced by the QREF\$. Used to perform a search on available activities within the task and identify the necessary activity addendum. Binary identity that is specified must not be  $\emptyset$  ( $V\emptyset \neq \emptyset$ ).

• RETURN

This entry is used to relinquish control to the executive at the conclusion of an asynchronous activity or task. The routine checks if referenced activity is task-permanent (i.e. it has binary identity). If it is task permanent, the SMOD just completed is deallocated and removed from chain associated with this activity. If there are any other SMODs linked to this activity they are given control. This activity addendum is not deallocated until conclusion of the task.

If this is not a task-permanent activity, it is checked to see if it is a fork from some other activity. If so, the activity just completed is deallocated and control given to forking activity providing a join has been given and that activity has no other outstanding forks.

If this is not fork from another activity, it is possible that the requestor may have no outstanding business which would cause reactivation (with the exception of queued activity registration), then the associated task is checked. If there exist no outstanding processable activities, forks, incomplete hardware level I/O requests or latent time of day restart requests, the task is terminated. Subsequent to task termination, facilities are deallocated and the next job task is sequenced or the job is terminated if no tasks remain. Otherwise, control is switched to some other task/activity active in the multi-program environment. No parameters are required for RETURN operator.

RETURN\$

Packet 

EXRN	Ø Ø Ø Ø 5
------	-----------

• FORMATOR

This is entered by a jump to ASCAN1. This routine receives command words through the storage module from the requestor and formats a channel executor module which is then queued to the appropriate CCB for subsequent execution of the input-output commands.

EXRN	Ø Ø Ø Ø 6
------	-----------

• DELAY

The Delay function is a way that task and task fragments can have themselves reactivated after a specified millisecond delay. The millisecond delay counter may be from 1 millisecond to a maximum delay of 24 hours. If the millisecond delay counter is over 5 minutes the time delay will be established as a time to activate and will be placed on the day clock timer routine and activation will be done on the gross timer of the day clock with a plus or minus variation of 6 seconds. If the delay is under 5 minutes it will be established on the real time clock queue with a plus or minus variation of 200 microseconds. The millisecond delay counter will be held in the Q register at the time of the Exec return.

DELAY\$

ENT.Q.VØ  
EXRN.00007

where VØ is the millisecond delay counter and may be in the form of W(X).

It should be noted that the delaying routine will only be placed on the C.P. upon completion of the delay time. Subsequent control could be further delayed to process higher priority activities.

Control will be returned to the instruction immediately following the EXEC RETURN with all registers restored with the exception of Q. Q will contain the difference between the actual time the routine wanted control and the time the switch to the routine was performed.

## 5.1 Fragmentation Requests

**Definition:** A fragmentation request (activity registration, fork or join) establishes an independently executable program. The request implicitly requires allocation of additional core to serve as the addendum necessary to execute the requested fragment. Once established, an activity or fork may make the same service requests as a task and will share with the task operational identity, primary cooperative streams, facility allocation, logging and accounting.

Standard Activity Registration defines a re-entrant activity to be registered with C.P. control for execution. The values of A, Q, B1-B7 are set to that of the requestor with B7 containing the address of the packet. Parameters specify the location, relative priority, data area, and relative index to be associated with the activity. After queueing control is returned to registering activity. The activity is initiated with all operational registers set to that of requestor in 15-bit mode.

REG\$ΔV0, V1, V2, V3, V4, V5, V6

Packet	EBJP*B7					\$+3		
	29		V1			15	14	0
	29	28	27	V5		15	14	0
	V4	V3	V6			V2		
EXRN					∅ ∅ ∅ 1 ∅			

V∅ - Address of activity in memory; this is implicit starting point of activity. If independently compiled program (V4=∅) this address must be a multiple of 1∅∅.

V1 - 15-bit address of data area in core.

V2 - 15-bit length of data area.

V3 - Data area mode

∅-indicates read/write lockin will be set to the data area defined by V1 and V2 and read will be permitted from any area.

1-indicates use of read/write lock of requestor.

The only means of reaching the data area is through use of registers B4-B7.

V4 - Activity mode indicator; zero specifies the activity is an entity and contains all referenced data exclusive of the declared data area and was independently compiled so that the first instruction is relative to address zero. Non-zero mode indicates that the

activity is integral to the request and RIR is not adjusted address of activity.

V5 - relative response priority ( $\emptyset$ -17) may be declared to attain a differentiation with respect to other activities currently operated within the task. When priority is not specified, the priority of the requestor is assumed.

V6 -  $\emptyset$  - use priority of previous, none specified  
 1 - priority has been specified

- Queue Processing Activity - a means of utilizing a task permanent activity to respond to a series of events. Transactions are accepted and queued by the system and the activity is executed whenever a queue entry exists. The activity signals completion for a given transaction by return of control via return operator. Omega re-executes the activity if any other SMODs remain linked to queued activity for execution.

After registration via a REGQ operator the activity becomes task permanent and can be referenced by a QREF any time before termination of the task. Use of this function allows the scheduling of events at occurrence. It is appropriate where no advantage can be gained from registration of concurrent executions by re-entrant code. Two operators are associated with use of this function, REGQ and QREF. The first defines the activity and the second supplies the data to the registers to be queued.

REGQ\$ $\Delta$ V $\emptyset$ , V1, V2, V3, V4

Packet	EBJP*B3					\$+3	
	29	V1			15	14	0
	29	28	27	26	15	14	0
	X	X	V4	V3		V2	
	EXRN					0	0 0 1 2

V $\emptyset$  - 15-bit address and starting point of activity.

V1 - 15-bit length of activity, zero implies read/write lock will remain set to that of requesting activity and the activity is considered an integral part of the compiler requesting activity. Non-zero length defines the area to be protected by memory lock-in and RIR is set to V $\emptyset$ .

V2 - Binary Identity of routine for further reference via the QREF operator. V $\emptyset$  =  $\emptyset$

V3 - Same as V5 for standard activity. (priority)

V4 - Same as V6 for standard activity.

- Fork - represents a method of activity registration in which the executive will correlate all forks from a given level so that completion of the synchronous activities from any level can be tested through the use of the JOIN function. All FORK activities are considered integral to the requestor and attain the same RIR and lock-in values of the requestor. At time of activation, forked activity will contain operational registers of requestor.

FORK\$ $\Delta$ V $\emptyset$

Packet	29	15	14	0
	ENT*B7		V $\emptyset$	
	EXRN		0 0 0 1 3	

V $\emptyset$  - 15-bit starting address of activity.

The FORK operator is more applicable to general batch processing programs than real time transaction processing since it provides a higher level interface and synchronization is on a gross basis. The only parameter required is a start address. An activity established by a fork may, in turn, establish other forks which provide additional levels of controlling parallel paths.

- A JOIN entry requests a wait until all asynchronous activities previously established by FORK have been completed as indicated by relinquish operator RETURN. The JOIN mnemonic sets a join bit in the activity addendum that remains set until the last forked activity from that activity has given the RETURN. Then the fork count goes to zero and control is returned to the original activity. A JOIN given in the task itself will wait on all forks outstanding within the task since the task is the base of all forking. A join from an activity will wait not only on those forks directly established by the activity, but also those forks set up by forks that are themselves direct forks from the requestor. No parameters are required for this operator.

JOIN\$

Packet	EXRN	0 0 0 1 4
--------	------	-----------

## 5.2 Restricted Task Control Functions

The restricted task control functions are those used by the OMEGA system and are not available to the general user.

- Register Control Thread

This function allows any part of the OMEGA system to establish a path starting at any of the primary OMEGA functions that acts in the same manner as a normal OMEGA call to the primary routine. Parameters needed are a 15 bit value to be stored in IFR lower to act as the OMEGA call and a JOB number of the task addendum to which the formed activity addendum must be linked. IFR value will be in register B1 and JOB number B2.

EXEC RETURN format is

EXRN.00020

The task control routine upon receipt of this function will form an activity addendum and a storage module with the value in B1 stored in IFR lower. This storage module will be the one the primary routine will process. One additional storage module will be formed to activate control at the primary OMEGA level. Next the task addendum whose JOB number matches with the value of B2 will be found and the activity addendum will be linked to it. Then the activity will be queued on the C.P. queue and control returned to the requestor.

- Abort Control Thread

This function allows any part of the OMEGA system to have removed from control the activity addendum it is presently running under. The activity addendum and all storage modules are placed back in the free core chain. No return control is allowed and no parameters are needed.

EXEC RETURN format is

EXRN.00021

- Special QREF

The special QREF is the same mechanism as the normal QREF with the exception that JOB number is also supplied. The correct task addendum will be found before the normal QREF is performed. Register B7 contains an address of a one word packet of information.

Packet format is

29	15	14	0
JOB #	BINARY IDENT.		

ENT.B7.Packet Address  
EXRN.00021

## 6.0 Secondary Exec Functions

Secondary Exec Elements are those required to load, activate, and terminate tasks introduced into the system via the I/O cooperative mechanism. Due to low usage of these functions in relation to the total system, the majority of routines are based on random access storage and loaded into core when the need arises.

### 6.1 Content Supervisor

The content supervisor is a resident exec routine responsible for loading and/or activation of drum based secondary EXEC functions used in performing service requests. Lib # of EXEC return instruction specifies service routine to activate and indicates if the service routine is re-entrant, allowing multiple simultaneous usage of routine, or non re-entrant requiring the content supervisor to control usage of the routine.

Each secondary Exec service routine is limited to 2,000g words in length including segmentation and buffering. Dependent upon core storage available and usage of routine, a copy of the service routine will be allowed to remain in core after completing a request and will be utilized to perform subsequent requests.

The following is a list of service routines which operate under control of the content supervisor and the service requests which activate them.

- Content Supervisor, library number 000. The following functions are processed directly by content supervisor, upon activation by a Direct Return (DRET1\$) from a secondary exec routine or on exec return in the case of the Purge request.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
00 B1=0	Internal	Release routine in IFR and return control to next storage module in string.
00 B1=1	Internal	Release routine in IFR and call routine listed in B2, B4 is set to original value of storage module causing activation.
00 B1=2	Internal	Release routine IFR. Deallocate the control thread of the current activity and release control to the exec.
01	Internal	Purge the highest order secondary exec routine in core, but not currently active.
02	Internal	Purge all inactive secondary exec routines currently in core.



03 Internal

Load and modify to running form the indicated element into the assigned core. B7 equals the absolute value of the following packet.

FC		LENGTH
# SEG		CORE BASE
FILE INCREMENT		

FC - The file within which the element may be found.

LENGTH - The length of the elements control portion or the maximum core used.

CORE BASE - The absolute address at which the element is to be loaded.

# SEG - The number of segments contained within the element. (2<sup>29</sup> - 2<sup>24</sup>)

FILE INC. - The increment relative to the base of the file containing the element at which the instruction of the control part start.

Functions 01, 02, 03 are requested by an EXRN\*2000X. Control is returned immediately following the EXRN instruction.

<u>Library Number</u>	<u>Name</u>	<u>Description</u>
001	Service Functions 1	Re-entrant routine responsible for high priority service requests.
002	Service Functions 2	Re-entrant routine responsible for processing high priority service requests.
003	I/O Error Recovery	Re-entrant routine responsible for input/output error recover for tape hardware handlers.
004	Console Control	Non re-entrant routine used to analyze and switch console operator messages.
005	Facility and storage assignment	Non re-entrant routine to process Assign and Release requests for mass storage and peripheral devices.
006	Service Functions 3	Non re-entrant routine to process search or form type requests.
007	Cooperative Service Routine	Non re-entrant routine to process service requests in the control of Unit record routines and I/O Cooperative Control.
010	Pre-Selection	Non re-entrant routine used to summarize scheduling data for task/activity.
011	Selection	Non re-entrant routine used to select and initiate task/activity.
012	Termination	Non re-entrant routine to deallocate and close task/activity routines and/or control threads.
013	Remote facility	Non re-entrant routine responsible for the assignment of remote communication devices and the load and activation of required remotes.
014	Library Service Routine	Non re-entrant routine responsible for transferring elements produced by generators, compilers, assemblers, and loaders to the job library.
015	Checkpoint and Restart	Un-defined

<u>Library Name</u>	<u>Name</u>	<u>Description</u>
016	Compaction	Responsible for compaction and purge of core and/or drum storage.
017	Dump	
020	I/O Error Recovery 2	Mass storage
021	I/O Error Recovery 3	Unit record peripherals
022	Phase 1 Rexecutor	Load REX oriented batch programs.
023	Phase 2 Rexecutor	Execute REX oriented batch programs.

### 6.1.1 Method of Operation

Content supervisor is activated upon an exec or worker task/activity executing an EXEC return instruction with two (2) exec call in bits positions 12-14. Upon activation B4 contains the address of storage module causing activation and performs the following functions.

- Unstring Library number, bit position 5-11, and function code, bit positions 0-4, contained in IFR of storage module. Library number and function code are entered into B registers 1 and 2 respectively.
- Validates library number and determines if service request is to be processed by core supervisor or drum based service routine.
- Content supervisor has two type of requests which are directed to it for processing: one to purge all routines not currently active, the other is the mechanism for returning control to the content supervisor by a secondary exec function and is performed by the following packet:

ENT*Q*	2
ENT*A*	12
7 7 3 4 0	0 0 0 0 0

Switch function code  
# of routine to switch to

When this packet is utilized, no storage module is allocated and control is returned to content supervisor. Upon receiving control C.S. will release secondary exec routine listed in IFR of previous storage module and then analyze B1 for subsequent action.

B1 = 0) Control will be returned to previous storage module

B1 = 1) Use B2 as the function code and Lib # of another secondary exec activity to schedule and run under previous storage module. B2 will be plugged into IFR.

B1 = 2) Release the routine in IFR of current storage module, deallocate the control thread of the current activity and release control to the exec.

- Calls for drum based secondary exec functions to satisfy service requests are handled as follows: (See accompanying block chart and table).

The called routine is located in the table and disposed by either queueing request for later activation or activating the called for routine. The content supervisor, upon activation of secondary exec functions, sets RIR to address of instructions, IFR to 17 bit B registers PLR to all of core and operational registers as follows.

B1 Library number of called service routine

B2 Function Code

B3 -----

B4 Address of storage module causing activation  
biased by lower lock limit

B5 The address of the core facility and storage summary  
biased by lower lock limit

B6 -----

B7 -----

KSSRIR - Increment within the storage module addressed  
by B4, contains the RIR of the secondary exec  
routine given control.

- The content supervisor allocates core for service routines in modules of 100g words up to a maximum size of 2,000g. Core storage is obtained from the general pool unless exhausted in which case one of the inactive service routines will be purged to obtain the core. Normally once a service routine is loaded it will remain in core until the storage is required. Inactive routines of a higher library number purged first.
- Secondary exec service routines are general non re-entrant code having full capabilities of segmentation and making additional service requests, I/O access as a normal activity. Their only constraints are as follows: size is limited to 2,000g; starting address is implicitly defined as their first location; they are self-initializing and have a special termination to the core supervisor. Each service request will be processed under task/activity performing request.

Content supervisor maintains the following list called "ETAB1" of routines operating under its control. Each entry in the list is ordered by Lib # and contains the following values.

Word 0	Current # of users	Lib # of routine
1	ZE	Length of routine
2		RIR of routine
3	Drum Increment to routine	
4		Queue Cell

Word 0 contains library number of routine; and current number of users, zero indicates routine is eligible for purge if core required.

Word 1 ZE is the file code of systems library used in drum call; length is the maximum size of routine.

Word 2 contains address of routine relative to base of machine and will be used as RIR setting during switch cycle and indicator

that routine is in core; zero indicates routine not in core.

Word 3 Contains drum increment of routine in systems library.

Word 4 Contains Queue cell to chain calls for routines currently active.

## 6.1.2 Control Transfer Functions Used In Secondary EXEC

### 1. Direct Return (DRET1\$)

ENT*Q*2
ENT*A*12
EXRN * $\emptyset$

Where used - All secondary EXEC routines

When used - When all responsibilities of routine have been completed.

Function - Will cause IFR and registers in LCR module to change and a transfer of control to the content supervisor via LCR. No storage module deallocation occurs.

Additional Parameters

1A:B<sub>1</sub> =  $\emptyset$  Release routine in IFR of current storage module and return control to next storage module in string.

1B:B<sub>1</sub> = 1 Release routine in IFR call routine whose library number and function code appear in B<sub>2</sub>.

1C:B<sub>1</sub> = 2 Release routine in IFR of current storage module, deallocate the control thread of the current activity and release control to EXEC.

### 2. Register Control Thread

ENT*B1*EXEC CALL
ENT*B2*JOB #
EXRN*REGCT

Where used - (Console handler, request for console control routine), (I/O cooperative, request for CSR). (RETURN, request for termination).

When used - When it is necessary to establish a control thread under another task addendum.

Function - Set up an activity addendum, LCR, and a storage module under the specified task addendum. B<sub>1</sub> is placed in IFR or LCR and storage module. Transfer control to routine in LCR, IFR location.

### 3. Switch Control Thread

ENT*A*JOB # & SWCT
ENT*Q*ACT. ADDM. ADDR.
EXRN*Ø

Where used - (CSR, from EXEC worker addendum before switch to pre-selection), (Pre-selection, worker to EXEC before switch to selection), (Selection, EXEC to worker before switch to initiation), (Termination, worker to EXEC before switch to selection).

When used - When it is necessary to switch from one addendum to another addendum.

Function - Remove the control thread for the current activity from its task addendum and link it to the indicated addendum.

### 4. Abort Control Thread

EXRN*ØØØ21

Where used - In Content Supervisor upon request of a secondary EXEC routine entry via DRET\$ with B<sub>1</sub> = 2. (Selective when no task can be selected.)

When used - When no further processing is possible and no point for return of control exists.

Function - Remove and discard the control thread from the current activity the release control to the dispatcher.

### 5. Direct Switch 1 (DSW1\$)

ENT*Q*ADDR RIR
ENT*A*00010
EXRN*00000

Where used - Content Supervisor

When used - When a secondary EXEC routine is given control after its availability is determined.

Function - Reset RIR and drum increment and transfer to address satisfied.



6. Direct Return 2 (DRET2\$)

ENT*A*13
EXRN*00000

Where used - Content Supervisor

When used - Upon a DRET1\$ from a secondary EXEC routine implying a return of control to the requestor.

Function - Deallocate last storage module on current activity and return control to it.

### 6.1.3 Secondary Exec Chain List

Within the Secondary Exec it is often necessary to utilize the services of other Secondary Exec routines. If this is done via a EXRN\*20XXX, the requesting routine will be locked out if it is not reentrant. Even if it is reentrant it will be required to remain in core at the same location until the requested service has been completed. When it is considered that the second level request may in turn cause requests to other routines it may result in a pyramid of routines necessary in core at one time to process a single request.

To alleviate this problem a chaining procedure is herein defined which allows a secondary exec routine to obtain the services of other secondary exec routines during which time it is released and may be used for another request or purged if the core is required.

It is necessary that any routine initiating a chain request be coded such that it may be purged and unloaded at a different location between the time that it makes a chained request and when it gets control returned from that request.

It is the responsibility of the routine initiating a chained request to obtain free core for the list and to set the list initially.

The content supervisor will update the list upon the entrance to and exit from the routines involved in the request.

When the initiating routine gets control back from the request the chain list will contain a list of all routines involved in the processing of the request and the status returned by each, along with a combined status which consists of the logical sum of all the returned status's.

•The requirements for the initiation of a chained request within the Secondary Exec are listed below.

- 1) Obtain free core sufficient to contain the list and all entries possible.
- 2) Set up the first 6 words of the list as shown in figure 1.
- 3) Set the sign bit in the IFR location of the current storage module and save the address of the list in KSCL of the storage module.

•The requirement of the initiator upon the completion of the request are as follows.

- 1) Release free core used by the list.

•The Content Supervisor will check for a chained request upon entrance from a Secondary Exec routine. If it is a chained request one of the following procedures determined by the type of exit from the Secondary Exec routine will be taken.

- 1) B1=0
  - a) Set the logical status and the individual status in the chain list, for the process just completed.
  - b) Clear the sign bit in the IFR location of the SMOD
  - c) Give control to the initiator using the return library number and function code (IFR) in the chain list.
- 2) B1=1
  - a) Perform status set as when B1=0.
  - b) Store call for routine in B2 in the chain list.
  - c) Process request for routine defined by B2.
- 3) B1=2
  - a) Set abort status in logical status word of the chain list.
  - b) Process as if B1=0

•The following is a coded routine which will form a chain list and do the initialization. The functions performed by this routine would be required for the initiation of a chained request.

FCLST            Form Chain List  
 Input - B4 = address of current SMOD  
           B1 = size of list required  
           B3 = return IFR setting desired.

Output- B7 = absolute base address of list  
           KSCL = address contained in B7  
           IFR = sign bit will be set in the lower.

Entrance        EBJP·B6·FCLIST

Exit            JP·B6

Registers Used Other Than Input/Output = A, Q

FCLST	CL*B7	→ Free core chain
	ENT*Q*B1	→ # words to request
	EXRN*MEMADD\$	→ Request core
	JP*FCLST4*ANEG	→ Didn't cut core
	ENT*B7*A	→ Address of list
	STR*B1*U(B7)	→ List length
	PUT*6*U(B7)	→ # words used
	ENT*A*LX(KSIFR+B4) ANEG	→ Currently in chained request
	SEL*SET*40000	→ No set sign bit
	STR*A*L(KSIFR+B4) APOS	→ Restore, nested request
	ENT*Q*W(KSCL+B4)	→ Pick up nested chain addr
	CL*Q	→ Clear nest link
	STR*Q*W(B7+4)	→ Set nest link
	STR*B7*W(KSCL+B4)	→ Set chain addr.
	STR*B3*U(B7+5)	→ Return IFR
	CL*L(B7+5)	→ Clear logical sum status
	JP*B6	→ Exit
FCLST4	EXRN*PURGEA\$	→ Purge unused routines
	JP*FCLST	→ Try again

SECONDARY EXEC CHAIN LIST

0	LIST LENGTH	# WORDS USED
1	ORIGINATOR	
2	STORAGE	
3	AREA	
4	ADDR PREVIOUS LIST	
5	RETURN IFR	COMBINED STATUS
6	REQUEST IFR	STATUS
7	REQUEST IFR	STATUS
.		
.		
.		
N		

Word  $\emptyset$

U - The length of the list as determined by the originator.

L - The number of words used in the list at the present time. This serves as an increment to the first free word in the list.

1-3 Storage area for use by the original requestor.

4 The address of a previous list in case the current list is a chain list originating within a chained request. Word 4 of the initial list will be  $\emptyset$ .

5 U - The IFR setting necessary to return control to the originator.

L - The combined status of all requests within the chain.

6 U - The IFR for the initial request made by the originator.

L - The status of the request.

7 U - the IFR of a request made by the routine requested by the originator or some other level routine.

STORAGE MODULE FOR CHAINED SECONDARY EXEC REQUEST

0	PUSH POP LINK		
1	ACT. ADDM. ADDR.		
2	1	IFR	KSIFR
3	RIR		
4	LOCK LIMITS		
5	P	B1	
6		B2	
7		B3	
10		B4	
11		B5	
12		B6	
13		B7	
14	A		
15	Q		
16	SECONDARY EXEC ROUTINE RIR		
17		ADDR CHAIN CELL	KSCL

Figure 2

The routine which initiates a chained request would set the sign bit in the lower of the IFR (Word 2) in the storage module. This would signal the content supervisor to update the chain list upon exit from a servicing routine.

If the sign bit in the lower of Word 2 is set, word 17 (KSCL) is assigned to be the chain link. Therefore, if the request is chained the KSCL location should not be destroyed in processing.

6.2 Service Functions (1) Library number 001 is a re-entrant service routine responsible for the following service requests.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	LOAD\$	Load indicated segment
02	LOADA\$	Load and activate indicated segment
03	UST\$	Unstring specified statement
04	CVT\$	Convert specified numeric
<u>05-37</u>	<u>Unassigned</u>	

Service Functions (1) Library number 001 is a re-entrant service routine responsible for the following service requests.

•LOAD A SEGMENT (Function code 01)

This request will load a segment into core from random access storage. This is an unconditional load directive, i.e., the segment will be loaded regardless of whether it is still resident in core from a previous load operation or not. The LOAD\$ operator is the direct method for having a program loaded into core. The indirect method is to jump or return jump to some externally defined label within the segment. If the segment is still resident in core, it will not be reloaded; thus a segment should be self-initializing if the indirect method of loading is used. If the segment is not in core, it will be loaded from random access storage and the jump or return jump performed.

This request will load the segment that contains the externally defined label reference found in the lower 15 bits of B7.

Operator: LOAD\$ALABEL

Packet:	ENT*B7	LABEL
	77540	20041

•LOAD A SEGMENT AND ACTIVATE (Function code 02)

This request is similar to the LOAD\$ operator. The segment containing the externally defined label found in the lower 15 bits of B7 will be loaded into core from random access storage. This is a unconditional load: the segment will be loaded regardless of whether the segment is resident in core from a previous load or not. After loading, control will be passed to the referenced label, i.e., a JP\*LABEL instruction will be performed.

Operator: LOADA\$ALABEL

Packet:	ENT*B7	LABEL
	77540	20042

•FIELDATA TO BINARY CONVERSION (Function code 04)

This request will convert a number from fieldata to binary. The fieldata number can either be octal or decimal. It must be positive. The largest octal number that can be handled is 7777777777; the largest possible decimal number is 536870911 ( $2^{29} - 1$ ). The fieldata number must be an integer. The resultant binary number will be positive and single precision (contained in one word). The fieldata number can be contained in one or more words of core. There can be leading and trailing fieldata spaces (05) or binary zeroes (00). While only leading spaces or zeroes will be allowed between



the left-hand end of the field and the most significant digit, the conversion routine will ignore anything following the first trailing fielddata space or binary zero. The core address of the left-hand end of the fielddata number field will be in lower 15 bits of B7. The A register will contain the number of words in the field. The following convention will be used to determine whether the fielddata number is octal or decimal. The first digit of an octal field will be a fielddata 0 (60); the first digit of a decimal field will be non-zero (61-71). If the conversion is successful, the binary number will be placed in the Q register and the A register set to 0. If an error is detected, the Q register will be set to 0 and a flag bit set in the A register.

A REGISTER FLAG

MEANING

1 ( $2^0$ bit)	Non-octal character (character other than 60-67) found in octal field.
2 ( $2^1$ bit)	Octal number greater than 7777777777
4 ( $2^2$ bit)	Non-decimal character (character other than 60-71) found in decimal field.
8 ( $2^3$ bit)	Decimal number greater than 536870911
16 ( $2^4$ bit)	No number found in field (field consists entirely of fielddata spaces or binary zeroes).
32 ( $2^5$ bit)	Number of words in field = 0.

Operator: CVT\$AV0,V1

where V0 = the address of left-hand end of fielddata field; and

V1 = size of fielddata field

Packet:

ENT*B7	LABEL
ENT*A	SIZE
77540	20044

Function Code 03 - Unstring the control information specified and form an unstringed list of the fields and the controls separating the fields.

Caller: Internal call from operating program via UST operator.

Parameters: B4 = Address of storage module causing activation biased by lower lock limit.

UST Operator: UST $\Delta$  V<sub>0</sub>, V<sub>1</sub>, V<sub>2</sub>, V<sub>3</sub>

V<sub>0</sub> = Base address of the control information relative to lower lock of the requestor.

V<sub>1</sub> = Base address of the deposit area to be used for the unstringed information relative to lower lock of the requestor.

V<sub>2</sub> = Length of the deposit area. If control information specified is a continuation of previous information, the sign bit should be set.

V<sub>3</sub> = Length of the pick up area to signal the end of the information when no terminating control is present. If V<sub>3</sub> =  $\emptyset$  data is unstringed until a terminating control is found. If V<sub>3</sub> is negative (the sign bit set), interim spaces are to be left in.

Packet:

ENT*B6	V0
ENT*B7	V1
ENT*A	V2
ENT*Q	V3
EXRN	00003

Addendum: Worker task/activity addendum

Function: . Unstring the control information in the following format:

FIELD 1 DESC  
FIELD 2 DESC  
FIELD 3 DESC  
FIELD 4 DESC  
ETC.

OPTIONS IN MASTER BIT FORM		
29	24-23	15-14
TC	# WORDS	IND TO FIELD

TC = terminating character for field.

# words = number of words in field.

Ind to field = index to field from base of unstringed area.

Exits:

- . Return to requestor:
  - "A" Register = 00000 00000 normal completion
  - "A" Register = 40000 00000 overflow of deposit area
  - "A" Register = 40000 00001 deposit area not within lock limits.
  - "A" Register = 40000 00002 invalid option
  - "Q" Register = number of fields if normal completion
- . Return control to content supervisor

6.3 Service Functions (2) Library number 002 is a re-entrant service routine responsible for the following service requests.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	TIMED\$	Supply edited time of day
02	TIMEL\$	Supply elapsed time for task
03	DATIM\$	Supply current date and time
04	XOFF\$	Set logical switches off
05	XON\$	Set logical switches on
06	XTEST\$	Supply current logical switch settings
07	TESTFOF\$	Floating point overflow test
10	TESTFUF\$	Floating point underflow test
11	TESTFL\$	Test floating point over/under flow
12	ERRADD\$	Establish error address
13	FOFADD\$	Establish floating point overflow address
14	FUFADD\$	Establish floating point underflow address
15	OPADD\$	Establish illegal operation address
16	TFC\$	Test file code for type of device
17	TCORE\$	Supply core limits of task
20	SET15\$	Set "B" registers to 15 bit mode
21	SET17\$	Set "B" register to 17 bit mode

Service Functions (2) Library number 002 is a re-entrant routine responsible for the following requests.

•TIME OF DAY (Function code 01)

This request will return the time of day at the time of the request. Hour, minute and second will be supplied. Hour and minute will be returned in the A register in the following format: HH : MM. The seconds will be returned in the Q register in the following format: ~~SS~~SS. Hour, minute, second and the colon (:) will all be binary zeros. All time will be decimal.

Operator: TIMED\$

Packet: 

7	7	5	4	0	2	0	1	0	1
---	---	---	---	---	---	---	---	---	---

•ELAPSED TIME (Function code 02)

This request will return the elapsed time for the task in the Q register. The time will be in binary and in units of 200 milliseconds.

Operator: TIMEL\$

Packet: 

7	7	5	4	0	2	0	1	0	2
---	---	---	---	---	---	---	---	---	---

•CURRENT DATE AND TIME (Function code 03)

This request will return the year and the day in the A register, in fielddata, in the following format: YYDDD, where YY is the year and DDD is the day (from 001 to 365). The Q register will contain the time of day in units of 200 milliseconds. The Q register will be in binary.

Operator: DATIM\$

Packet: 

7	7	5	4	0	2	0	1	0	3
---	---	---	---	---	---	---	---	---	---

•SET LOGICAL SWITCHES OFF (Function code 04)

This request will set the logical switches indicated in the Q register to 0 (off). The Q register will contain the names of the logical switches (A-E), in any order, in fielddata, that are to be turned off (set to 0).

Operator: XOFF\$ switches

Packet: 

ENT*Q*W(\$+1)*SKIP									
fielddata switches									
7	7	5	4	0	2	0	1	0	4

•SET LOGICAL SWITCHES ON (Function code 05)

This request will set the logical switches indicated in the Q register to 1 (on). The Q register will contain the names of the logical switches (from A-E), in any order, in fielddata, that are to be turned on (set to 1)

Operator: XON\$ switches

Packet:

ENT*Q*W(\$+1)*SKIP	
fielddata switches	
7 7 5 4 0	2 0 1 0 5

•SUPPLY CURRENT LOGICAL SWITCHES (Function code 06)

This request will supply the current logical switch settings in bits 25-29 of the A register as follows:

<u>A Register</u>	<u>Logical Switch</u>
29	A
28	B
27	C
26	D
25	E

If the bit for a particular switch is 1, the switch was on; if it is 0, the switch was off. As an example, if switches B, D and E are on and switches A and C are off, bits 25-29 of the A register will be as follows: 01011, where bit 29 is the left hand bit. Bits 0-24 will be cleared to 0.

Operator: XTEST\$

Packet:

7 7 5 4 0	2 0 1 0 6
-----------	-----------

•FLOATING POINT OVERFLOW TEST (Function code 07)

This request tests the floating point overflow switch. If it is off, the A register is set to 0. If it is on, the switch is turned off and the A register is set to 1.

Operator: TESTFOF\$

Packet:

7 7 5 4 0	2 0 1 1 0
-----------	-----------

•FLOATING POINT UNDERFLOW (Function code 10)

This request tests the floating point underflow switch. If it is off, the A register is set to 0. If it is on, the switch is turned off and the A register is set to 1.

Operator: TESTFUF\$

Packet: 

7	7	5	4	0	2	0	1	1	0
---	---	---	---	---	---	---	---	---	---

•TEST FOR FLOATING POINT ERROR (Function code 11)

This request essentially combines the tests for floating point overflow and floating point underflow. The switches will be tested in the following order: floating point overflow; floating point underflow. If neither the floating point overflow nor underflow switch is set, the A register is set to 0. If the floating point overflow switch is set, the floating point overflow and underflow switches are turned off (floating point underflow switch is turned off automatically in the event that one of the two switches was set from a previous floating point error condition that was not tested; if the floating point underflow switch was not on, nothing is, in effect, done) and the A register is set to 1. If the overflow switch is off and the underflow switch on, the underflow switch is turned off and the A register is set to -1 (777777776).

Operator: TESTFL\$

Packet: 

7	7	5	4	0	2	0	1	1	1
---	---	---	---	---	---	---	---	---	---

•ESTABLISH ERROR ADDRESS (Function code 12)

This request takes the error address in B7 and places it in the task addendum. Address is relative lower lock of requestor. Control can be transferred to this address in the event of a program error, e.g. memory references outside memory bounds, attempting to use privileged instructions. When operating with the TEST PACKAGE this function is reserved for its use.

Operator: ERRADD\$ATag

Packet: 

ENT*B7	LABEL								
7	7	5	4	0	2	0	1	1	2

•ESTABLISH FLOATING POINT OVERFLOW ERROR ADDRESS (Function code 13)

This request takes the floating point overflow error address in B7 and places it in the task addendum. Address is relative to lower lock of requestor. Upon floating point overflow program control is transferred to specified address with interrupt values in A and Q registers.

Operator: FOFADD\$ATag

Packet: 

ENT*B7	LABEL								
-7	7	5	4	0	2	0	1	1	3

•ESTABLISH FLOATING POINT UNDERFLOW ERROR ADDRESS (Function code 14)

This request takes the floating point underflow error address in B7 and places it in the task addendum. Address is relative to lower lock of requestor. Upon floating point underflow program control is transferred to specified address with interrupt values in A and Q registers.

Operator: FUFADD\$ΔTag

Packet: 

ENT*B7	LABEL
7 7 5 4 0	2 0 1 1 3

•ESTABLISH ILLEGAL OPERATOR ADDRESS (Function code 15)

This request takes the illegal operator error address in B7 and places it in the task addendum. Address is relative to lower lock of requestor. Program control will be transferred to specified address when ever program executes an illegal instruction code. When operating under control of TEST PACKAGE function is reserved for its use.

Operator: OPADD\$ΔTag

Packet: 

ENT*B7	LABEL
7 7 5 4 0	2 0 1 1 5

•TEST FILE CODE FOR PERIPHERAL TYPE (Function code 16)

This requests the numeric peripheral type code of the unit or mass storage assigned to the file code contained in the Q register. File code is in its field data form.

Operator: TFC\$Δfile code

Packet: 

ENT*Q	File Code
7 7 5 4 0	2 0 1 1 6

Numeric peripheral type is returned in the "A" register right justified. If device is random access, the number of words allocated to file code will be returned in the Q register.

•REQUEST CORE SIZE (Function code 17)

This request will supply three pieces of information: the lower lock address (in A); the RIR (in Q) and the number of words, in binary, allocated to the program (in B7)

Operator: TCORE\$

Packet: 

7 7 5 4 0	2 0 1 1 7
-----------	-----------



•SET IFR 15 BIT B REGISTERS (Function code 20)

This request will set the B register mode bit in the IFR for 15 bit B registers. All normal worker programs are original established in this mode.

Operator: SET15\$

Packet: 

7	7	5	4	0	2	0	1	2	0
---	---	---	---	---	---	---	---	---	---

•SET IFR FOR 17 BIT B REGISTERS (Function code 21)

This request will set the B register mode bit in the IFR for 17 bit B registers which is the abnormal mode of operation reserved for EXEC, RT/COMM, etc.

Operator: SET17\$

Packet: 

7	7	5	4	0	2	0	1	2	1
---	---	---	---	---	---	---	---	---	---

## 6.4 I/O Error Recovery

## 6.5 Console Control

01	Unassigned	
02	Unassigned	
03	Unassigned	
04	Unassigned	
05	Unassigned	
06	Unassigned	
07	Unassigned	
10	CHANGE\$	Operator directives with regard to files
11	MOUNT\$	Operator directives with regard to files
12	UNMOUNT\$	Operator directives with regard to files
13	LOG\$	Submit systems information to log or console
14	MOUNTN	MOUNT without a delay
15	CHANGEN	CHANGE without a delay

•CHANGE FILE DEVICE (Function code 10)

This is a standard method of notifying the computer operator that a particular file device (e.g., a reel of tape) is to be labelled and unmounted and an alternate file mounted in its place. This operator is necessary for such communication since worker programs will not be allowed to use physical

channel/unit designations. Magnetic tapes will be positioned at the beginning of the tape. The requesting activity will be delayed until the computer operator signals that the file has been changed.

Operator: CHANGE\$AFile code, file identifiers

Packet:

EBJP*B7	N
FILE CODE	
FILE IDENTIFIERS	
77540	20050

variable length and format

N

•MOUNT A FILE (Function code 11)

This request directs the computer operator to mount an input-output file on a particular file device. This operator is necessary for such communication since worker programs will not be allowed to use channel/unit designations. Magnetic tapes will be positioned at the beginning of the tape. The requesting activity will be delayed until the computer operator signals that the file has been mounted.

Operator: MOUNT\$AFile code, file identifier

Packet:

EBJP*B7	N
FILE CODE	
FILE IDENTIFIER	
77540	20051

variable format and length

N

•DISMOUNT A FILE (Function code 12)

The computer operator will be requested to dismount an input-output file from a particular file device and properly label it. The task will be delayed until the computer operator signals that the file has been dismounted.

Operator: DISMOUNT\$AFile Code, file identifier

EBJP*B7	N
FILE CODE	
FILE IDENTIFIER	
77540	20052

variable length and format

N

\*LOG (Function code 13)

This is the standard method of sending a message to the computer operator and/or entering data into the systems log. If data is being entered into the systems log, it may also be typed on the computer operator's console. An optional delay after a message is typed out on the console can be requested to allow the computer operator to perform some action.

Operator: LOG\$ΔOptionsΔliteral

where "literal" is the message to the computer operator and/or data to the systems log.

The options are:

T = Type message on console, no delay

R = Type message on console, wait for operator response

L = Enter literal into systems log

A blank field is equivalent to the T option.

Packet:

EBJP*B7	N
OPTIONS & LITERAL	
77540	20053

variable length and format

N

When control is returned after a log request with R option the Q register may contain 1 to 5 characters as a response to the delay and are useable by the requestor.

.MOUNT A FILE - NO DELAY (Function code 14)

This request is almost identical to the MOUNT\$ operator. It directs the computer operator to mount an input-output file on a particular file device. However, no delay will be initiated. The task will be continued after the console typeout.

Operator: none

Packet:

EBJP*B7	N
FILE CODE	
FILE IDENTIFIER	
77540	20054

variable length and format

N

\*DISMOUNT A FILE - NO DELAY (Function code 15)

This request is similar to the DISMOUNT\$ operator. It directs the computer operator to dismount an input-output file and properly label it. No delay will be initiated. The task will be continued after the console typeout.

Operator: none

Packet:	EBJP*B7	N	
	FILE CODE		
	FILE IDENTIFIER		variable length and format
	77540	20055	N

\*CHANGE FILE DEVICE - NO DELAY (Function code 16)

This request is similar to the CHANGE\$ operator. It directs the computer operator to dismount a file, label it, and mount an alternate file in its place. No delay will be initiated. The task will be continued after the console typeout.

Operator: none

	EBJP*B7	N	
	FILE CODE		
	FILE IDENTIFIER		variable length and format
	77540	20056	N

6.6 Facility and Storage Assignment - Library number 005 is a non re-entrant routine responsible for processing the following service requests.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	ASG\$	Assign designated facility to task
02	FREE\$	Release designated facility from task
03	SWITCH\$	Switch units assigned to file codes
04	Internal	Release all non-hold assignments
05	Internal	Release all assignments
06	Internal	Assignments submitted by selection
07	Internal	Switch submitted by selection



## Functional Description of Function Codes

### Function Code 01 - Assignment Request

**Caller:** A task/activity requesting the assignment of a peripheral through the control statement interpreter or an internal request made at execution time.

**Parameter:** B7 = the address of the ASG statement relative to the lower lock of the requestor.

**Addendum:** Worker task/activity addendum

**Function:**

- a) Unstring the control statement
- b) Read in the facility map
- c) Locate the peripheral name requested
- d) Make an assignment from the peripherals listed under that name and build associated tables.
- e) Load the associated handler if not in core
- f) Initialize the unit as prescribed by options
- g) Set up log function for assignment printout
- h) Switch to log routine for printout and delay

**Exits:** DRET\$ to control supervisor, B1 =  $\emptyset$  B2 = console control.

**Returned**

**Status:** A = logical address if mass storage assignment or P-TYPE, CHAN/UNIT if unit assignment.  
A = 77777400XX implies unsuccessful assignment

### Function code 02 - Free request

**Caller:** A task/activity requesting the release of a peripheral through a control statement or an internal request.

**Parameters:** B7 = the address of the free statement relative to the lower lock of the requestor.

**Addendum:** Worker task/activity addendum

**Function:**

- a) Unstring the control statement
- b) Locate the mass storage list or UCB associated with the file code specified.
- c) Release the storage or units from core summary
- d) Release core used for random access storage list and UCB.
- e) Go to master file directory routine if the file is to be registered
- f) Prepare a log message for the console
- g) Switch to log routine for printout

Exits: DRET1\$ to content supervisor B1 = 1 B2 = console control

Returned

Status: A ≥ 0 implies successful completion of release  
A = 77777400XX implies abnormal condition such as  
the file code not being assigned.

#### Function Code 03 - Switch request

Caller: A task/activity requesting the switch of two  
file codes through a control statement or an  
internal request made at execution time.

Parameter: B7 = the address of the FREE statement relative  
to the lower lock of the requestor.

Addendum: Worker task/activity addendum

Function: a) Unstring control statement  
b) Locate file codes  
c) Switch the links to the unit control blocks  
under the individual file codes.

Exits: DRET1\$ B1 = 0 Return to requestor

Returned

Status: A ≥ 0 implies successful completion  
A = 7777740001 implies necessary parameters were  
not present  
A = 7777740002 implies an invalid file code was  
specified.

#### Function code 04 - Release all non-hold assignments

Caller: Termination upon the termination of a task.

Parameter: All parameters are available within the addendum  
which is currently activ.

Addendum: Worker task/activity addendum

Function: a) Make a pass on the file code list erasing  
UCB links of all units not in a hold condition.  
b) If unit is currently active delay until all  
requests are complete and then deallocate  
any random access storage list and UCB.  
c) Update accounting information in job description.  
d) Release units or storage to system.  
e) Release locks on released areas.

Exits: DRET1\$ B1 = 0 return to requestor

Function code 05 - Release all assignments

Caller: Termination routine upon job termination

Parameters: All parameters are available within the addendums which is currently active.

Addendum: Worker task/activity addendum

Function: Same as function code 04 except that all files assigned to the job are released.

Function code 06 - Assignment of a peripheral requested by initiation.

Caller: Initiation upon the processing of pre-load time facility requests.

Addendum: Worker task/activity addendum

Function: a) Same as function code 01 except a special entrance is used for entry into the log routine which will disregard any delay request and return to initiation with the status in the A register as before and the peripheral name under which the assignment was made.

Exits: DRET1\$ B1 = 1 B2 - console control FC XX

Returned Status: Same as Function code 1

Function code 07 - Switch of two file codes requested by initiation

Caller: Initiation upon the processing of pre-load time switch requests.

Addendum: Worker task/activity addendum

Function: Same as function code 03 except for a special exit

Exit: DRET1\$ B1 = 1 B2 = initiation (selection FC 05)

Returned Status: Same as function code 03.

### 6.6.1 Method of Operation

The ASG service request prepares a file for use by setting up a unit control block according to the specifications implied by the assign statement parameters and the particular device assigned.

The sequence of functions employed to process an assign control statement are as follows:

- a) Unstring ASG control statement
- b) Locate given peripheral code which will identify peripheral type desired and required device handler. Check availability of device or storage in "facility and storage summary" if available continue, otherwise exit.
- c) Reserve indicated device or storage in summary. Read in device descriptor and form unit control block setting options for peripheral and load, if not currently active, device handler. Set file code increment to unit control block and set RIR and Drum increment of device handler in unit control block.
- d) Perform operator type-outs directing tape mounting or retrieve file descriptors for random access storage. Exit upon completion of above.
- e) Maintain by Job # a list of all facilities assigned.

## ASSIGN Statement Format and Interpretation

### Format

1
2
3
4  
 # ASG  $\triangle$  Options  $\triangle$  Peripheral Name, File Code, Estimate, File Identification

### • Interpretation of Parameter Fields

<u>Peripheral</u>	<u>Meaningful Options</u>	<u>Required Parameters</u>	<u>Optional Parameters</u>
TAPE	E,O,H,M,L,J,R, S,T,V,W,U,N	1, 2	3, 4
MASS STORAGE	J,S,V,H,M,L	1, 2	3, 4
PRINTER	J,V,W	1, 2	3, 4
CARD	J,T,V,W	1, 2	3, 4
PAPER TAPE	J,V,W	1, 2	3, 4
CORE	J,S,V,W	1, 3	4
COMM			

### • Option Meaning

<u>Option Letter</u>	<u>Meaning</u>
E	- Even parity (compatible mode) applicable on compatible units only.
O	- Odd parity (Binary) assumed if no O or E option present.
H (TAPE)	- Highest density applicable to the subsystem (800 ppi)
(MASS STORAGE)	- High grade mass storage requested.
M (TAPE)	- Medium density applicable to the subsystem (556 ppi)
(MASS STORAGE)	- Medium grade mass storage requested.
L (TAPE)	- Lowest density applicable to the subsystem (200 ppi)
(MASS STORAGE)	- Low grade mass storage requested.
J	- Hold assignment for duration of job unless explicitly released. Absence of the J option implies assignment will be released upon termination of the task.
R	- Rewind the assigned unit without interlock.

Option LetterMeaning

- S - Declares the file is in the Master File Directory
- T - Translate from FD, to BCD coming in and from FD to BCD going out. (Applicable on compatible units with translate hardware.)
- V - Assignment not required for the execution of the task.
- W - Wait for operator response after file identifier is printed on the console printer.
- U - Indicates automatic block numbering is to be applied to any tape files written.
- N - Implies the noise record constant defined at system generation will be applied.

## . Peripheral Name Interpretation

The peripheral name is a mnemonic name of up to five characters in length which dictates a set of units acceptable for assignment on this request. A set of standard peripheral names are used by the system and any user. Additional peripheral names may be defined at system generation time. A list of the standard peripheral names and their names appears below.

TAPE	Any UNISERVO
UN3C	UNISERVO IIIC
UN6C	UNISERVO VIC
UN8C	UNISERVO VIIIC
CRIN	Card Reader other than primary input
CROUT	Card Punch other than secondary output
PTIN	Paper Tape Reader
PTOUT	Paper Tape Punch
PRINT	High Speed Printer other than primary output
RAN	Any random access device
FHAN	FASTRAND
FH432	FH432 Drum
FH880	FH880 Drum
SEQ	Any sequential file device (random access on tape)
CORE	Additional core adjacent to end of routine.

. File Code Interpretation

The file code is an alphabetic character by which the unit assigned will be referenced in input, output operations. The characters A through Y are available for general use. The character Z is reserved for the system files.

Each letter A-Y may be broken into subsets of 26 double file codes, for example A could be broken down into AA, AB,---AZ.

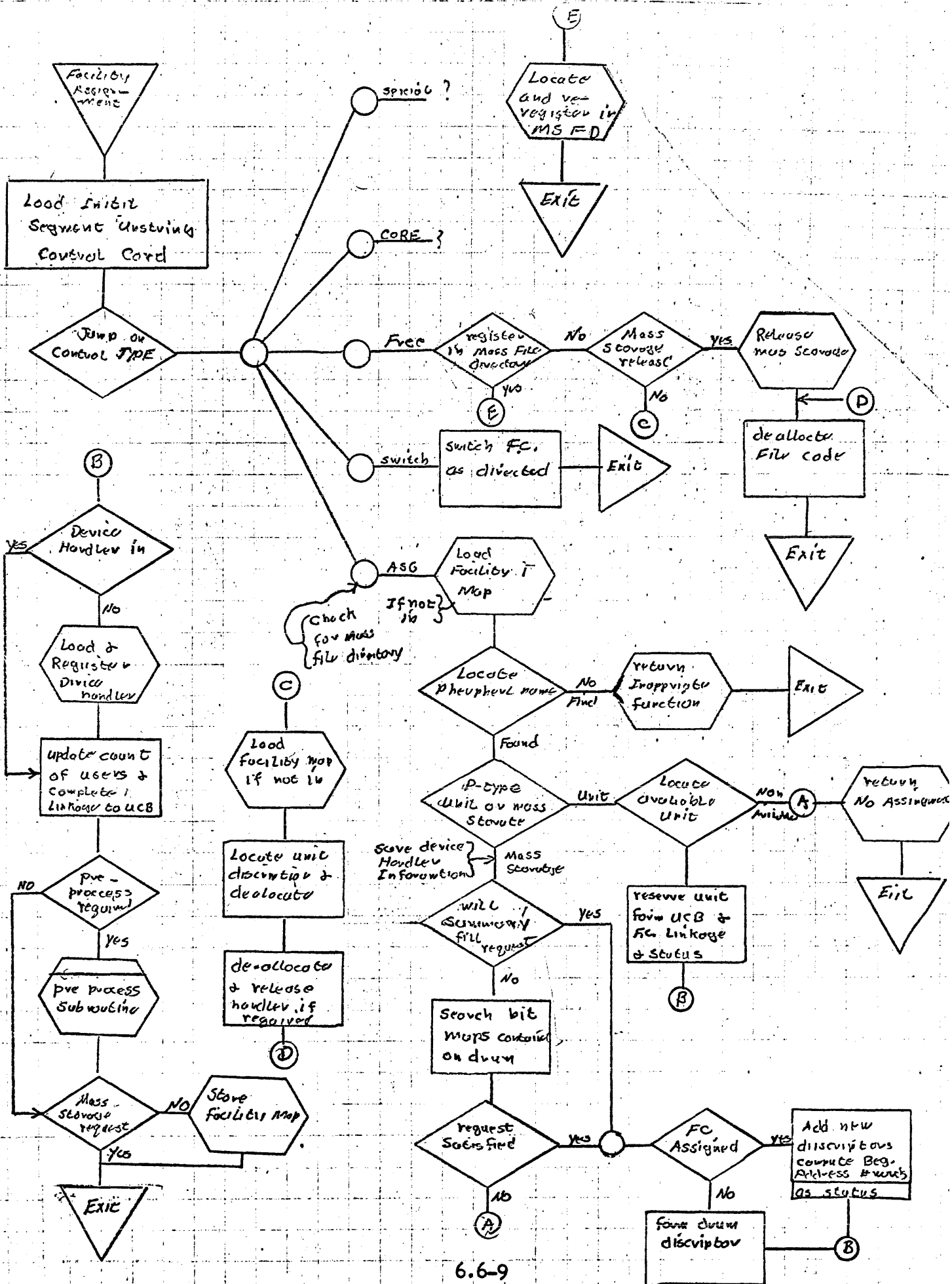
The system file code is broken down into a subset containing double file codes. The system file and their assigned file codes appear below.

ZA	Primary input unit record
ZB	Primary output unit record
ZC	Secondary output unit record
ZD	Cooperative library
ZE	Systems library
ZF	Job library
ZG	Systems log
ZH	Scratch
ZI	Scratch

. Estimate Interpretation

The estimate parameter has different meaning depending on the type of peripheral. A list of the interpretations for different peripherals appears below.

TAPE	blank (unless a sequential file is requested, in which case the estimate would be of the same form as that of mass storage.
MASS STORAGE	[minimum/maximum] (defines the minimum and the maximum amount of storage requested. If available the maximum area or any amount down to the minimum is assigned. If the minimum amount is not available a non-assignment status is returned.  [area] (defines an amount of storage requested. If no V option is present the area is taken as the minimum amount of area that is acceptable. If a V option is present any part of the area requested that is available is assigned.
PRINTER	[UM/LM/PL] (UM defines the upper margin, LM defines the lower margin, PL defines the number of printable lines on a page. If this parameter is absent the standard system format will be assumed.
CARD	Blank
PAPER TAPE	Blank
CORE	Same interpretation as mass storage.





### 6.6.2 Peripheral Unit Assignment

Facility map is a drum stored table describing peripheral units and their device handlers. This table is used by facility assignment to assign facilities and initialize request units and locate their appropriate device handler. The file increment to the facility map and its length are maintained in the core summary (Table \_\_\_\_). The address of which is provided by the content supervisors in B5. Words 1 through N and X through X+N define grouping of peripherals under specific peripheral names.

0	△ TO UNIT LIST			△ TO HANDLER LIST		
1	PERIPHERAL NAME 1					
2	A	B	C	▽ <sup>26</sup> HANDLER DESC. <sup>15</sup>	▽ <sup>14</sup>	UNIT DESC. <sup>0</sup>
3	A	B	C	▽ HANDLER DESC.	▽	UNIT DESC.
4	}					
.						
.						
N						
X	PERIPHERAL NAME 2					
X+1	A	B	C	▽ HANDLER DESC.	▽	UNIT DESC.
X+2	A	B	C	▽ HANDLER DESC.	▽	UNIT DESC.
X+3	}					
.						
.						
.						

Word 0

U - The increment from the base of the map to the first unit description.

L - The increment from the base of the map to the first handler description.

Word 1

A peripheral name by which assignment will be requested.

- Word 2-3      2<sup>29</sup> (A) If set 2<sup>29</sup> indicates the end of the list under this peripheral name.
- 2<sup>28</sup> (B) Undefined.
- 2<sup>27</sup> (C) Undefined.
- 2<sup>26-2</sup><sup>15</sup> The Index to the handler description to be used on this assignment. This index is relative to the base of the handler descriptions list.
- 2<sup>14-2</sup><sup>0</sup> The index to the unit description to be used for assignment. This index is relative to the base of the unit description list.
- Word X            Another peripheral name under which assignment is made.
- Word X+1-X+3    Describe the units and handler to be assigned under this peripheral name as do words 2-3.

#### Device Handler Description

To provide flexibility required to allow several distinct device handlers to be eligible for control of a particular device, the peripheral mnemonic specifies both unit and handler as determined at systems generation time. At time of assignment OMEGA will check to see if requested handler is in core. If it is, assignment will be linked to it; if not, handler will be loaded and registered. The handler name and version is overlaid with the handler load information (OA-2A) at initialization time.

#### Handler Description

0	HANDLER	
1	NAME	
2	VERSION	
OA	29 HANDLER LENGTH	17 16 Ø OR HANDLER ADDR. 0
1A	FILE INCREMENT TO HANDLER	
2A	I/O	# ASSIGNMENTS

- Word 0-2      contains name/version of device handler
- OA            contains absolute address and length of routine, the absolute address is Ø if the handler is not in core.

- Word 1A contains drum address of routine
- 2A Number of assignments currently made to routine, if  $\emptyset$  handler will be purged from core when 2<sup>29</sup>th is equal to  $\emptyset$ . 2<sup>29</sup>th set to one indicates once handler is loaded it is to be retained as a permanent resident. Initialization number refers to the proper routine to initialize the unit for the particular handler.

Unit Description:

Contains information pertinent to the particular unit.

Word

0	P - TYPE	LENGTH OF DESCRIPT
1	CH/UNIT	CH/UNIT
2		LENGTH OF UCB
Information Pertinent to Unit		

- Word 0 contains peripheral type number recognized by the system and # of words contained in unit description.
- 1 contains ch and unit of device if dual channel both upper and lower will contain an entry.
- 2 contains the length of the unit control block that will be formed upon assignment.
- 3 N contains information used by facility initialization and/or to be contained in UCB.

Peripheral Types:

Each type of external peripheral device is described to the system at generation time through use of a number.

Mass Storage Devices

- 00 FH432 Drum subsystem
- 01 FH880 Drum subsystem
- 02 Modular FASTRAND
- 03 FASTRAND I
- 04 FASTRAND II
- 05-07 Unassigned

Mass Storage Devices (continued)

- 14 UNISERVO IIIA
- 15
- 16
- 17

Unit Record Devices

- 20 High Speed Card Reader
- 21 High Speed Card Punch
- 22
- 23 High Speed Printer
- 24 1004 Reader
- 25 1004 Punch
- 26 1004 Printer
- 27
- 30 Paper Tape Reader
- 31 Paper Tape Punch
- 32
- 33
- 34
- 35
- 36
- 37

Facility and Storage Summary

As part of the resident EXEC element, a summary of peripherals available is maintained along with the random access storage summaries.

Word

0	ADDRESS OF FREE CORE CORE			
1	FILE INCREMENT TO FACILITY MAP			
2	LENGTH OF FACILITY MAP			
3	FILE INCREMENT TO SELECTION MAP			
4	LENGTH OF SELECTION MAP			
5	FILE INCREMENT TO JOB STACK			
6	ADDRESS OF CHANNEL CONTROL BLOCKS			
7	LOCK SETTING			
10 Channel 0	A	B	C	P-TYPE SUMMARY LENGTH $\Delta$ TO CHANNEL SUMMARY
11 Channel 1	A	B	C	P-TYPE SUMMARY LENGTH $\Delta$ TO CHANNEL SUMMARY
}	}			
33 Channel 23	A	B	C	P-TYPE SUMMARY LENGTH $\Delta$ TO CHANNEL SUMMARY
CHANNEL SUMMARY				

As part of the resident EXEC element, a summary of available peripherals is maintained along with a series of values required by the secondary EXEC routines. Any reference to this summary should use the mnemonic increment indicated on the description.

- Word  $\emptyset$  The address of the free core chain. (FSFCC)
- 1 The file increment to the facility map on the system library file (ZE). (FSIFM)
- 2 The length of the facility map on the system library file. (FSFML)
- 3 The file increment to the selection map on the system library file (ZE). (FSISM)
- 4 The length of the selection map on the system library file. (FSSML)
- 5 The file increment to the job stack module on the cooperative library file (ZD). (FSIJS)
- 6 The address of the channel control block list. (FSACCB)
- 7 A lock location used to lock out references to the facility and storage summary. (FSL)
- 10  $2^{29}$  (A) set to 1 when peripheral on the channel are depleted. (FSCS)
- $2^{28}$  (B) When set an IOC occupies the channel.
- $2^{27}$  (C) Undefined
- $2^{26} - 2^{21}$  A number defining the specific type of peripheral.
- $2^{20} - 2^{15}$  The length of the channel summary.
- $2^{14} - 2^0$  The index to the channel summary relative to the base (WORD  $\emptyset$ ).

Peripheral Entry:

Channel Summary $\emptyset$	MASTER BITS LEFT JUSTIFIED
Channel Summary 1	MASTER BITS LEFT JUSTIFIED

One bit is reserved for each unit on the channel. The bits are justified left. If a bit is not set, the corresponding unit is available. If the bit is set the unit is not available.

### 6.6.3 Random Access Drum Allocation

Assignment of random access storage is in fixed modules the size of which is dependent upon the type of device and number of units on a channel. Each channel contains a summary composed of a bit map used to reflect the availability of modules and develop their starting address; and, a summary used to reduce search time required for assignment. The Drum summary is maintained by facility assignment in the first one or two modules of the described storage.

The following table contains module size utilized by UNIVAC to allocate random access storage. Values and mapping may be changed to reflect installation needs.

FH432 Drum - capacity per drum 262,126 words

<u># of Drums</u>	<u># Words per mod.</u>	<u># of Modules</u>	<u>Length of Summary</u>	<u># of Modules</u>
1	128	2,048	97	1
2	256	2,048	164	1
3	256	3,072	231	1
4	256	4,096	298	2
5	256	5,120	365	2
6	256	6,144	432	2
7	512	3,584	499	1
8	512	4,096	566	2
9	512	4,608	633	2

FH880 Drum - capacity per drum 786,432 words

<u># of Drums</u>	<u># Words per mod.</u>	<u># of Modules</u>	<u>Length of Summary</u>	<u># of Modules</u>
1	256	3,072	143	1
2	256	6,144	255	1
3	512	4,608	184	1
4	512	6,144	235	1
5	512	7,680	286	1
6	512	9,216	338	1
7	512	10,752	389	1
8	512	12,288	440	1

FASTRAND I - Capacity per unit 12,876,128 words (See Note 1)

<u># OF UNITS</u>	<u># WORDS PER MOD</u>	<u># OF MODULES</u>	<u>LENGTH OF SUMMARY</u>
1	2,112	6,144	235
2	2,112	12,288	440
3	2,112	18,432	645
4	2,112	24,576	850
5	2,112	30,720	1,055
6	2,112	36,864	1,260
7	2,112	43,008	1,465
8	2,112	49,152	1,670

FASTRAND II - Capacity per unit 25,952,256 words (See Note 1)

<u># OF UNITS</u>	<u># WORDS PER MOD</u>	<u># OF MODULES</u>	<u>LENGTH OF SUMMARY</u>
1	2,112	12,288	440
2	2,112	24,576	850
3	2,112	36,864	1,260
4	2,112	49,152	1,670
5	4,224	30,720	1,055
6	4,224	36,864	1,260
7	4,224	43,008	1,465
8	4,224	49,152	1,670

Note 1. Allocation module size is be sector to allow "SEARCH" track functions.

Random Access Summary

0	1	Channel #	Peripheral type
1	Capacity of Channel		
2	Available Capacity of Channel		
3	# of words in module .	# of words in sector	
4	# modules currently available		
5	Word count of last search	Length of bit map	
6	# of times bit map referenced		
7	# of times reference futile		
10	Physical address of Group $\emptyset$		
11	Physical address of Group 1		
12	Physical address of group 2		
13	Physical address of group 3		
14	Physical address of group 4		
15	Physical address of group 5		
16	Physical address of group 6		
17	Physical address of group 7		
20	# of Modules available in group $\emptyset$		
21	# of modules available in group 1		
22	# of modules available in group 2		
23	# of modules available in group 3		
24	# of modules available in group 4		
25	# of modules available in group 5		
26	# of modules available in group 6		
27	# of modules available in group 7		

An effort is made through allocation to maintain 4 groups which represent 1/2, 1/4, 1/8 and 1/16 of available storage, allowing the other 4 groups represent continuous areas of random length.



## Bit Map

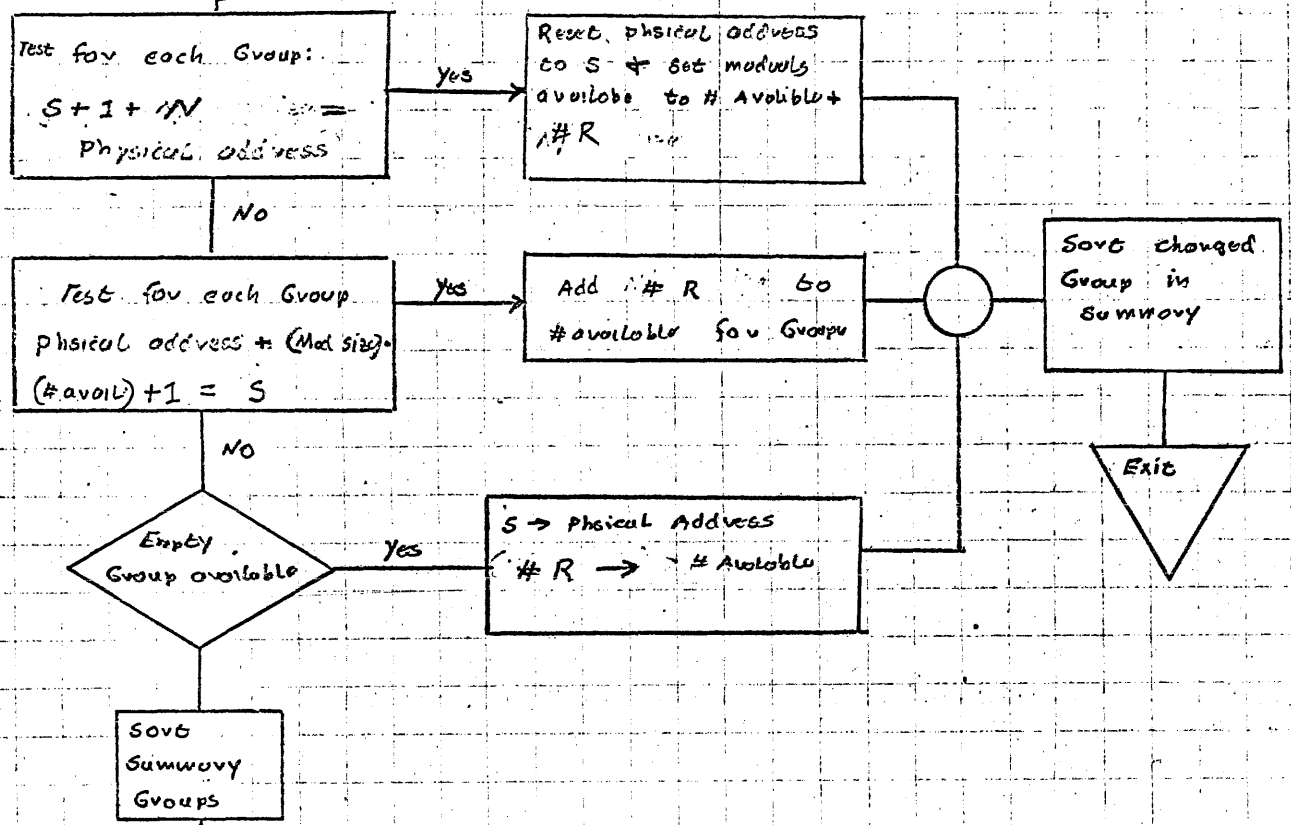
Bit maps are of variable length dependent upon module size and # of word positions assignable on channel. Each bit position within a bit map declares if the corresponding module is free ( $\emptyset$ ) or is reserved (1) and is used to map into a physical address, relative to zero, of the module.p

Physical address X (30 X word position + bit position) X (Module size)

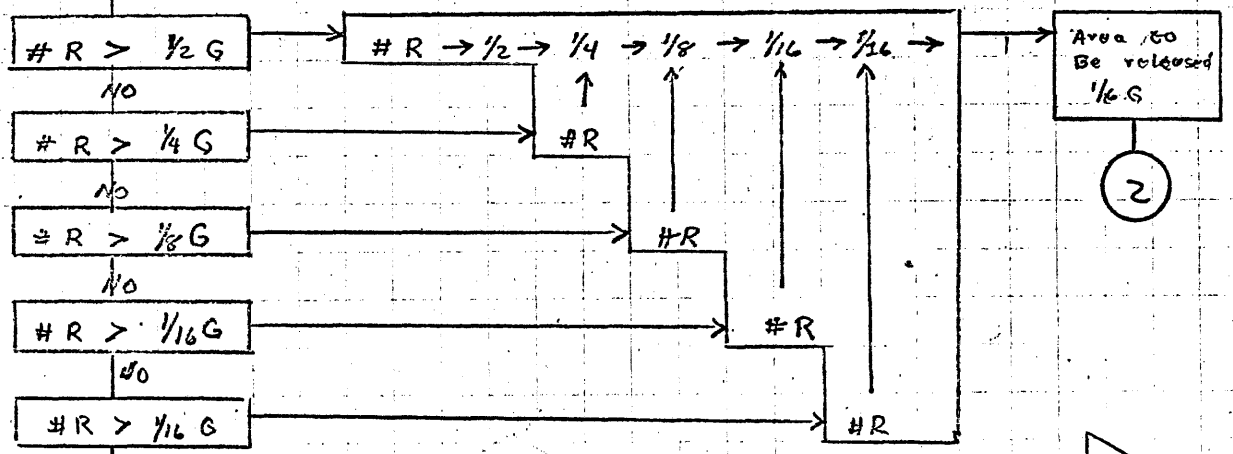
Storager Release

Start Address = S  
 Number of words = N  
 N/Mod size = #R

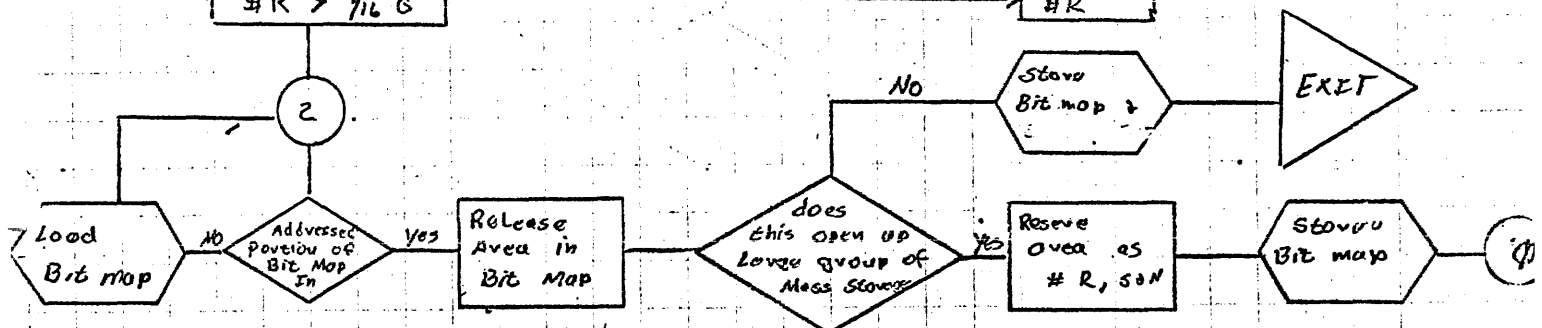
Load Summary for channel  
 If not in

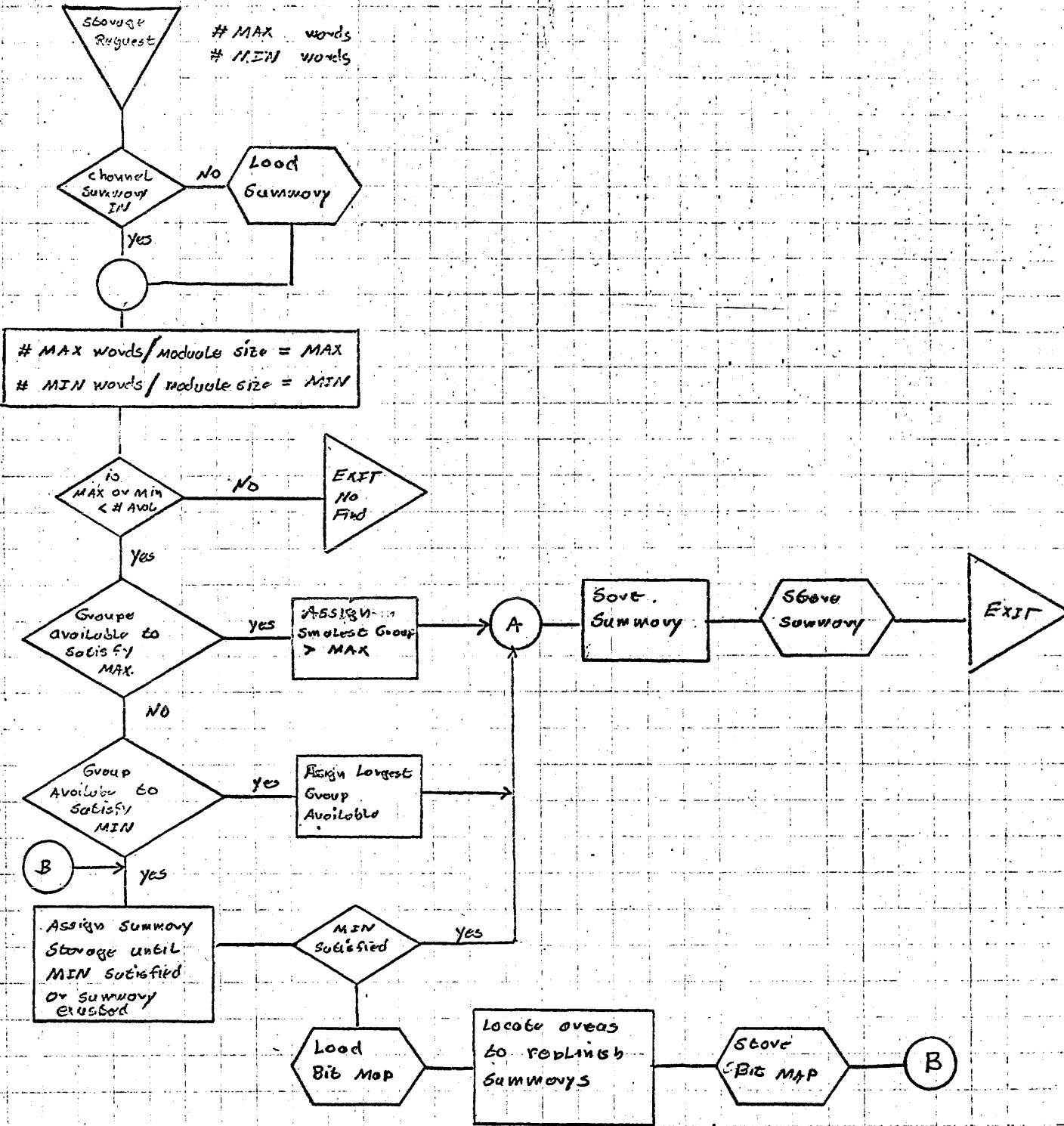


1

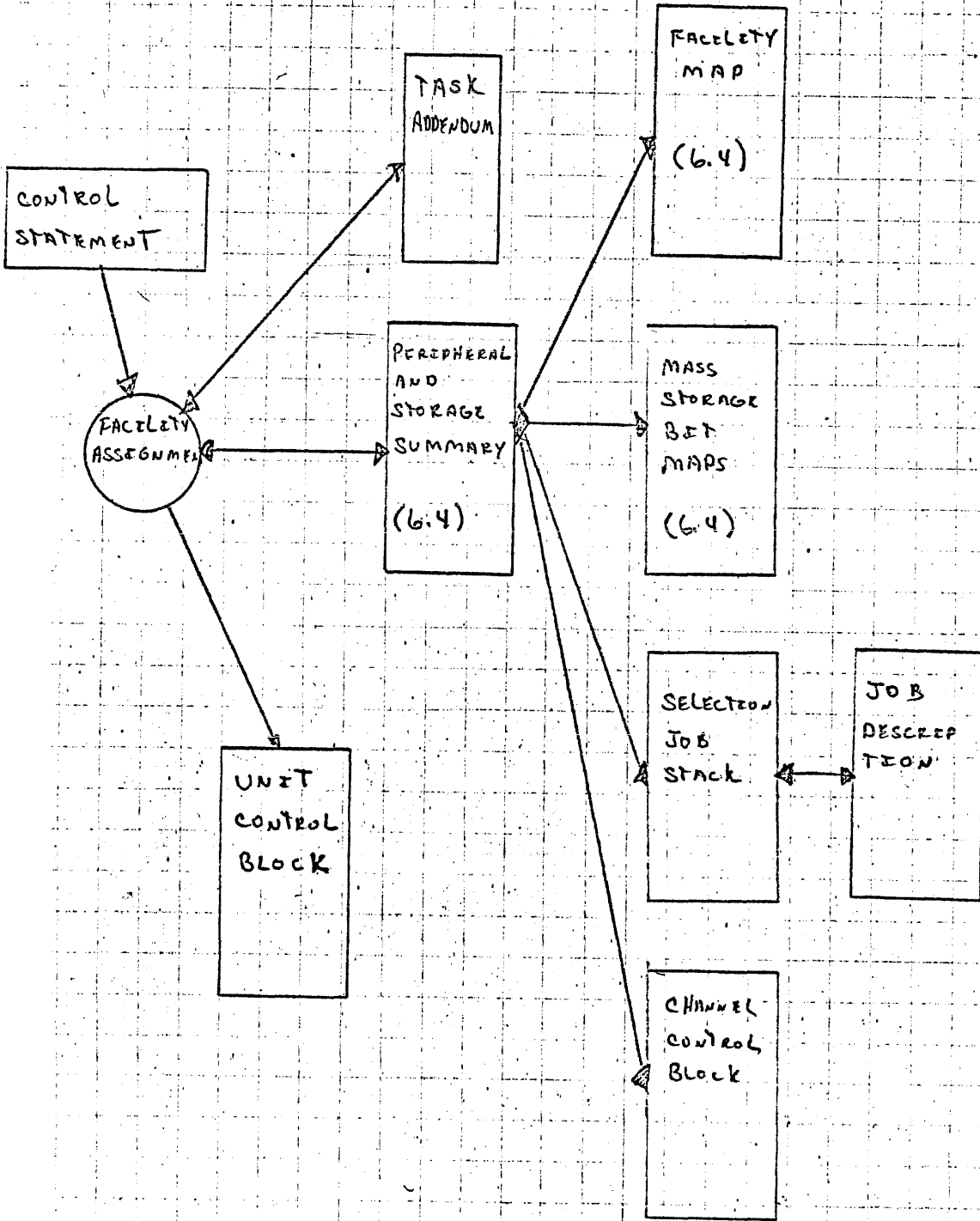


2





• FACELETY ASSIGNMENT TABLE LINKAGE



## 6.6.4

### Master File Directory

#### Description

The Master File Directory catalogs all mass storage files and/or tape files held permanently or semi-permanently by the system. The directory may be constrained to a maximum size at systems generation time and/or extended during operation.

The purpose of the Master File Directory is to retain files between jobs or between the repetition of a given job. The user is relieved of the responsibility of determining at object time physical location of the file and may extend, contract or delete the file during execution.

#### Interface

Entries in the directory are cataloged by a formal request. Each request for a file must use a numeric key 3-7 digits as set by Systems generation denoting his user number and a numeric key 3-7 digits defining his file number. User obtains assignment of a file through use of the ASG control statement and may register an updated version through use of the FREE control statement.

#### Directory Index

Number of entries in the directory index is determined at systems generation time by specifying the number of user indexes desired. For each index assigned an additional index is required to supply number of files qualifiers contained, and 7 digit account number to which charges for storage space are to be made. Directory index #000 is general and reserved for semi-permanent files.

#### Directory Index

Channel #	Physical address of File Index	All binary 1's if unassigned
--------------	--------------------------------	---------------------------------

Directory index is composed of one word entries ordered by user number and covers the span of user numbers defined at systems generation time.

#### File Index

Channel #	Physical address of File Qualifier
--------------	------------------------------------

Each file index is composed of one word entries ordered by file number and covers the range of file numbers defined at systems generation time.

## File Qualifier

0	5 Character password for file reference	
1	Date of last change to file	
2	# of references since last purge	# of days to retain
3	# constituting a unit	current # of units
4	Total # of units to date	
5	File type	# of words in descriptor
7	File Descriptor	
N		

## Password

The five character password may be employed to control the use of confidential data and prevent unauthorized access. For each request for assignment of a protected file, the user must present with the user and file number, the designated password. Any attempt to request assignment of a protected file without the password will cause the Job to be aborted with appropriate messages.

## Words 1-2

Each time a file is explicitly released to the system, date is changed to current date. Number of references since last purge is a count of file assignments made to operating tasks of current file. Number of days to retain is a number supplied by the user specifying the number of days file is to be retained in the directory after last change.

Periodically mass storage files which have expired number of days to retain will be recorded on magnetic tape and their assigned mass storage will be released to the general pool. At time of purge, a summary of deleted files will be submitted to primary output cooperatives listing file characteristics and accounting data.

## Words 3-4

Are used to maintain accounting information. "Number constituting a unit" is a user specified value normal number of words of mass storage to be used for accounting purposes per unit of time. Unit time is assumed as a day. "Current number of units is the largest size of the file for any one day.

"Total number of units to date" is a cumulative total of units for the file; updated each time file is rerequested in the system.

$$\text{Word } 4 = (\text{current date} - \text{word } 1) \cdot (\text{current \# of units}) + \text{word } 4$$

### File Descriptors

File descriptors are used to describe the physical characteristics of the file. These are maintained by the system and are of two types, mass storage and tape.

### Mass Storage

Channel #	Physical address of block 1
0	# of words
Channel #	Physical address of block 2
0	# of words
	.
	.
	.
1	# of words

### External Tape Files

Physical tape # (operator directive)
Symbolic name of unit
Recording options

## File Access

Access to a file contained in the Master File Directory cannot be performed until an ASG control statement is given specifying the file code to be used, user number, file number and password if required. Example: A request for file number 932 under user index 035 to be assigned to file code B would appear as follows:

```
ASG^ARANR,B, 035/932
```

Once assignment has been made, the user activity may perform all packet level or file control I/O commands available in the system including extension or contraction of file. However, if user has changed the limits of the file, he must reregister file through use of FREE control statement in order to maintain update. Example: the above file being reregistered in the directory would appear as follows:

```
FREE^PAB, 035/932
```



### 6.6.5 Facility Assignment Initialization

Parameters required for system generation used to form systems records and/or initialize facility assignment according to configuration and/or desired user options.

a) For each channel of subchannel, the following information is required:

- . Channel number
- . Peripheral type
- . Mnemonic name of peripheral type
- . Number of units
- . Cost per unit of time for user to retain word or device.
- . Name of handler responsible for channel or subchannel

b) Option for accounting information

- . Normally accounting method is a summary of units assigned to Job.

$\{ (\text{Length of time held} \times \text{number used}) \text{ cost per unit time}$

- . Option method is to submit to systems log each assignment listing type of device, number of units and length of time retained.

c) Establishment of User Index

- . 0-7 digit number of user indexes
- . Channel number on which user index should be maintained.

d) Establishment of qualifier index

- . User index number
- . Account number
- . Number of files
- . Optional Password
- . Channel number on which index should be maintained

6.7 Service Function 3 - Library number 006 is a non-reentrant routine used to create and release core chains, process Send and Receive requests, simulate hardware search, and retrieve an element.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	FCHAIN\$	Form a core chain
02	RCHAIN\$	Release a core chain
03	Unassigned	
04	SEND\$	Transmit parameters to Exec storage
05	RECEIVE	Complement of SEND
06	SEARCH\$	Simulated hardware search through repetitive read.
07	Unassigned	
10	Unassigned	
11	SENDP	Send parameters
12	RECEIVEP	Receive parameters
13	Unassigned	
14	SEARCHTL\$	Search track and lock
15	SEARCHPL\$	Search position and lock
16	SEARCHL\$	Same as SEARCH\$ with lock
17	BLOCKSL\$	Block search on drum with lock
20	FETCH\$	Load absolute element in assigned core
21	FETCHR\$	Same as FETCH\$ with activity registration
22	FETCHL\$	Locate named element and read the table of contents for that element.
23	FETCHRQ\$	Same as FETCHR\$ with queued activity registration
24	Unassigned	
25	Unassigned	
26	Unassigned	
27	Unassigned	

Service Functions - Library number 006 is a non-re-entrant service routine used to process the following service requests.

•REQUEST CHAIN (Function code 01)

This request will allocate core chain. These chains are available to RT/comm only and can be used to load worker programs, buffers or data pass areas. Each link will be a continuous memory area; links within a chain will occupy contiguous areas of memory. Links will be obtained from free storage. During peak periods of processing, RT/comm may establish additional chains, at the expense of lower priority programs, to accomodate overflow.

Each chain must be assigned a unique number. Since numbers 0-5 are reserved for the executive, this number must be 6 or more. Links within a chain may be declared either fixed module or variable length. The fixed module chain is composed of links of a specified size. These modules are requested one at a time as needed. The fixed module chain has the advantage of having a faster request/release mechanism as well as requiring one less parameter at usage time. The second type of chain, variable, allows the user to variable sized pieces of core. This type of chain is more flexible than the fixed module chain and wastes less memory.

The number of words in the chain (variable chain) or per module (fixed module chain) must be an even number, i.e., divisible by 2. If the parameter V1 is not an even number, 1 will be added to it to make it even.

Operator: FCHAIN\$ΔVO,V1,V2

For fixed module chains:

V0 - chain number (from 6-N);  
V1 - number of words per module; and  
V2 - number of modules

For variable chains:

V0 - chain number (from 6-N)  
V1 - total number of words in chain; and  
V2 - 0

Packet:

ENT*B7	V0
ENT*Q	V1
ENT*A	V2
7 7 5 4 0	2 0 1 4 1

The A register will be set zero if request completed, negative if core not available. The Q register will contain the number of words or modules allocated.

•RELEASE CHAIN (Function code 02)

This request is used by a RT/comm program to deallocate a previously declared chain. Care must be used so as not to release the chain before all expansions have been returned to the general storage pool.

Operator: RCHAIN\$ΔVO

where VO - chain number

Packet:

ENT*B7	VO
7 7 5 4 0	2 0 1 4 2

•TRANSMIT DATA (Function code 04)

This request will transmit limited data sets from an activity or task to executive storage. This will allow an independently executed activity or task within the job to retrieve the transmitted data by means of a RECEIVE\$ operator. All transmitted data will have a five character octal identification attached to it. This identification will allow the receiving element to specify a specific data set. This identification will not be transferred to the receiving element. It should be noted that once a data set is received by an independent element, it is purged from executive storage and cannot be received again.

Operator: SEND\$ΔVO, V1,V2

where VO - base address of data field relative to lower lock;  
V1 - number of words in the data field; and  
V2 - a 5 character octal identification (zero implies no identification.)

Packet:

ENT*B7	Base Address
ENT*A	No. of Parameter Words
ENT*Q	Identification
7 7 5 4 0	2 0 1 4 4

•RECEIVE DATA (Function code 05)

This operator will transfer data from executive storage to the requestor. This data will have previously been sent to executive storage by a SEND\$ operator initiated by an independent or asynchronously executed activity or previous task. Data may have a unique identification attached to it by the SEND\$ operator that transmitted it to executive storage. This identification will make it possible to select sets of data. In general, data sets with the same identification as the identification field of this operator (next page) will be received by the requestor in the order that they

were sent to executive storage, i.e., on a first in, first out, (FIFO) basis. The requestor will receive only that data that was transmitted by one SEND\$ operation. Thus there may be more data in executive storage with the same identification. As a data set is transferred to the requestor, it is purged from executive storage. This function may also be used to transfer COMMON from one chained element to the next. This operator will not transfer control card data (see RECEIVE\$ operator).

When control is returned to the activity, the A and Q registers will contain the following status information:

<u>REGISTER</u>	<u>CONTENTS</u>
Q	Number of data words transferred to receiving field. Q set negative if no data for particular identification left to be transferred.
A	Set to 0 if data set fits within receiving field. Set non-zero if data set overflowed the receiving field. Receiving field will be filled with as much data as it will hold. A register will then contain number of words left in data set. Data transferred will be purged. Next RECEIVE\$ with this identification will transfer remaining words of data set.

Operator: RECEIVE\$ΔVO,V1,V2

where V0 - base address of receiving field, relative to lower lock;  
 V1 - maximum number of words in receiving field; and  
 V2 - 5 character octal identification (0 implies no identification)

Packet:

ENT*B7	Base Address
ENT*A	Field Size
ENT*Q	Identification
7 7 5 4 0	2 0 1 4 5

•SEARCH A FILE (Function code 06)

This function will search a file in the forward direction for a block (called the find block) whose first word is equal to some specified searchword. If such a block is found, it is read into the buffer area. This read will be performed regardless of the lock setting of the buffer area (the buffer area will not be disturbed unless the proper block is found). The search will continue until either the block is found or an end of file is detected. Control will not be returned to the requesting activity until the search process has been completed. To avoid typing up the synchronizer, the search will be performed by software; hardware search features will not be utilized. Only one file or tape reel will be searched.

A search of a random access storage device is a fixed block search. The number of words parameter (V1) is the block size. The logical address of any block (excluding the first) will be the logical address of the previous block plus the number of words (V1).

If the search is successful, the find block will be read into the buffer area and normal I/O status with the exception of B7 and A. The A register will contain the logical address of the find block. B7 will contain the number of blocks searched (count includes the find block). Tape files will be positioned after the find block.

If the search is unsuccessful due to end of file, E.O. F. status word will be returned in the A register ( if the search is unsuccessful for any other reason, the proper status word will be in the A register. B7 will contain number of blocks read.

Operator: SEARCH\$ΔV0,V1,V2,V3,V4

where V0 - file code;  
 V1 - number of words in buffer area;  
 V2 - buffer base relative to lower lock;  
 V3 - logical address from base of file (normally used for random access storage devices); and  
 V4 - 10 character octal searchword

Packet:

EBJP*B7	N
File Code	Buffer Base
Buffer Base	
Logical Address	
Searchword	
7 7 5 4 0	2 0 1 4 6

Note: A buffer size of zero will turn this request into a locate function. Nothing will be read into memory if the block is found. Everything else remains the same.

• TRANSMIT CONTROL CARD PARAMETERS (Function code 11)

This request will transmit a limited set of parameters from a control card or an unsolicited operator type-in to executive storage. A worker program will then be able to retrieve these parameters by means of the RECEIVEP\$ or RECEIVE\$ operator. This function requires that the base address, relative to lower lock, of the control card parameter field be placed in B7 and the length of the parameter field in the A register.

Operator: SENDP\$ΔV0,V1

where V0 - is base address relative to lower lock of parameter field  
 V1 - contains number of words in parameter field

Packet:

ENT*B7	Base Address
ENT*A	No. of Parameter Words
7 7 5 4 0	2 0 1 5 1

•RECEIVE CONTROL CARD PARAMETERS (Function code 12)

This request will transfer control card parameters from executive storage to the requesting activity. These control card parameters will have been transmitted to executive storage by a SENDP\$ operator. After the control card information is transferred to the requestor it will be purged from executive storage. Unlike the RECEIVE\$ operator, all control card parameters, regardless of the number of different cards that are stored, will be received by the requestor.

When control is returned to the activity, the A and Q registers will contain the following status information:

REGISTER

CONTENTS

Q	Number of words of control card parameters transferred to receiving field. Q set negative if no control card parameters left to be transferred.
A	Set to 0 if parameters fit within receiving field. Set non-zero if parameters overflow with as many parameters as it will hold. A register will then contain number of words of parameters left. Transferred parameters will be purged. Next RECEIVEP\$ will transfer remaining words of parameters.

Operator: RECEIVEP\$AVO,V1

where VO - base address of receiving field, relative to lower lock; and

V1 - maximum number of wrds in receiving field.

Packet:

ENT*B7	Base Address
ENT*Q	Field Size
7 7 5 4 0	2 0 1 5 2

•SEARCH FASTRAND TRACK WITH LOCK (Function code 14)

This function is essentially a search track and READL\$. It is applicable to FASTRAND equipment only. The SEARCHT\$ function will be a part of this routine. The user should be aware that the SEARCHT\$ is a hardware function and installation standards regarding use of the SEARCHT\$ operator should be checked before using the SEARCHTL\$ operator. This operator will compare the first word of consecutive FASTRAND sectors against a specified searchword. If the end of track is encountered before a match is made, the search is terminated and an unsuccessful

search status code returned in the A register. If a sector is found whose first word matches the searchword, a READL\$ will be issued for parameterized data transfer. If the area is locked out the read (and, therefore, the activity) will be delayed until the area is free. When the area is free, the block will be read into the buffer area and entered into the lock list. The first word of the buffer area will be the searchword, i.e., the read begins from the searchword. For successful find the A register will be set positive and contain the logical address of the word on FASTRAND that equalled the searchword. All other status returns hold as stated for normal I/O.

Operator:                   SEARCHCTL\$AVO,V1,V2,V3,V4

where V0 - file code;  
 V1 - number of words in buffer;  
 V2 - base address of buffer, relative to lower lock;  
 V3 - logical address relative to base of file; and  
 V4 - 10 character octal searchword

Packet:

EBJP*B7	N
File Code	Buffer Length
Buffer Base	
Logical Address	
Search Word	
7 7 5 4 0	2 0 1 5 4

N

Note: If the buffer length is set to zero this request will become equivalent to a locate function. Nothing is read into core. The logical lock list is not consulted. Everything else remains the same.

•SEARCH FASTRAND POSITION WITH LOCK (Function code 15)

This function is essentially a search position (SEARCHP\$) and READL\$ operator. It is applicable to FASTRAND equipment only. The SEARCHP\$ function will be a part of this routine. The user should be aware that the SEARCHP\$ is a hardware function and installation standards regarding use of the SEARCHP\$ operator should be checked before using the SEARCHLP\$ operator. This operator will compare the first word of consecutive FASTRAND sectors against a specified searchword. If the end of position is encountered before a match is made, the search is terminated and an unsuccessful search status code returned in the A register. If a sector is found whose first word matches the searchword, a READL\$ will be issued for parameterized data transfer. If the area is locked out, the read (and, therefore, the activity) will be delayed until the area is free. When the area is free, the block will be read into the buffer area and entered into the lock list. The first word of the buffer area will be the searchword, i.e., the read begins at the searchword. For successful find the A register will be set positive and will contain the logical address of the find on FASTRAND that equalled the searchword. All other status returns hold as for normal I/O.



Operator: SEARCHLP\$ΔVO,V1,V2,V3,V4

where V0 - file code;  
V1 - number of words in buffer;  
V2 - buffer base relative to lower lock;  
V3 - logical address relative to base of file; and  
V4 - 10 character octal searchword

Packet:

EBJP*B7	N
File Code	Buffer Size
Buffer Base	
Logical Address	
Search Word	
7 7 5 4 0	2 0 1 5 5

 N

Note: If the buffer size is set zero this request will become equivalent to a locate function. Nothing is read into core. The logical lock list is not consulted. Everything else remains the same.

\*SEARCH LOCK (Function code 16)

This request is virtually the same as the SEARCH\$ operator. The user is directed to the SEARCH\$ description for specifications regarding normal and error conditions. There is one important difference between this operator and the SEARCH\$ operator. When a successful search occurs, the SEARCH\$ operator performs the READ\$ regardless of the logical lock status of the file. This operator does not. This operator performs a READL\$ (this is only pertinent if the file is on a random access storage device). If the area is locked out, the read (and, therefore, the activity) is delayed until the area is released. When the area is free, the read is performed. The accessed area will then be placed on the lock list. The request will go on to normal completion.

Operator: SEARCHL\$ΔVO,V1,V2,V3,V4

where V0 - file code;  
V1 - number of words in buffer area;  
V2 - buffer base relative to lower lock;  
V3 - logical address from base of file (normally used for random access storage devices); and  
V4 - 10 character octal searchword

Packet:

EBJP*B7	N
File Code	Buffer Size
Buffer Address	
Logical Address	
Search Word	
7 7 5 4 0	2 0 1 5 6

 N

Note: A buffer size of zero will turn this request into a locate function. Nothing will read into memory if the block is not found. Since the find block is not accessed, logical lock will be ignored.

•BLOCK SEARCH FH DRUM WITH LOCK (Function code 17)

This function is essentially a block search that performs a READL on the find area. It is applicable to Flying Head drum equipment (FH880, FH432, etc.) only. The function BLOCKS\$ will be a part of this routine. The user should be aware that BLOCKS\$ is a hardware function and installation standards regarding use of the BLOCKS\$ operator should be checked before using this BLOCKSL\$ operator. This operator will compare consecutive words on drum against a specified searchword. If an end of block sentinel (a word of all binary ones 7777777777) is encountered before a match is made, the search will terminate and an unsuccessful search status code returned in the A register. If the find area is locked out the read (and, therefore, the activity) will be delayed until the area is free. When the area is free, the block will be read into the buffer area. The first word in the buffer area will be the searchword, i.e., the read begins from the searchword. The area that was read will be entered into the lock list. The A register will be set positive and will contain the logical address of the word on drum that equalled the searchword. All other status conditions hold as in normal I/O.

Operator: BLOCKSL\$AV0,V1,V2,V3,V4

where V0 - file code;  
 V1 - number of words in buffer area;  
 V2 - buffer base relative to lower lock;  
 V3 - logical address relative to base of file; and  
 V4 - 10 character octal searchword

Packet:

EBJP*B7	N
File Code	Buffer Base
Buffer Base	
Logical Address	
Searchword	
7 7 5 4 0	2 0 1 5 7

N

Note: If the buffer size is set to zero this request will be equivalent to a locate function. Nothing will be read into core. The lock list will not be consulted. Everything else remains the same.

•SUBROUTINE LOAD (Function code 20)

This operator will load a named absolute library program into a specified location. This is not a segment but rather a subroutine load. The operating task can control operation of absolute programs. The program that is loaded will not be activated until a fragmentation request is made. The operating base and memory lockout protection associated with the fetched subroutine is a subset defined by the requestor through activity registration. If the request is successfully completed, the A register will be set positive. If the subroutine could not be loaded, the A register will be set to the appropriate systems status code.

Operator:            FETCH\$ΔVO,V1,V2

- where VO - base address, relative to lower lock, of core area where subroutine is to be loaded;
  - V1 - name/version of called element; and
  - V2 - library in which call absolute element is contained.
- This may be

SYSTEM - systems library  
 JOB - job library  
 GROUP/library number - named group library previously linked to job. If field is blank, the job library is assumed.

Packet:

EBJP*B7	\$+7
Base Address	
N - - - - -	-N
N - - - - -	-N
V - - - - -	-V
Library Type	
Library Number If Group Lib (0 IF NA)	
7 7 5 4 0	2 0 1 6 0

Note: The base address must be a multiple of 100 (octal), i.e., the two right hand digits must be 00.

•LOAD SUBROUTINE AND REGISTER AS ACTIVITY (Function code 21)

This function is a combination of the FETCH\$ and REG\$ operators. The requested subroutine will be loaded into the specified core area and registered as an activity. The user is directed to the documentation for each of these requests for details. For this request, the base address of the FETCH\$ function will be the same as the address of activity in memory of the REG\$ function. The activity mode indicator of the REG\$ function is always 0 for this combined function. This is because the FETCH\$ operation sets the RIR to the base address of the subroutine. If the request is successfully completed, the A

register will be set to appropriate systems file code.

Operator:            FETCHR\$AVO,V1, V3,V4,V5,V6

- where V0 - base address, relative to lower lock, of core area where subroutine is to be loaded (also implicit starting point of activity);  
 V1 - name/version of called element;  
 V2 - library in which called absolute activity is contained; this may be:

SYSTEM - system library  
 JOB- job library  
 GROUP/library number - named group library previously linked to job;

- V3 - address of data area in core, relative to lower lock;  
 V4 - length of data area;  
 V5 - data area mode: zero indicates read/write lockin will be set to the data area defined by V3 and V4 and read will be permitted from any area (B registers B4 through B7 are set to 17 bit mode); non-zero indicates use read/write lock of requestors; and  
 V6 - relative response priority (0-17); if priority not specified, priority of requestor is assumed.

Packet:

EJJP*B7		\$+11	
Base Address			
N - - - - -		N	
N - - - - -		N	
V - - - - -		V	
Library Tape			
Library Number (O IF NA)			
Address of Data Area			
2 <sup>29</sup> 0	2 <sup>28</sup> V5	Priority	Length of Data Area
7 7 5 4 0			2 0 1 6 1

Note: Base address (V1) V3 and V4 must be a multiple of 100 (octal).

• LOCATE ROUTINE AND READ THE TABLE OF CONTENTS (Function code 22)

The function is a subset of the FETCH operation. The requested routine will be located on the specified library. The table of contents for the element will be retrieved into a 9 word area specified by the requestor.

Operator            FETCHL\$ VO, V1, V2

where VO - base address, relative to lower lock of a 9 word area where the table of contents is to be placed.

V1 - name/version of specified element

V2 - Library in which the named routine is contained, this may be:

SYSTEM - system library

JOB - job library

GROUP/library number - a group library linked to job

Packet

EBJP*B7	\$+6
	BASE ADDRESS
N- - - - -	- - - - -N
N- - - - -	- - - - -N
V- - - - -	- - - - -V
LIBRARY TYPE	

.LOAD SUBROUTINE AND REGISTER AS QUEUE PROCESSED ACTIVITY (Function code 23)

This function is a combination of the FETCH\$ and REGQ\$ operators. The requested subroutine will be loaded into the specified core area and then registered as queue processed activity. The user is directed to the specifications for the FETCH\$ and REGQ\$ operators for detail. For this request, the base address of the FETCH\$ function will be the same as the starting point of the activity of the REGQ\$ function. If the request is successfully completed, the A register will be set positive. The A register will be set negative if the subroutine could not be loaded.

Operator:                    FETCHREGQ\$ΔV0, V1, V2,V3,V4,V5

- where V0 - base address, relative to lower lock of core area where subroutine is to be loaded (also implicitly starting point of activity);
- V1 - name/version of called element;
- V2 - library in which called absolute activity is contained; this may be:

- SYSTEM - system library
- JOB - job library
- GROUP/library number - named group library previously linked to job;

- V3 - length of activity; zero implies read/write lock will remain set to that of requesting-activity and the activity considered an integral part of the compiler requesting activity; non-zero length defines the area to be protected by memory lock-in and sets the RIR to the value in V0;
- V4 - five character (octal) identifier used for reference by QREF\$ operator; and
- V5 - relative response priority (0-17); if priority not specified, priority of requestor is assumed.

Packet:

EBJP*B&	\$+11
Base Address	
N - - - - -	N
N - - - - -	N
V - - - - -	V
Library Type	
Library No. If Group Lib (O IF NA)	
Length of Activity	Identifier
	Priority
7 7 5 4 0	2 0 1 6 3

Note: Base Address must be a multiple of 100 (octal)

## 6.8 Cooperative Service Routine C.S.R.

Library number 007 is a non re-entrant routine used to perform service requests for I/O cooperative control and for controlling the loading activation and processing of Unit Record Routines. The following is a summary of function codes to C.S.R. followed by a description of each function containing parameters required,

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	Internal	Load and activate primary input unit record routine
02	CALL\$	Same as 01
03	Internal	JOB card sensed, set up necessary linkage
04	Internal	Close primary input stream and return
05	Internal	Close primary input stream and terminate
06	Internal	Activate primary output U.R. routine
07	Internal	Terminate primary output U.R. routine
10	Internal	Activate secondary output U.R. routine
11	Internal	Terminate secondary output U.R. routine
12	Internal	Load and activate named edit routine
13	Internal	Terminate edit routine
14	Unassigned	
15	Unassigned	
16	Internal	I/O cooperative library overflow possible to occur
17	Internal	Deallocate overflow mass storage extension to I/O cooperative library
20	Internal	Primary or secondary output module overflow
21	Internal	Primary or secondary output module error overflow

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
22	Internal	I/O cooperative module drum error
23	Unassigned	
24	Unassigned	
25	Unassigned	
26	Unassigned	
27	Unassigned	
30	Unassigned	
31	Unassigned	
32	Unassigned	
33	Unassigned	

Functional description of function codes

Function code 01 - Load and activate primary input unit record routine indicating scheduling information and limited data is available to be entered into the system and processed.

Caller: Console Control routine upon receiving an unsolicited request from computer operator to begin a scheduling pass from a particular device.

UR  Name/Version  S

Type of device is determined by facility statement collected with named unit record routine.

Parameters: B7=Address of unsolicited call for primary input unit record routine.

# of Char.	# of Words
Name	
Name	
Version	

•# of words - allocated by console handler from free core chain to contain operator's message.

•# of characters contained in message.

Addendum: Exec task activity addendum

Function: •Locate and validate existence of called for unit record routine, queue request if insufficient



cooperative mass storage.

- From task addendum and link to task addendum chain. Switch activity addendum and perform call to pre-selection.
- Deallocate core used to contain call
- Form job summary containing call to unit record routines

- Exits:
- If error conditions submit diagnostic message to console operator and exit to content supervisor via XX function code.
  - If request queued, exit content supervisor via XX function code.
  - Successful, register activity to worker addendum and switch to pre-selection F.C. XX via content supervisor.

Function code 02 - Same as function code

Caller: Internal call from operating program via CALL packet

Parameters: B7 = address of CALL packet containing name/version of primary unit record routine.

Call Operator: CALL\$ Name/Version

Packet:

EBJP*B7	\$+4
Name	
Name	
Version	
7 7 5 4 0	2 0 3 4 2

Addendum: Operating worker task/activity addendum

- Function:
- Locate and validate existence of called for unit record routine, queue request if insufficient cooperative mass storage.
  - Form task addendum and link to task addendum chain. Register activity and perform call to pre-selection. Set completion status code 00 in A register of caller.
  - Form job summary containing call for unit record routine.

Exits:       •Return to requestor

              "A" register = 0000000000 Call completed  
              "A" register = 7777740006 Routine could not be located  
              "A" register = 7777740001 Invalid unit record routine

              •Register call to pre-selection FC 01 under new  
              task/activity.

Function code 03 - signaling the start of a primary input control stream

Caller:       primary input unit record routine upon sensing  
              a job card

Parameters:   B7 set to the address of a completed module  
              description block. (See figure       ) describing  
              a completed primary input module in which the  
              JOB card is the first item.

Addendum:    Operating worker task/activity addendum

Function:     •Allocate drum modules and store submitted primary  
              input module in I/O cooperative library.

              •Set up primary input chain

              •Register call for pre-selection to activate  
              schedule sequence.

Exit:         •Return to requestor upon successful completion  
              "A" registers 00

Function code 04 and 05

Caller:       Primary input unit record routine requesting the  
              close of current control stream and return of  
              control under new task addendum (04) or the termina-  
              tion of routine (05).

Addendum:    Operating under control of worker task/activity  
              addendum.

Parameters:   Last storage module and description packet. If  
              incomplete, module item count must reflect # of  
              items contained in B7 set to address of  
              description packet.

Function:     •Close primary input stream and store last module  
              if stream was stored.

              •POP any outstanding requests for primary input on  
              this stream.

- If function code 04 form new task addendum, link UCB and activity addendum.
- If function code 05 deallocate facility assignment ZA, core storage of U.R.

Exit:       Function code 04 Return control to requestor  
               Function code 05 RETURN exit through content supervisor

Function Code 06 - Load and activate primary output U.R. routine

- Caller:       •Termination due to one of the following: all tasks within the job stream have been completed. Task has terminated and cooperative library contains excessive primary output for task.
- Cooperative service routine due to excessive primary output accumulated for task as signaled by I/O cooperative control calling C.S.R. via Function code 20.

Addendum:   Operating worker task/activity addendum

- Function:     •Retrieve job summary and locate called for unit record routine.
- Form I/O cooperative module containing CALL card as first item and job identification as second item and link to head of primary output stream.

Exit:        Pre-selection to form selection summary for unit record routine.

Function Code 07 - terminate primary output unit record routine.

Caller:       Primary output unit record routine upon sensing on E.O.F. from I/O cooperative control.

Addendum:   Operating worker task/activity addendum

- Function:     •Request cooperative control to close primary output stream.
- If non-standard output routine release core and facility assignment.
- If standard output routine; search for call of a standard U.R. output routine. If find made, switch control to new task addendum, repeat step 2 Function code 06 and register U.R. for reactivation.

Exit:        Call termination

Function code 10 - Load and activate secondary output U.R. routine

- Caller:
- "Termination" due to one of the following: All tasks within the job stream have been completed. Task has terminated and cooperative library contains excessive secondary output for task.
  - "Cooperative service routine" due to excessive secondary output accumulated for task as signaled by cooperative control calling C.S.R. via Function code 20.

Addendum: Operating worker task/activity addendum

- Function:
- Retrieve job summary and located called for unit record routine. Set call bit in "Job Stuck" for secondary output.
  - Form cooperative drum module containing call card as first item and identification as second item and link to head of secondary output chain.

Exit: Pre-selection to form selection summary for unit record routine.

Function code 11 - Terminate secondary output unit record routine.

Caller: Secondary output unit record routine upon sensing an end-of-stream status from cooperative control.

Addendum: Operating worker task/activity addendum.

- Function:
- Request cooperative control to close output stream.
  - If non-standard unit record routine release core and facility assignment and exit by normal return mechanism.
  - If standard output routine; search for call of a standard secondary output unit record routine. If find made switch control to new task addendum, repeat step 2 of C.S.R. Function code 10, reset storage module to restart unit record. If no find made, deallocate core and facility.

Exit: Call termination via normal RETURN through content supervisor.

Function code 16 - Allocate additional mass storage for I/O cooperative library and form bit map for acquired mass storage.

Caller: • Systems initialization to effect allocation of base map overflow.

•From C.S.R. as the result of processing Function code 20 of C.S.R.

Parameters: None

Addendum: Worker task/activity addendum

Function: •If maximum number of additional extensions to cooperative library has not been obtained submit the following facility request statement through the normal facility assignment routine.

ASG\$ $\Delta\Delta$ I/O LIB, ZD, 95040D/190080D

where - IOLIB is type of random access storage required to satisfy request as determined at time of "facility map" formation; Preferably IOLIB should be equaled to medium grademass storage FH880.

ZD indicates cooperative library file code

95040D/190080D are the minimum/maximum number of words used on extension requests. 190080D words will provide 960 modules and use up a  $37_8$  position bit map.

•Allocate core and form a  $37_8$  position bit map setting end bit position to reflect lock of maximum allocation from facility assignment. 190,080 words of mass storage provides 960 modules at 198 words.

Adjust module counters and link to "Bit map descriptors" (See 7.4.2)

Exit: Return to requestor "A" set 0—0 if allocation successful. "A" set 7777740006 indicating mass storage unavailable.

Function Code 17 - Deallocate an unused mass storage extension to cooperative library.

Caller: "Cooperative Control" when it has detected all storage modules within a particular bit map have been released.

Parameters: None

Addendum: Task/activity addendum of program causing release of last module.

**Function:** •Check for any activities which were pushed on "C.S.R." chain cell due to overflow of a stream or a primary input U.R. routine which was not activated due to lack of mass storage. If either is found ready its lost control and re-enter point, "pop" request and exit to original caller.

•Check last bit map in chain of bit maps (extensions) is empty. If so, deallocate mass storage extension, release bit map, core and repeat this step. If last bit map is not empty exit to caller.

**Exit:** Control is always returned to call or with successful completion status code "A" register = 0——0.

Function code 20 - a call to indicate one of the primary or secondary output streams has reached the maximum number of modules which can be contained on the cooperative library at any one point in time. This number is set by systems convention.

**Caller:** Cooperative control upon sensing overflow.

**Parameters:** B2 register set "0" or "1" indicating which stream primary or secondary respectively has overflowed.

**Addendum:** Task/activity addendum of program causing overflow.

**Function:** Perform the following sequence of checks.

- a) Read "Job Stack" and check to see if unit record routine has been called for. If so, skip to step C.
- b) "REGCT" call for C.S.R. function code 06 or 10 to call primary or secondary output routine.
- c) Execute C.S.R. subroutine used process Function code 16 to obtain additional mass storage. If additional mass storage is obtained return control to requestor. If additional mass storage is not obtained "PUSH" request of content supervisor chain for lock of mass storage return to requestor as next function to perform on subsequent "POP".

**Exit:** Return program control to cooperative control if additional mass storage can be obtained, otherwise PUSH activity until unit record is activated.

## 6.9 Pre-Select

The function of pre-selection is to obtain explicit and implied scheduling parameters which have been presented to the system by the control language, from a pass through the loader or are registered in the systems library. The scheduling parameters are summarized for use by selection. All control statements required task/activity are readied for CCI phase.

Pre-selection is activated by the following function codes

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	Internal	Pre-select primary input unit record routine
02	Internal	Pre-select next task contained in control stream
03	Internal	Pre-select primary output unit record routine
04	Internal	Pre-select secondary output unit record routine
05	Internal	Pre-select edit routine for primary input
06	Internal	Control card interpreter
07-33	Unassigned	

## Functional Description of Function Codes

Function code 1, 3, 4, 5: Ready call for unit record routine for selection and initiation.

Requestor: Cooperative Service Routine upon validating a call for the load and activation of a U.R. routine.

Exits: Release Pre-selection and queue a call for Selection under exec addendum.

Addendum: Operating worker task/activity addendum

Processing Procedure:

- a. Find the Job Description through the appropriate entry in the JOB Stack.
- b. Locate the appropriate control statement within the Job Description.
- c. Locate the specified routine on the library.
- d. Preprocess and form summaries for any control statement attached to absolute element.
- e. Update the Job Description and the Job Stack.

Parameters: FC 1 EXRN\*20401

FC 3 EXRN\*20403

FC 4 EXRN\*20404

FC 5 EXRN\*20405

Function code 2: Ready a task for selection and initiation

Requestor: Cooperative Service Routine upon sense of a job statement or by Termination for selection of next task in job already started.

Exits: Release Pre-selection and queue a call for Selection under the exec addendum.

Processing Procedure:

- a. Request an image from cooperative control.
- b. If JOB cards set up Job Stack, Job Description



- c. Pre-process all control statements through the task execution statement building the summaries.
- d. Locate the specified element on the library and pre-process any control statements associated with the absolute element.
- e. Set up load function in summary
- f. Update job stack bit settings

Function Code 6 - interpret the control statement and transfer control to routine responsible for processing the control statement.

Requestor: Any worker program requesting the processing of a control statement within his primary input stream.

Exits: •Release control card interpreter and transfer control to processing routine. (DRET1\$ B1 = 1 B2 = processing routine).  
 •Release control card interpreter and return control to requestor (DRET1\$ B1 = 0)

Addendum: Requestor task/activity addendum.

Processing Procedure: a) Examine control statement operator.  
 b) Search list of acceptable functions.  
 c) Set up switch to processing routine if not processed by CCI.  
 d) Switch to processing routine.  
 e) If processing is responsibility of CCI process and return control to requestor.

Parameters: B7 = address of control statement (SMOD)  
 Q = length of control statement (SMOD)

## 6.10 Selection

The function of selection is to determine which task or activity from the current pre-selected candidates shall be introduced into the mix of active programs based on selection priority and available facilities. Once an element has been selected it will be allocated and activated by serial executing control statements as formed by pre-selection.

Selection is composed of two phases: Select phase and control card interpretation phase (C.C.I.). Activation of selection is by the following function codes:

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	Internal	Select task or activity from current stack.
02-04	Unassigned	
05	Internal	Initiate selected activity/task
06-33	Unassigned	

## Function Description of Function Codes

Function Code 1: Select a task or an activity from the Job Stack for activation.

Requestor: Pre-selection

Exits:

- If a task is selected establish an activity addendum and storage module or worker addendum requesting the Initiation phase of Selection, and return to the Content Supervisor requesting a return to the exec switches releasing selection.
- If no task selected return to Content Supervisor requesting a release of selection and return to exec switches.

Addendum: Exec task/activity addendum.

Processing Procedure :

- a. Deallocate addendums of inactive tasks placing addendums and held U.C.B. in selection cooperative bit map storage.
- b. Summarize peripherals available on system.
- c. Go through priority chain in the Job Stack selecting a job or appropriate task.
- d. Establish worker addendums in core and switch to worker addendum issuing a request for the Initiation phase.

Parameter: FC 1 EXRN\*20441

Function Code 5: Initiate the task selected.

Requestor: Selection

Exits: Queue request for Selection under exec addendum.

Processing Procedure:

- a. Process the control statements contained in Pre-selection control statement summary using the Control Card Interpreter.
- b. Load absolute element and modify to running form.
- c. Queue a request to start the routine.

Parameter:

FC 5

ENT*Q*W(VO)
EXRN*20445

V) = the address at which the map number  
and start position of the control  
statement summary.

## 6.11 Termination

Termination (library number 10) is a non re-entrant routine responsible for the removal of a task or activity from the system along with its core and peripheral which are not to be held from one task to another. It is activated upon the issue of a RETURN operation by a routine with no outstanding activities or upon the ABORT or ERROR exit of some routine.

Functions:	<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
	01	Internal	A task/activity issues a RETURN with no outstanding activities. Deallocate the appropriate task/activity.
	02	ABORT\$	A task/activity signals the abortion of the job from the system.
	03	ERROR\$	A task has indicated that termination is desired because of error conditions.
	04	Internal	Deallocate the specified unit record routine.
	05	Internal	A task/activity has encountered a fault condition. Abort the job from the system.

### Functional Description of Function Codes

Function Code 01 - A task has completed all functions implying the deallocation of all core and non-held peripherals. The activation of termination was initiated by the basic exec when a RETURN was issued by the routine with no outstanding activities to which control can be given. This is the normal task termination procedure.

Caller: Basic Exec, Return Mechanism

Parameters: Entrance is made by a REGCT function. No additional parameter is required.

Addendum: Worker Task/Activity Addendum

- Function:
- a) Release core assigned to task.
  - b) Release all non-held peripheral of task.
  - c) Throw away non-control statements for task down to first control statement.
  - d) Log all pertinent information for core, peripheral in the Job Description module.
  - e) If the end of the control stream has been reached, deallocate all facilities, deallocate primary input U.R. routine if not already done, clear job library of entries for this job, clear all mass storage locks, console queues, etc.
  - f) Get accounting information from Job Description and submit to primary output.
  - g) Close out primary and secondary output streams.
  - h) If primary output not pre-selected, make return to CSR requesting activation of primary output unit record routine.
  - i) If secondary output not pre-selected and active, make return to CSR requesting activation of secondary unit record routine.
  - j) If output routines active, abort control thread and wait for reactivation through a request for unit routine deallocation.
  - k) If all activities are complete, switch control thread to exec addendum with re-entry to deallocate task and activity addendum of job.
  - l) Make a switch to selection for the selection of another task.

- Exits:
1. ABCT, to wait for unit record routine completion.
  2. DRET\$1, through C.S. to C.S.R. calling or activation of unit record routine.
  3. SWITCT and DRET\$1 requesting the deallocation of the task and activity addendums.

Function Code 02 - A task/activity requests the removal of the entire job from the system.

Caller: Worker Routine Via Exec

Addendum: Worker Task/Activity Addendum

- Function:
- a) Release core.
  - b) Throw away all primary input
  - c) Release all facilities.
  - d) Delete Job library using L.S.R. function code 2
  - e) Clear mass storage locks, etc.
  - f) Put out task accounting information and abort exit condition.
  - g) Put out job accounting information on P.O.

- h) Close out P.O. and S.O., set termination indication.
- i) Put out P.O. and S.O.
- j) Release output unit record routine facilities.
- k) Log all job accounting information.
- l) Switch central thread and go to selection for deallocation of addendums.

Exits: DRET1\$ B1=1 B2=CSR to CSR for pre-selection of output unit record routines.

ABCT: for deallocation of control thread when unit record routines still active.

SWCT and DRET\$ for deallocation of addendums.

Function Code 03 - A task/activity has indicated on termination of the current task is necessary because of an error condition.

Caller: Worker Routine Via Exec

Addendum: Worker Task/Activity Addendum

Function: Essentially the same as Function Code 1

Exits: Same as Function Code 1

Function Code 04 - Deallocate and release facilities of unit record routine.

Caller: Cooperative Service Routine

Addendum: Worker Task/Activity Addendum

Function: a) Release core and facilities of unit record routine.  
b) If task is in process of termination check completions of all functions.

Exits: SWCT to Exec  
DRET1\$ B1=1 B2=Selection

## 6.13 Library Maintenance Service Request

Library number 006 is a non-reentrant routine used to enter an element in the Job Library and to delete elements or portions of elements already in the Job Library. The following is a summary of function codes to Library Maintenance Service Request followed by a description of each function containing parameters required, caller, addendum, exits, and test conditions.

<u>Function Code</u>	<u>Operator</u>	<u>Description</u>
01	Internal	Enter element in Job Library.
02	Delete	Delete specified elements from the Job Library.
03	Internal	Delete all elements associated with the current job from the Job Library.

### Functional Description of Function Codes

Function Code 01 - Enter an element in the Job Library.

Caller: Processor (SPURT, COBOL, etc.)

Parameters: B4 = Address of storage module causing activation biased by lower lock limit.

B7 (in storage module) = Address of processor output TOC.

A Register = Job number of element

A Register (in storage module) = File code of element.

Addendum: Worker task/activity addendum

Function:

- Create Internal TOC for element and enter it in TOC module chain.
- Perform all necessary updating of Job Directory, Bit Map, and TOC Module.
- Transfer element from mass storage to the Job Library.

Exit:

- Return control to content supervisor upon completion.



Function Code 02 - Delete specified elements from the Job Library.

Caller: # Delete statement through the primary input stream.

Parameters: B4 = Address of storage module initiating activity biased by lower lock limits.

B7 = Address to the control statement causing activation.

Addendum: Worker task/activity addendum

Function: The purpose is to release prime mass storage back to library maintenance or to the system. The effected elements are specified on the delete statement in the following format:

# DELETE option NAME/VERSION, ETC.

If the (V) option is used, the name/version are interpreted as group library names. The group library is an extension of the job library. To release this area of storage the function issues a "FREE" statement to the system. Then, the associated library links are removed from the job directory. If no option is present, the elements are interpreted as resident on the job library. The element's area of storage is released to library maintenance and the element's associated TOC is deleted.

Exit: Direct return to requestor.

Function Code 03 - Delete all elements associated with the current job.

Caller:

Parameters: (A) register = Job Number

Addendum: Worker task/activity

The purpose is to return all storage associated to a Job back to maintenance or to the system. The job directory is searched for job and group library links. The links, TOCS, and element area are delete from master storage directory. If a group library is associated to a job, storage is returned to system through a "FREE" statement. Next, the links and job number is removed from directory.

Exit: Direct return to requestor.

### 6.13.1 Processor Interface

All processors operating under the control of OMEGA utilize OMEGA services and functions in the performance of their duties. An explanation of some of the services provided and the method of employing these services follows.

#### •Control Statement Access

In the process of loading and initiating a processor the control statement resulting in the call of the processor is removed from the input control stream. Since all processors read the statement to interpret data contained therein, this card is placed on mass storage via the SENDP\$ operation. The individual processor must issue a corresponding RECIEVEP\$ operation in order to obtain this statement. (See Secondary Exec documentation)

#### •Register Settings

When the individual processor is given control the A register will contain the job number for which the processor is performing its task. RIR is set to the processors base. The lock limits are set to include the core area assigned to the processor. IFR is set to 15 bit B registers.

#### •Available Services

All OMEGA service available to a worker routine are available to the processor.

#### •Option Interpretation

The following is a list of options applicable to individual processors, their meaning and action to be taken because of the option's presence:

Options	- Y = Accept the results of the processing as correct even though errors were detected. The processor would put out an element marked as error free, even though some non-critical error(s) were present. If a critical error occurs which will inhibit the validity of the output, the Y option is ignored and the element marked as being in error.
---------	---

X = Abort the remainder of the job if any errors are detected by the processor. If neither a X or Y option is present and errors are detected, the job will continue, but any attempt to collect and execute the routine will be inhibited unless options are provided on the LOAD and GO statements verifying the validity of the element.

Even though an abort condition occurs the processing will continue in the respect that any listing output will be obtained if possible. No output other than the listing will be produced on the abort condition. The processor will issue an ABORT\$ function which will cause OMEGA to abort the entire job with all incompletd tasks being aborted also.

Z = Error out the task if an error occurs during processing. Any listing requested will be completed, although no element will be produced by the processor. This causes the termination of the current task, but will result in the processing of tasks within the job stream.

The processor will issue an ERROR\$ to initiate this action.

L = Produce a complete listing of the routine being processed. The listing will consist of all pertinent data. Any summarization and error listing will also be given.

N = Suppress the source and object code listing implied by an L option. No information will be printed except for certain error diagnostics.

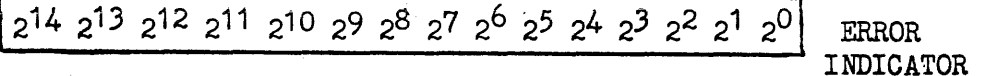
S = Produce a list of symbol definitions which can be used by the diagnostic routine. These symbols will consist of mnemonic source labels within the routine being processed along the relative position of the label.

## •Error Indication

The error indicator in the lower half of the first word of the toc is set to indicate the status of the element processed. Subsequent processors of this element will be influenced by this error indicator along with the processor options as to the action to be taken when an element marked as containing an error is encountered.

The error indicator is broken down into a series of lists, each of which implies a category of error(s) which occurred during the production of the element.

A breakdown of the categories of errors possible and the fields assigned to individual processors concerned with the manipulation of elements. If an error condition is encountered the processing will set the bit in its assigned field which defines the category in which the error falls and the corresponding bit in the general field.



General - General error setting used by all processors

- 2<sup>14</sup> - If set indicates a critical type of error occurred which makes the validity of the element improbable. (e.g. overflow of available storage, loss of data, etc.)
- 2<sup>13</sup> - If set indicates a generative type error which will probably interfere with the collection or execution of the element.
- 2<sup>12</sup> - If set indicates a generative or declarative type error which may or may not interfere with the further processing of the element.

Compiler - Error settings reserved for error conditions encountered during the compilation of the element by the processor (COBOL, FORTRAN, etc.)

2<sup>11</sup> - 2<sup>9</sup>

Assembler - Error setting reserved for error conditions encountered during the assembly of the element.

2<sup>8</sup> - 2<sup>6</sup>

Loader - Error settings reserved for error conditions encountered during the collection of elements.

2<sup>5</sup> - 2<sup>3</sup>

Library Maintenance - Error setting reserved for errors encountered upon the manipulation of the element by the library maintenance routines.

2<sup>2</sup> - 2<sup>0</sup>

•Service Request

The Library Service Routine serves as the means by which an element produced by some processor may be entered into the job library complex, from which it may be operated on or used by other systems or user programs.

The Library Service Routine (LSR) is used to transfer three different types of elements (RB, ABSOLUTE, SOURCE). The description of the TOC header necessary for LSR to process the element may be found in Figure 1 of this section.

The linkage provided by the processor to the LSR is defined below.

ENT*B7*TOC ADDR
ENT*A*FILE CODE
EXRN*LSR1

LRS1 EQUALS\*20601

where TOC ADDR - the base address of the toc which defines the element to be entered into the job library.

FILE CODE - the file code under which the element to be entered into the job library may be found.

After the transfer of the element from the specified file to the job library has been completed the requesting processor will be given control at the address following the request for LSR.

•Termination

The normal method of the termination of any task/activity is the issuance of a RETURN\$ function with no outstanding activities to be completed.

If the options imply an error exit upon sensing an error this is accomplished by the issuing of the following function. No control is returned to the requestor.

ERROR\$

If the options imply an abort condition is called for, this can be accomplished by the issuing of the following function. No control returned to the requestor.

ABORT\$

EXTERNAL TOC FORMATS (PROCESSOR OUTPUT)

RB

		Error Ind		
N	N	N	N	N
N	N	N	N	N
V	V	V	V	V
Increment	to EDEF			
Increment	to XREF	# of EDEF		
Increment	to CC	# of XREF		
Increment	to INFO	# of CC		
Increment	to SDEF	# INFO		
Increment	to TEXT	# SDEF		
Length of	TEXT			

This toc is produced by the assemblies and compilers. It contains a description of where the individual parts of the element are on the scratch file of the processor. Before termination a processor will make a service request calling for library maintenance which will in turn place the element in the job library.

ABS

		Error Ind		
N	N	N	N	N
N	N	N	N	N
V	V	V	V	V
Increment	to ASG Images			
Max Core Used	# Images			
Increment	to ELT Base			
Length	of ELT			
Increment	to SDEF	# SDEFs		
# Segments	Length of Control			

The absolute toc is produced by the collector-loader. The procedure for handling is similar to that for an RB toc in that an internal service request is made for library maintenance.

SOURCE

		Error Ind		
N	N	N	N	N
N	N	N	N	N
V	V	V	V	V
Increment	to ELT			
Sub Type	# Images			
Element	Length			

The source toc could be produced by any processor. Usually it would be concerned with central card images. A sub type is provided whereby the source code could be indicated as compressed. The element length is required in case the sub type indicated compressed, otherwise 20 word images are assumed.

### 6.30.1 Selection Job Stack

0	LARGEST JOB #	# JOBS IN MODULE		
1	INCREMENT TO NEXT JOB STACK			
2	INCREMENT TO BIT MAP MODULE			
3	BASE PRIORITY LINK			
4				
5				
6	BIT SETTINGS			
7	SELECT PRIORITY	PRIORITY LINK	} JOB ENTRY 1	
10	INCREMENT TO JOB DESC. MODULE			
11	MINIMUM CORE	MAXIMUM CORE		
12	LENGTH TASK SUMMARY	MAP #	START POSITION	} JOB ENTRY 2
13				



## Selection Job Stack Description

### •General Description

The selection job stack is maintained in the I/O cooperatives mass storage file. A link to the base job stack module is maintained in the 5th word of the core summary map.

A job number and a cell within the job stack will be assigned by the cooperative service routine when a job card is sensed in the input stream. Although each job cell in the job stack contains the job number, the job number can be calculated by the position of the cell within the module or subsequent modules. Each module is capable of hold 40g job cells.

After the cooperative service routine has set up the job cell in the job stack, along with the establishment of a task addendum, an entry is made to the preselection phase of selection.

Preselection will examine all control statements in the input stream, through the first control statement resulting in an execution request. In examining these control statements preselection will set up a facility summary consisting of all facilities requested in the control stream and by the routine to be executed. This summary is preserved in a storage module on the cooperative mass storage file. It will be used by selection in determining if the job can be loaded and executed within the mix of jobs in the system.

•Detailed Description

- Word 0 - U = The largest job number that has been assigned.
- L = The current number of job entries in this particular job stack. The maximum is 60 octal for any one stack module.
- Word 1 = A link to the next module of the job stack if one is present. Subsequent modules will have the same format of the initial one except for the sign bit of word 0.
- Word 2 = A link to a module used for storage of bit maps describing areas taken up by selection summaries.
- Word 3 = Base link to job of highest priority
- Word 6 = The start of a job entry. The bit settings indicate the status of the job at any one time.

Bit Settings	Meaning
229	JOB terminated, clear from system
228	End of control stream reached
227	Primary input called
226	" " preselected
225	" " selected
224	" " initiated and active
223	Primary output called
222	" " preselected
221	" " selected
220	" " initiated and active
219	Secondary output called
218	" " preselected
217	" " selected
216	" " initiated and active
215	Edit Routine called
214	" " preselected
213	" " selected
212	" " initiated and active
211	
210	
29	
28	
27	
26	
25	
24	Task rolled out
23	" active
22	" initiated
21	" selected
20	" preselected

Word 7 - U = The select priority which is assigned according to the following conditions.

- a. Priority specified by user.
- b. The number of tasks within the job that have been completed.
- c. The amount of primary input
- d. The number of peripherals required by the task.
- e. The number of times a task has been passed up.

- L = The job number assigned by the cooperative service routine.

Word 10 = The increment to the job description module on the cooperative library file (ZD).

Word 11 - U = The minimum amount of core the task can use if it is available.

- L = The maximum amount of core the task can use if it is available.

Word 12 - U = The length of the initial task summary.

- L = Link to job of next highest priority

$2^{14}-2^6$  = The map number which defines the base storage module used for summary storage.

$2^5-2^0$  = The bit position defining the base sub cell used within the storage module.

6.30.2 Job Description Module

HEADER

0	INCREMENT TO NEXT JOB DESC. MODULE OR TO ITSELF	
1	MODULE LENGTH	INDEX TO NEXT AVAIL. LOC.
2	FILE INCREMENT TO CURRENT MODULE	
3	LENGTH TASK ADDM & UCB	MAP # AND START
4	LENGTH CONSTANT ACCT. INFO	INDEX TO CONSTANT ACCT INFO
5	FILE INCREMENT TO MODULE WITH CONST. ACCT. INFO	
6	LENGTH JOB IDENTIFICATION	INDEX TO JOB I.D.
	FILE INCREMENT TO MODULE WITH JOB I.D.	
	LENGTH PRIMARY INPUT U.R. INFO	INDEX TO P.I.U.R. INFO
	FILE INCREMENT TO MODULE WITH P.I.U.R. INFO	
	LENGTH PRIMARY OUTPUT U.P. INFO	INDEX TO P.O.U.R. INFO
	FILE INCREMENT TO MODULE WITH P.O.U.R. INFO	
	LENGTH SECONDARY OUTPUT U.R. INFO	INDEX TO S.O.U.R.
	FILE INCREMENT TO MODULE WITH S.O.U.R. INFO	
	LENGTH EDIT ROUTINE INFO	INDEX TO EDIT ROUTINE INFO
	FILE INCREMENT TO MODULE WITH EDIT INFO	
	LENGTH OTHER ROUTINE INFO	INDEX TO OTHER ROUTINE INFO
	FILE INCREMENT TO MODULE WITH OTHER INFO	
	LENGTH ACCOUTING INFO	INDEX TO ACCOUTING INFO
	FILE INCREMENT TO MODULE WITH ACCOUNT INFO	
	LENGTH DUMP INFO	INDEX TO DUMP INFO
	FILE INCREMENT TO MODULE WITH DUMP INFO	
	LENGTH CHANNEL Ø SUMMARY	INDEX TO CHANNEL Ø SUMMARY
	FILE INCREMENT TO MODULE WITH CHANNEL Ø SUMMARY	
	)	
	LENGTH CHANNEL 23 SUMMARY	INDEX TO CHANNEL 23 SUMMARY
64	FILE INCREMENT TO MODULE WITH CHANNEL 23 SUMMARY	

JOB I.D.

A

JOB IDENTIFICATION

P.I.U.R

B

(JOB STATEMENT FROM INPUT STREAM)  
LENGTH INPUT U.R. SUMMARY                      MAP # & START OF INPUT U.R.  
PRIMARY INPUT UNIT RECORD ROUTINE

P.O.U.R

C

CALL CONTROL INFORMATION  
LENGTH PRIMARY OUTPUT U.R. SUMMARY    MAP # & START OF P. OUTPUT U.R.  
PRIMARY OUTPUT UNIT RECORD

S.O.U.R

D

ROUTINE CALL CONTROL INFORMATION  
LENGTH SEC. OUTPUT U.R. SUMMARY        MAP # & START OF S. OUTPUT U.R.  
SECONDARY OUTPUT UNIT RECORD

EDIT R.

E

ROUTINE CALL CONTROL INFORMATION  
LENGTH EDIT SUMMARY                      MAP & START OF EDIT  
EDIT ROUTINE CONTROL

STATEMENT INFORMATION

OTHER R.

F	LENGTH OTHER SUMMARY	MAP # & START OTHER
	OTHER ROUTINE CALL	

CONSTANT ACCT.

GO

	CONTROL INFORMATION	
	DATE OF JOB ENTRANCE	
1	TIME OF JOB ENTRANCE	
2	HR:	MN: SEC
3	TOTAL CP UNITS USED BY JOB	
4	AMOUNT PRI INPUT	AMOUNT PRI OUTPUT
5	AMOUNT SEC. OUTPUT	
6	DATE OF TASK SELECTION	
7	TIME OF TASK SELECTION	
10	HR:	MN: SEC

CHANNEL SUMMARY

HO

	P-TYPE	
1	TOTAL STORAGE OR # UNITS ASSIGNED	
2	TOTAL TIME PERIPHERAL HELD	
3	TIME OF LAST ASSIGNMENT	
4	HR:	MN: SEC
5	AMOUNT OR # UNITS LAST ASSIGNMENT	

## Job Description Module

### •General Description

The job description module is linked off the job stack entry for a particular job. Each job has a separate job description module or chain of modules. The storage for the modules is obtained from the cooperative library file (ZD). The job description module contains unit record routine descriptions, job identification, and accounting information associated with the job. The module is originally produced by the cooperative service routine upon processing a JOB control statement. Entries and modifications to the module are made during preselection, selection, facility assignment, facility release, and termination.

•Detailed Description

- Word Ø Increment to the next job description module in the chain. If only one module exists the link is to itself. Each module in the chain has a link as the first word of the module.
- 1 U The length of the I/O cooperative module being used. Normally this is 306g.
- L The index relative to the base of the module currently being used for deposit of information.
- 2 The file increment to the module currently being used for deposit.
- 3 U The length of the task addendum and unit control block stored.
- L The map number and start position of the storage used for the task addendum. See ( ) for function of bit map storage facility.
- 4-64 Each two word entry in the rest of the header defines the location of some information. The upper defines the length of the data, the lower defines to index relative to the base of the module containing the information. The second word gives the file increment to the module containing the information. This increment may be to the same module containing the header information.
- A The job identification normally consists of the job control statement from the primary input stream.
- B The primary unit record routine information consists of a link through the selection list map to a summarization of the unit record routine requirements for facilities, core, etc. The control statement calling for an input unit record follows the summary link. The summary link is provided by preselection. The control statement is added by the cooperative service routine for processing by preselection.
- C Same as B except for primary output unit record routine.
- D Same as B except for secondary output unit record routine.
- E Same as B except for the edit routine.



- F Same as B except for other routines.
- Go The date upon which the job was entered into the system.
- 1-2 The time in hours, minutes, and seconds which the job was entered into the system.
- 3 The total central processor time units (200us) used by the job. This is a cumulative total for the whole job.
- 4-U The number of cooperative modules used for the primary input.
- L The number of cooperative modules used for primary output.
- 5-U The number of cooperative modules used for secondary output.
- L Spare
- 6 The date upon which the current task was selected.
- 7-10 The time in hours, minutes, seconds at which the task was selected.
- Ho -U The peripheral type of the units on this channel.
- L Unassigned
- 1 The total storage or number of units assigned during the job.
- 2 The cumulative time that the peripheral was held.
- 3-4 The time at which the last assignment was made.
- 5 The amount of storage or the number of units assigned.

6.30.3 PRESUM MODULE

0	LINK NEXT MODULE			
1	# WORDS			
2	PERIPHERAL NAME			
3	# UNITS REQUIRED	MASS STORAGE REQUIRED		
4	# ENTRIES	LENGTH SEL SUMMARY		
5	P-TYPE	CHAN	CHAN	LENGTH UNIT SUMMARY
6	MASTER BIT UNIT			
7	MASK			
10	P-TYPE	CHAN	CHAN	LENGTH UNIT SUM <sup>O=MS</sup>

## Presum Module Description

The Presum Module is the initial module linked off the job stack, or the job description in the case of a unit record or edit routine summarization. It contains the information necessary for the selection routine to determine if a routine can be loaded.

The Presum Module is produced by pre-selection from the Assign statements encountered in the job control stream. It is maintained on the cooperative library file (ZD), through the bit map module. The storage used by the summaries is released back into the bit map once the task has been relisted and initiated.

Word  $\emptyset$  = Link to another module if one is present. The link is broken down into a 6 bit start position and a 9 bit map number referring to the bit map module. The sign bit is set if the link is to another presum module and not set if the link is to the facility summary.

Word 1 = Number of words within the next summary module.

Word 2 = The start of a summary cell. The name is a 5 character name of the peripheral specified on the Assign statement.

Word 3 = Bits  $2^{29}$ - $2^{24}$  - the number of individual units requested under the peripheral name.  
Bits  $2^{23}$ - $2^0$  - the amount of mass storage requested.

Word 4 = U = the number of channel entries in the channel summary.  
L = length of channel summary for this name.

Word 5 =  $2^{29}$ - $2^{25}$  = Peripheral type  
 $2^{24}$ - $2^{20}$  = Channel number  
 $2^{19}$ - $2^{15}$  = Channel number  
 $2^{14}$ - $2^0$  = Length of unit summary for this channel.

Word 6-7 = Master bit setting for those units on the channel which would be acceptable for assignment.

Word 10 = Indicator for mass channel.

Words 2-10 represent a sample entry. The entries will vary with the different peripheral names. All words except for word 2 are obtained from the selection facility summary map which is a congestion of the regular facility map.

BIT MAP MODULE FOR LINKING SELECTION SUMMARIES

0	MODULE LENGTH		
1	LINK TO NEXT BIT MAP MODULE		
2		# USED CELLS	
3			
4			
5			
6	INC TO STORAGE MODULE		
7		# FREE SUB CELLS	} STORAGE CELL 1 (MAP #1)
10	SUB CELL BIT SETTINGS		
11	INC TO STORAGE MODULE		
12		# FREE SUB CELLS	} STORAGE CELL 2 (MAP #2)
13	SUB CELL BIT SETTINGS		
304			
305			

## Bit Map Module Description

### . General Description

Each bit map module has a 6 word header portion which defines the link to the next bit map module if there is any additional modules, and the number of free and used storage cells within that module. There is a maximum of  $100_8$  storage cells within one module. Each storage cell has a map number dictated by its position within the module, for example words 6-10 of the initial bit map module would be referred to as storage cell one with a map number of 1.

Each storage cell defines a  $306_8$  word module on the cooperative mass storage file, which is used for the storage of selection summary data, etc. Each of the storage modules have a 2 word header and  $204_8$  words of storage which group (sub cell) is represented by a bit setting in the third word of a bit map storage cell. As a sub cell is filled with data the appropriate bit in the bit map storage cell is set to a binary one.

As the summary information is processed by selection the cells are freed for use again. If a complete module is empty it is released back into the cooperative pool.

The bit map module is maintained on the cooperative library file (ZD).

### . Detailed Description

- Word  $\emptyset$  = U - The length of the module containing the bit map.
- 1 = A link to another bit map module or  $\emptyset$  if no other module exists.
- 2 = U - The number storage cells used within this bit map module.
- 3-5 = Free
- 6 = Is the start of the first storage cell. It contains an increment to a storage module to be used for storage. The storage module to be used for storage. The storage modules are obtained from the cooperative library file (ZD).
- 7 = The number of free sub cells within the indicated module. Each subcell consists of 7 words.
- 10 = A series of bit setting representing subcells within a storage module. Each bit  $2^{29}-2^2$  represents a 7 word subcell in a storage module. The bits are set as the storage is used.
- 11-13 = Another storage cell defining another  $306_8$  word of storage.

Selection FACILITY SUMMARY MODULE

0	MAP & START OF NEXT	
1	# WORDS	
2	# WORDS	EXRN ENTRY
3	CONTROL  INFORMATION	
X	# WORDS	EXRN ENTRY
	CONTROL  INFORMATION	
Y	# WORDS	EXRN ENTRY
Y+1	FILE CODE	LENGTH OF CONTROL
Y+2		
Y+3	FILE INCREMENT TO ELEMENT	
Y+4		# SEGMENTS

## Selection Facility Summary Module Description.

### . General Description

The selection facility summary is linked off the pre-selection module and contains all control statement summarizations which will be processed upon selection of the associated task. All parameters necessary are contained in the selection facility summary including information for the load of the routine.

The selection facility summary is produced by pre-selection and used by the initiation phase of selection. The storage used for the summary is released upon the selection of the routine summarized. The modules reside in the cooperative library file (ZD).

### . Detailed Description

- Word  $\emptyset$  = The link to the next module of control summaries or  $\emptyset$  if there isn't any more modules.
- 1 = The number of data words contained in this module of control information.
- 2 = U = The number of words occupied by this control statement summarization.
- L = The lower half of the exec return instruction necessary for the entry to the routine responsible for processing the control statement.
- 3 to N= The control statement information whose address is conveyed to the processing routine. The control statement may be an ASG, LOG, etc.
- X to Y= Another control statement. There may be any number of separate control statements.
- Y = The last entry in the selection facility summary which contains the information necessary to load the selected routine.
- U = The number of words in the load entry.
- L = The exec return necessary to process the load.
- Y+1 = U = The file code which is to be used in the loading of the element.
- L = The length of the control part of the element to be loaded. This is equivalent the number of words to read in loading the element.

- Y+2 = Reserved for the base address of the routine to be loaded.
- Y+3 = The increment relative to the base of the file indicated in word Y+1, at which the initial instruction of the routine is located.
- Y+4 = U = Unused

L = The number of sub-segments contained in the routine being loaded.

After the read of the routine from mass storage the segment descriptors (see Loader documentation) will be updated to put the loaded element in operating order.

The parts of the segment descriptor modified are, the file code which is replaced by the file code used for the routine load from mass storage. The core bases of all segments will be modified by the base of the routine, the file increment to the subsegments will be modified by the base of the routine on mass storage.



6.30.4 Selection Facility Map

0	PERIPHERAL NAME 0						
1	# ENTRIES				LENGTH SEL SUMMARY		
	29	25	24	20	19	15	14
2	P-TYPE		CHAN		CHAN		LENGTH UNIT SUMMARY
3	MASTER BIT						
4	UNIT MASK						
	29	25	24	20	19	15	14
X	P-TYPE		CHAN		CHAN		LENGTH UNIT SUMMARY
X+1	MASTER BIT UNIT SUMMARY						
Y	PERIPHERAL NAME 1						
Y+1	# ENTRIES				LENGTH SEL SUMMARY		
	29	25	24	20	19	15	14
Y+2	P-TYPE		CHAN		CHAN		LENGTH UNIT SUMMARY
	MASTER BIT UNIT MASK						
Z							

•Detailed Description

Word $\emptyset$	The five character name used for the assignment of the peripheral.
1	U - The number of variable length entries that appear under this peripheral name. There is one entry for each channel that contains the peripherals that could possibly satisfy the routines request. In the drawing words 2 to X, X to Y are separate entries under this peripheral name.
2	$2^{29}-2^{25}$ - A five bit P-TYPE that defines the particular type of equipment on that channel. $2^{24}-2^{20}$ - This field will be blank except when dual channel is utilized in which case it is a five bit channel number upon which the units are located. $2^{19}-2^{15}$ - A five bit channel number upon which the units are located. $2^{14}-2^0$ - the length of the unit summary mask contained in words 3 to X.
3	A variable length summary mask with one bits in the locations representing units which could satisfy the request. This mask will be applied against the master bitted units for this channel in the core facility and storage summary during selection to determine if sufficient unit are available for the selection of the task. If the channel has mass storage equipment on it, the master bit unit mask is omitted.
X	Defines another channel which contains units capable of satisfying the request of the routine being summarized. Description is the same as word 2.
X+1	Master bitted units for alternate channel. Description is the same as for word 3,4.
Y-Z	A summarization of the units acceptable under another peripheral name.

•Selection Facility Map Description

•General Description

The selection facility map is a congested version of the facility map. It contains all the peripheral names and unit specifications that are used for the assignment of units. The portion of this map under a particular peripheral name used on an assign statement is included in the preselection summary to enable the selection routine to determine if sufficient units are available to satisfy a routine requirements.

The selection facility map is produced during initialization and used by preselection. It resides on the system library file (ZE). The file increment to the selection facility map and its length are contained in the core facility and storage summary (see ) whose address is passed in B5 by the content supervisor.

## 7.0 I/O Cooperative Mechanism

The Input/Output cooperative mechanisms are the system elements by which: OMEGA retrieves all scheduling information, in the form of control streams; and submits accounting and actions taken by OMEGA as the result of processing schedule parameters.

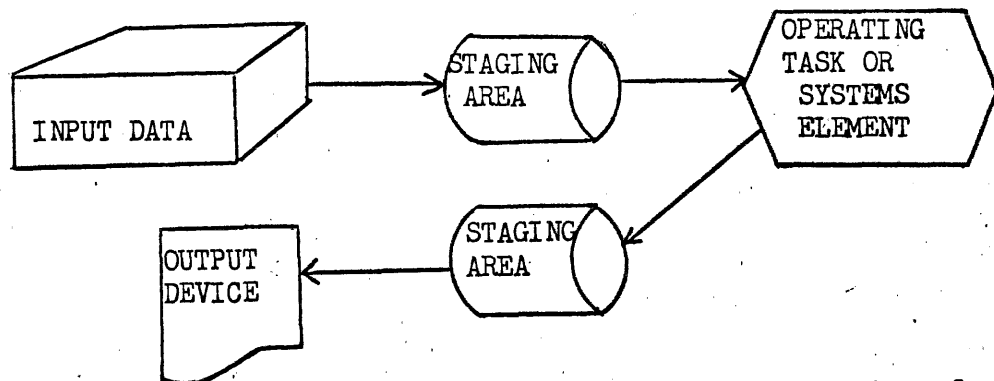
The cooperative mechanism has been designed and implemented to control the two way transmission of three streams defined as follows:

- Primary input - device card, tape, drum remote - used to contain OMEGA schedule parameters, limited data, source code to assemblers and/or compilers, program parameters, etc.
- Primary output - Hard copy of program scheduling results, listing from assemblers and/or compilers, limited data, etc. Device normally high speed printer, Univac 1004, remote devices.
- Secondary output - device card, tape, drum remote used to contain assembler and/or compiler, object code and limited data.

## 7.1 Cooperative Features

The Input/Output cooperative mechanism provides the user and system with the following advantages and features:

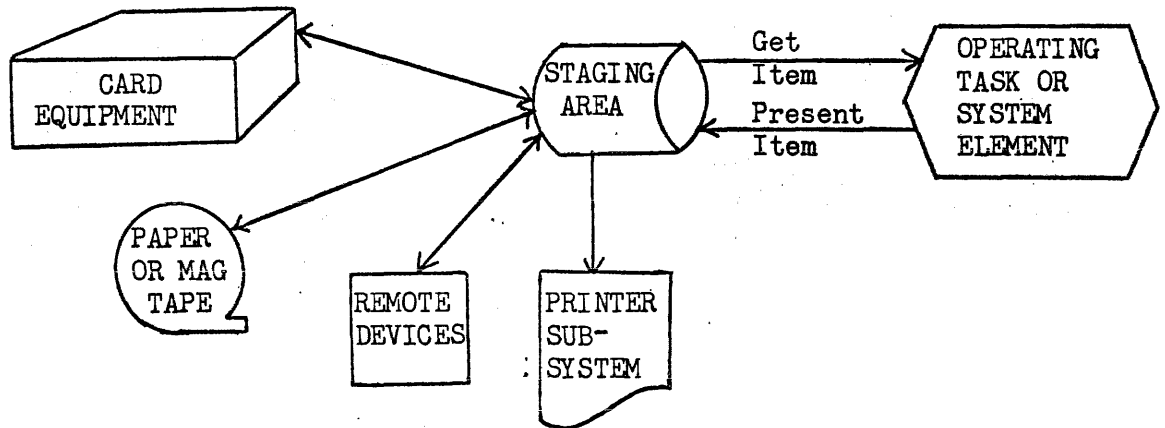
- The staging of low speed Input/Output data to mass storage to balance intermittent system utilization with the slow rate peripheral devices.



Illustrates two way transmission of stream.

Staging allows the device to operate at full capacity within the controlled constraints of the staging area. The buffering to mass storage permits the parallel utilization of low-speed devices by operating tasks in a multi-program environment.

- Provides OMEGA, compilers and/or assemblers, other system elements and the user programs a consistent mechanism, independent of device to obtain and/or submit data. This feature purges system elements of redundant code required to assign, recognize and handle varied devices.



Illustrates device independent worker interface

- A pooled staging area is maintained by I/O cooperative control allowing multiple streams of primary and secondary data streams to utilize a shared library. Data for any one stream is threaded by chains to the task. Cooperative control expands and contracts the "cooperative library" as required to maintain the system. In addition to expansion of mass storage, I/O cooperative control will invoke temporary suspension and/or Roll-out, Roll-in procedures to control overflow conditions.

## 7.2 I/O Cooperative Elements

The elements required to effect the Input/Output cooperative mechanism are described as follows and their interaction is depicted in Figure 7-1.

Input Unit Record Routine - individual routines responsible for reading data images from assigned device and forming these into a buffer the size of which is determined by assigned size of storage modules used for buffering. Upon completion of a buffer (module), unit record routine submits module for staging by cooperative control.

I/O Cooperative Control - is a basic systems element, normal resident, responsible for: staging and chaining, Input/Output to mass storage, recognizing overflow conditions and calling the "Cooperative Service Routine" for action, processing requests from operating program and/or system elements for primary and secondary Input/Output.

Primary/secondary Output Unit Record Routine - individual routines responsible for recording, punching or transmitting items contained in output streams.

Cooperative Service Routine - is a secondary exec element, normal drum stored and responsible for the following with regard to cooperative mechanism.

- Activate selection and load process of unit record and edit routines.
- Initialize and terminate drum chains.
- Allocate additional mass storage and/or otherwise control cooperative library.
- Perform error recovery for cooperative control.

Edit Routine - individual routines called by "SOURCE" statement to merge and delete supplementary data into primary control stream. Normally used to apply correction cards against an existing source data file for assemblers and/or compilers. May be modified to merge and/or update normal data files being entered through the primary input stream.

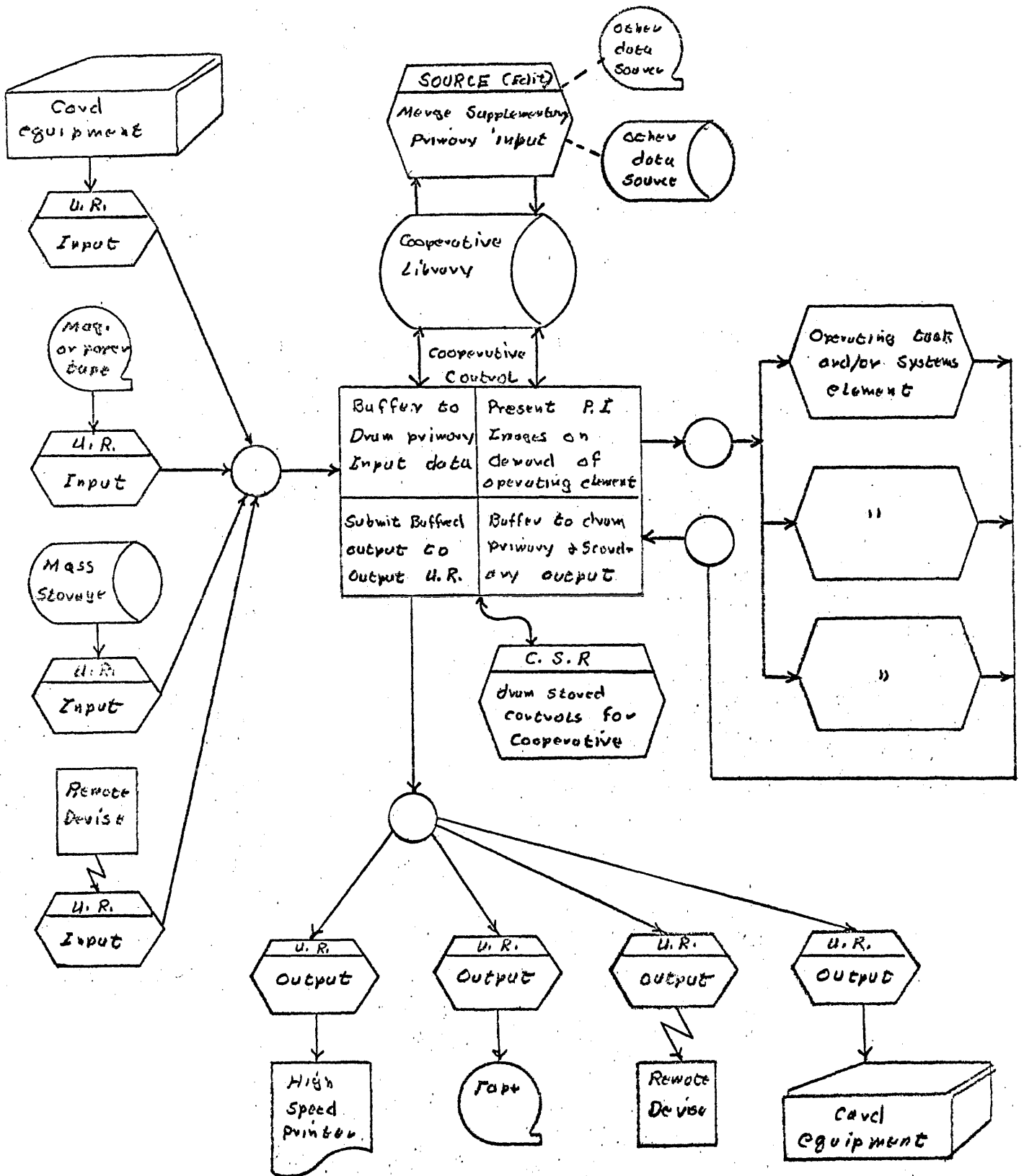


Figure 7-1

### 7.3 Input Unit Record Routine

Input unit record routines are normal worker program elements responsible for reading and forming into storage modules data to be used as primary input. Each primary input U.R. is programmed to handle a particular device or type of devices and is loaded and activated by "Cooperative Service Routine" by one of the following:

- Console operator type-in to OMEGA

UR Name/Version

where: Name/Version is that of the unit record routine being requested. Absence of name/version implies the normal systems primary input unit record routine.

- Internal or external "START" control card. The "START" statement identifies a pre-stored job stream contained on mass storage which is to be scheduled as an independent JOB. (see
- An unsolicited communications interrupt on a device assigned to the system in which the first message contained "FROM" control statement. (see

The sequence of functions performed is as follows and illustrated in Figure 7-2.

- a) Read items using file code ZA from assigned device until a JOB control card is encountered. Pack Job card and subsequent images into drum module (Figure 7.5-1) as items deleting full words of trailing space codes from each image. Upon completion of a drum module perform following call to the cooperative service routine to set up proper chain descriptions and store module.

```
ENT*B7*Module buffer descriptor  
EXRN*2 0 3 4 3
```

- b) Read and pack items into drum module as in step (a) and submit completed modules to I/O cooperative control with the following call. All error status codes for image read may be stored in the upper of item length descriptor and will be passed onto requesting program.

```
ENT*B7*Module buffer descriptor  
EXRN*3 0 0 0 4
```

Repeat above until a JOB or FIN control card is encountered. JOB card go to step (c). FIN card or end-of-file go to step (d).



- c) Submit current drum module with an adjusted item count excluded JOB card to a "Cooperative Service Routine" with the following call which will cause primary input stream closure.

```
ENT*B7*Module buffer descriptor  
EXRN*2 0 3 4 4
```

Upon return of control move JOB card as first item in a drum module and go to step (a) with JOB card.

- d) Submit current drum module with adjusted item count to exclude FIN card to "Cooperative Service Routine" with the following call.

```
ENT*B7*Module buffer descriptor  
EXRN*2 0 3 4 5
```

Upon submission of this call control will not be returned, primary input stream will be closed and routine will be deallocated and terminated.

Input unit record routines are designed and programmed under the following constraints.

- Assignment of input device used to read images is by normal "ASG" control card collected with the unit record routine object code element via the LOADER.
- All U.R. routines are activated and terminated via "Cooperative Service Routine (CSR)" and may not FORK or be fragmented.
- Each routine contains its drum module and description used to present items to I/O cooperative control. It is assumed each routine will delete full words of trailing space codes from each image to compact items in drum module.
- Each routine is initially activated at its first location.

### 7.3.1 Sample Input Unit Record

7.4 Cooperative Control is responsible for the following functions:

- Maintain a poll of drum modules used to retain primary input, output, and secondary output. In performing this function cooperative control maintains chaining of modules to the task addendum for each stream, recognizes overflow conditions and drum read/write errors.
- Provides the interface with the exec for unit record routines used in reading primary input or writing primary and secondary output.
- Process service requests for primary input images and/or submissions of primary and secondary output by operating worker programs or system element. To perform this function cooperative control maintains core buffers used to read staging module into core for the transfer of images and verification of worker parameters.

7.4.1 Calls

Cooperative control is a passive exec element activated via EXRN\*300FC. When activated cooperative control attains the identity of the requesting task/activity addendum. The following is a summary and functional description of each function code recognized cooperative control.

<u>Function Code</u>	<u>Call</u>	<u>Description</u>
00	Internal	Call for next primary input image
01	CARD\$	Call for next non control primary input image
02	PRINT\$	Submit primary output image
03	PUNCH\$	Submit secondary output image
04	Internal	Submit primary input module
05	Internal	Request next primary or secondary output module
06	Internal	Request drum module from cooperative library
07	Internal	Release " " " " "
10	Internal	Request release of all cooperative core buffers
11	Internal	Store last drum module for Job's primary input
12	Internal	Store last drum and close primary or secondary stream

Function code 00 - Transfer next image, control or data, contained in Job's primary input stream to the requestor.

Caller: Restricted to use by OMEGA system elements in order to process control statements used in the scheduling and execution of tasks.

Parameters: B7 = to base address of the buffer to which image is to be transferred.

Function: •Determine existence of an image. If none available PUSH request until unit record routine supplies another module. If stream is closed and empty return end-of-real status.

- If required, locate core buffer and read drum module from head of primary input chain. If drum error occurs call cooperative service routine with F.C. 22.

- Transfer image to requestor and update item counts.

- Deallocate drum module and adjust chain links if module was emptied due to request. Return control to requestor.

**Exit:** The following status conditions are possible upon return of control to requestor:

- "A" register = 0000000000 indicate normal completion. "Q" register will be set to number of words contained in transferred image.

- "A" register = 7777740002 - implies deposit address specified by B7 plus image length are outside the program lock bounds of the requestor.

- "A" register = 7777740003 - unrecoverable drum error was encountered for the image that was transferred. Image may or may not be valid or complete. In the event drum chaining was in question primary input stream will be deallocated and subsequent requests will result in an 05 status.

- "A" register = 7777740005 - end-of-primary input stream has been reached and deallocated. Upon subsequent requests of primary input the 05 status will be returned to the requestor for a total of three times. If a fourth request is made the following diagnostic message will be submitted to the primary output stream and the "ERROR" return exit will be executed for the activity. Diagnostic:

END OF PRIMARY INPUT STREAM LOOP

Function Code 01 - transfer next non control image contained in Job's primary input stream to the requestor.

**Caller:** Operating task or system elements for the retrieval of source language statements to compilers and/or assemblers, parameters and limited data.

**Parameters:** B7 = to base address of the buffer to which image will be transferred.

**Function:** •Determine existence of an image. If none available "PUSH" request until unit record routine supplies another module. If stream is closed and empty, return end of primary input status. 05

- If required locate core buffer and read drum module from head of primary input chain. Call "cooperative service routine" with FC 22 if drum error.
- If item is not a control statement (#) in position one, transfer item to requestor and update item counts.
- Deallocate drum module and adjust chain links if module was emptied due to request. Return control to requestor.

Exit: The following status codes are possible upon return of control to requestor.

- A register = 000000000 - indicates successful completion of request. "Q" register contains number of words in the image transferred.
- A register - 7777740002 implies address specified by B7 plus image length, are outside the program lock bounds of the requestor.
- A register = 7777740003 unrecoverable drum error was encountered for the image that was transferred. Image may or may not be valid or complete. In the event drum chaining was in question primary input stream will be deallocated and subsequent requests will result in an 05 status.
- A register - 7777740004 - indicates next image in the primary input stream is a control card (#) sign in first position. This feature can be used to indicate end of data cards for this task. Upon subsequent requests via Function code 01 the 04 status will be returned to requestor for a total of three times. If a fourth request is made an "END OR PRIMARY INPUT FILE LOOP" diagnostic will be submitted to the primary output stream and the "ERROR" return exit will be given for the activity.
- A register - 7777740005 - end of jobs primary input stream. The primary input descriptor is deallocated. Upon subsequent requests of primary input the 05 status will be returned to the requestor for a total of three times. If a fourth request is made an "END OF PRIMARY INPUT STREAM LOOP" diagnostic message will be submitted to the primary output stream and the ERROR return exit will be given for the activity.

Function Code 02 - requests the described print image be submitted to the primary output stream. This function is applicable to worker programs and system elements for the submission of accounting, diagnostic, scheduling messages and limited user hard copy.

Caller: Operating task and/or systems elements.

Parameters: B7 = to base address of image relative to PLR.  
Q = number of words contained within the image.  
A = number of lines to space paper before print.  
. Systems primary output unit record routines recognize 778 or greater as a skip to next page.

Function: •Validate requestors parameter limits

- If required locate core buffer and load an existing incomplete module from tail of primary output stream descriptor.
- Transfer image to buffer module deleting full words of trailing space codes in order to conserve drum storage. If image cannot be contained in the drum module allocate next link in the chain and store full module, transferring image into new drum module.
- Upon storing a drum module perform the following tests:

Test for task/activity exceeding estimated number of pages specified on the Job control card. If exceeded, call "cooperative service routine"(CSR), function code (21) CSR will: submit "PRIMARY OUTPUT STREAM OVERFLOW" diagnostic message to primary output, reset overflow indicator and perform an "ERROR" exit under the current activity addendum.

Check for an outstanding PUSH request from a primary output U.R. routine selected to process the chain.

Check number of modules contained in this chain against the maximum set for the installation, if equal, call C.S.R. with function code 20 for some disposing action. Maximum set for distributed version of OMEGA allows for a 3 to 5 minute backup of print images at 600 lines per minute.

Exit: The following status codes are possible upon return of control to requestor.

- A register = 0000000000 indicates successful completion of the request.
- A register = 7777740002 implies that the base address plus image length are not within the program lock limits of the requestor.

Function code 03 - requests the described image to be submitted to the secondary output stream. The secondary output stream is normally thought of as card punch, paper tape or some slow speed mechanical punch device.

Caller: An operating task/activity. Applicable to worker programs and for the submission of compiler and/or assembler source or object code.

Parameters: B7 = to base address of image relative to PLR  
Q = number of words contained in the image

Function: •Validate requestors parameters

- If required, locate drum module buffer, descriptor and load an existing incomplete module from tail of secondary output stream.
- Transfer image to module buffer deleting full words of trailing space codes to conserve drum storage. If image cannot be contained in drum module, allocate next drum link and store completed module transferring requested item into new buffer.
- Upon storing of a completed module perform the following tests.

Test for task/activity exceeding estimated number of images specified on the Job control statement. If exceeded call C.S.R. via F.C. 21 which will perform the following: Submit "SECONDARY OUTPUT STREAM OVERFLOW" diagnostic message to the primary output stream. Reset overflow indicator. Perform "ERROR" exit under the current activity addendum.

Check for an outstanding PUSH request from a secondary output U.R. routine selected to process the chain.



Test number of modules contained in this chain against maximum set for installation. If equal call C.S.R. with function code 20 for some disposing action. Maximum set for distributed version allows for a 3 to 5 minute backup of 16 word images at 200 images per minute.

**Exit:** The following status codes are possible upon return of control to requestor.

- A register = 0000000000 indicates successful completion of the request.
- A register = 7777740002 implies that the base address plus image length are not within the program lock limits of the requestor.

Function code 04 - requests the chaining and storing of indicated drum module to the tail of the primary input chain descriptor contained in the requestors task addendum.

**Caller:** Primary input unit record routine.

**Parameters:** B7 = address of drum module descriptor (see 7.5.2)

- Function:**
- Validate that the drum module and descriptor are within the requestors program lock limits and buffer does not exceed  $198_{10}$  words.
  - Allocate drum link, adjust chain addresses and module counts.
  - POP any outstanding item requests.
  - Store module in cooperative library and return control.

**Exit:** The following status conditions are possible upon return control to the requestor.

- A register = 0000000000 indicating successful completion of the request.
- A register = 7777740002 implies request did not pass PLR validation tests.
- A register = 7777740003 implies invalid parameter or buffer exceeded  $198_{10}$  words.

Function code 05 - requests the delinking, allocating and transfer of the next module from the head of primary or secondary output stream.

**Caller:** Primary and/or secondary output unit record routines.

Parameters: B7 = address of buffer module descriptor  
B6 = contains indicator as to which chain (0) primary  
(1) secondary.

Function: •Validate buffer module descriptor.

•PUSH request if no modules available.

•READ drum module into described buffer.

•Deallocate chain link, update module counts and adjust addresses. Return control to requestor.

Exit: The following status conditions are possible upon return of control to requestor.

•A register = 0000000000 indicating successful completion of request.

•A register = 7777740002 implies buffer descriptor violates program lock limits.

•A register = 7777740005 end of primary or secondary output stream.

Function code 06 - requests 198 words from cooperative library.

Caller: Restricted to use by OMEGA systems routines.

Parameters: None

Function: Retrieve drum module from cooperative allocation routine. If modules unavailable request will be PUSHed to "cooperative control" until module becomes available.

Exit: Upon return of control "A" register contains logical increment of drum module relative to ZD file code.

Function code 07 - complement of Function code 06; requests the release of a 198<sub>10</sub> words of mass storage procured from the cooperative library.

Caller: Restricted to use by OMEGA system elements.

Parameters: A register = to logical increment of mass storage being released.

Function: Release drum module to cooperative and "POP" any outstanding requests for cooperative mass storage.

Exit: A register = 000000000 implies request complete.

Function code 10 - deallocate all outstanding core buffers linked to task addendums.

Caller: OMEGA systems element performing a compaction of core function.

Parameters: None

Function: Follow task addendum chain to perform the following:  
Locate allocated core buffer containing an incomplete buffer module and descriptor. Restore to drum each one located and release the core storage.

Exit: A register set to 000000000 indicating completion of the function.

Function code 11 - requests the storing of indicated drum module to the tail of primary input chain descriptor contained in the requestors task addendum.

Caller: Cooperative service routine when activated via 04 or 05 function code.

Parameters: B7 = address of drum module descriptor (see 7.5.2).

Function: •Store module described by B7 pointer and update module counts.

•"POP" any outstanding image requests for this chain.

Exit: Normal return to requestor A register set o 000000000.

Function code 12 - request storing of last drum module, deallocation and closure of input phase of primary or secondary output stream.

Caller: OMEGA systems element during termination process of a Job or task.

Parameters: B2 register set to (0) indicating primary output stream is to be closed or B2 set to (1) indicating secondary output stream is to be closed.

Function: •If required located core module buffer, descriptor and load existing tail of primary or secondary stream.

•Store 7———7 as logical drum increment to next drum module. Record drum module to mass storage and update module counts. Check for any outstanding

request for the module from output unit record routine. If found, POP request.

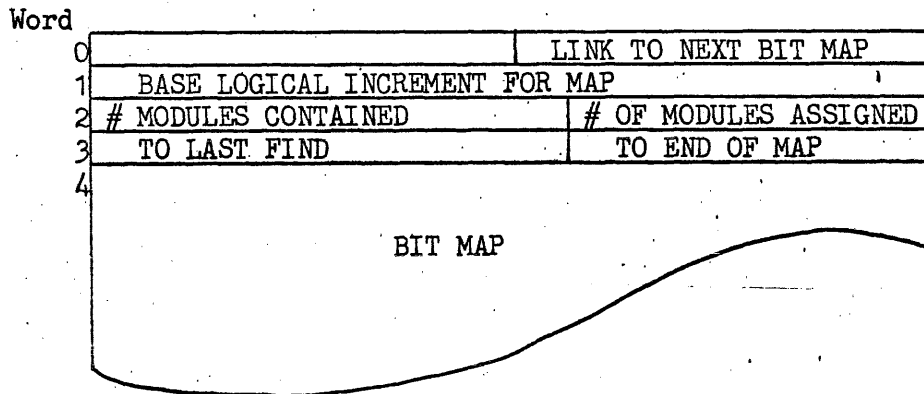
•Release core buffer to core allocator used to retain drum buffer and descriptor..

Exit: Return control to requestor with successful completion status A = 000000000.

### 7.4.2 Cooperative Library

Cooperative library is a random access file subdivided into modules and used as a pool for the cooperative mechanism and other systems elements requiring small pieces of mass storage to load programs and otherwise control the OMEGA environment.

An element within cooperative control maintains the library allocating and releasing modules upon request or need. Address and availability of each module is maintained via a bit map maintained in core. Format of bit map is as follows:



- Word 0 - Used to link bit maps reflecting additional allocations of mass storage to cooperative library. Zero indicates end of chain.
- Word 1 - Contains base logical drum address to which constructed increment from bit map is added to form complete address.
- Word 2 - Upper contains number of modules original mass storage allocation provides in map. Lower is used to maintain a count of modules currently being used. If upper and lower are equal upon release of a module a call is made to C.S.R. Function code 17 for possible release of a map.
- Word 3 - Upper contains an increment, relative to word  $\emptyset$ , of where last find for a module request. Lower contains an increment, relative to word  $\emptyset$  to last word of map.
- Word 4-N - Each bit map word contains 30 drum modules showing their availability, and address. Each module is represented by a binary bit if set to (1) module is available, (0) module is unavailable. The address for each module is formed by following formula:

$$\text{Module address} = [(30 \cdot W) + B] M + D$$

where: W = is word count within bit map of available module relative to word 4

B = bit position of available module within (W) 0 through 29

M = module size set at 198

D = the content of word 1

To determine the bit position within a map of released module the following procedure is used:

- Determine which bit map module belongs  $D_1$  released module  $D_2$

- $[(\text{released module} - D_1) / M] / 30$  quotient = W, remainder = B

### Cooperative Library Allocation

Allocation of random access storage and forming the corresponding bit maps is performed by the cooperative service routine (C.S.R) upon demand of I/O cooperative control or systems initialization. This is performed by executing a call to C.S.R. with function code 16 (see ). Upon return of control status, word will indicate successful allocation or not. I/O cooperative control will request additional allocations of mass storage each time total modules available drop below constant specified in bit map chain descriptor.

### Bit Map Chain Descriptor

Contains chain call to bit maps and other descriptive information with regard to module allocation. And is contained with the "System Table Links" (see ).

Word

0	MINIMUM DRUM MODS	CURRENT # DRUM MODS
1		LIBRARY BIT MAP CELL
2		PUSH/POP CELL (C.C.)
3	MAXIMUM # OF CORE BUF.	CURRENT # OF CORE BUF.
4	# OF BUFFER STEALS	# OF BIT MAPS FORMED
5	PRIMARY OUTPUT MAX.	SECONDARY OUTPUT MAX.
6		C.S.R PUSH/POP CELL

Word 0 - Upper contains minimum number of modules system should degenerate to before additional allocations of mass storage is made. If additional storage is not received upon request cooperative control will

allow library to drain and "PUSH" all unsatisfied requests until module is released. Lower contains a count of total available modules summarized from all bit maps.

Word 1 - Contains chain cell to cooperative library bit maps.

Word 2 - Contains PUSH/POP cell for request which could not be performed by cooperative control.

Word 3 - Upper contains count of maximum number of core buffers which should be assigned within system at any one time to contain primary or secondary data. Lower contains a count of core buffers currently allocated and used for primary or secondary data.

Word 4 - Is used for statistical data. Upper - number of times a core buffer area was stolen from one stream to satisfy another request. Lower contains a count of additional mass storage extensions which were effected for the cooperative library.

Word 5 - Contains maximum number of drum modules which should be contained on drum for any one stream before some disposing action is called for. When any one stream reaches the maximum "cooperative control" will call C.S.R. by function code 20 to cause one of the following to occur: activate output unit record routine, allocate additional mass storage, temporarily suspend requesting program.

Word 6 - Is used PUSH/POP cell for cooperative service routine.

#### Additional Uses of Cooperative Library

Currently OMEGA utilizes the cooperative library to maintain lists and tables used in selection, allocation and control of tasks. The following lists are currently contained and assure 198 word modules.

- Selection Job Stack (see 6.30.1)
- Job descriptor module (see 6.30.2)
- Pre-summary module (see 6.30.3)
- Selection facility map (see 6.30.4)

### 7.4.3 Functional Routines

DPOP - routine to 'POP' certain request that have been 'PUSHed' on a general 'PUSH' chain.

Caller: Cooperative Control

Parameters: B6 = address of current task addendum  
B2 = type code by which requests are to be 'POPed' as follows:

Ø1 = POP a request for an input image if the task is the correct one.

Ø2 = POP a request for an output image if the task is the correct one.

Ø3 = POP a request for entry to DGET.

Ø4 = POP a request for entry to DRGET.

Ø5 = POP a request for a core buffer.

Ø6 = POP a request for a drum module.

Function:

- Search the general "PUSH" chain by type code for a request which has been 'PUSHed' with that code in the A register.
- In the case of types Ø1 and Ø2 a match must also be made on the task that the request was made during.
- If a match was made, 'POP' the request that the match was made during. Exit in either case.

Exit: Return on completion.

DGET - services requests for 202D word core buffers.

Caller: Cooperative Control

Parameters: B6 = address of current task addendum

Function:

- If the maximum number of buffers has not been allocated, make a call to core allocation for a buffer.
- If the maximum has been allocated, or if the above request is not filled, follow the task addendum chains to locate one of the allocated buffers which is not in use.

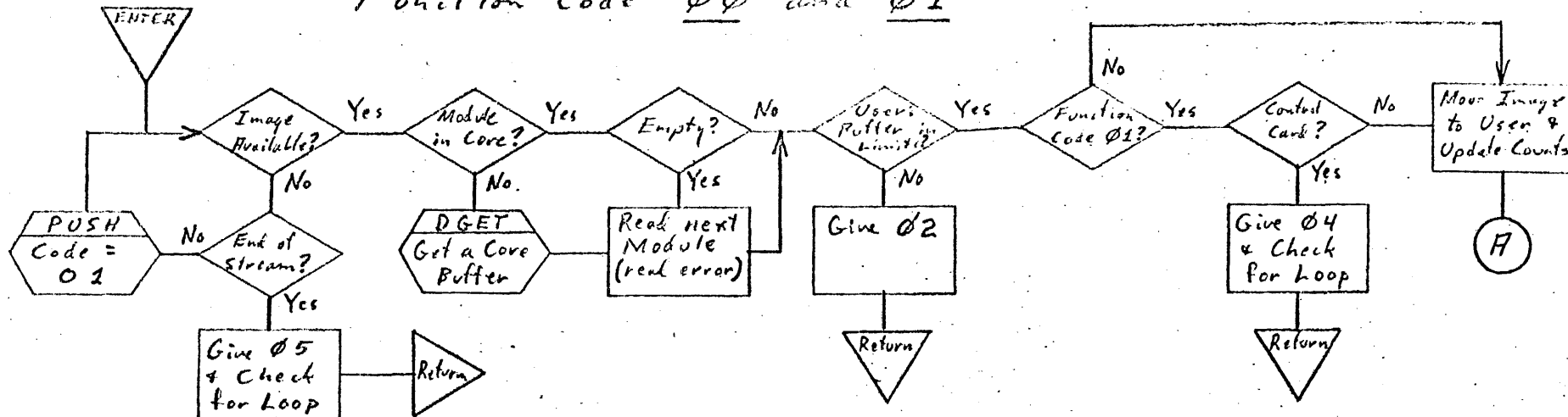


•If none such are found, 'PUSH' the request until a buffer becomes available.

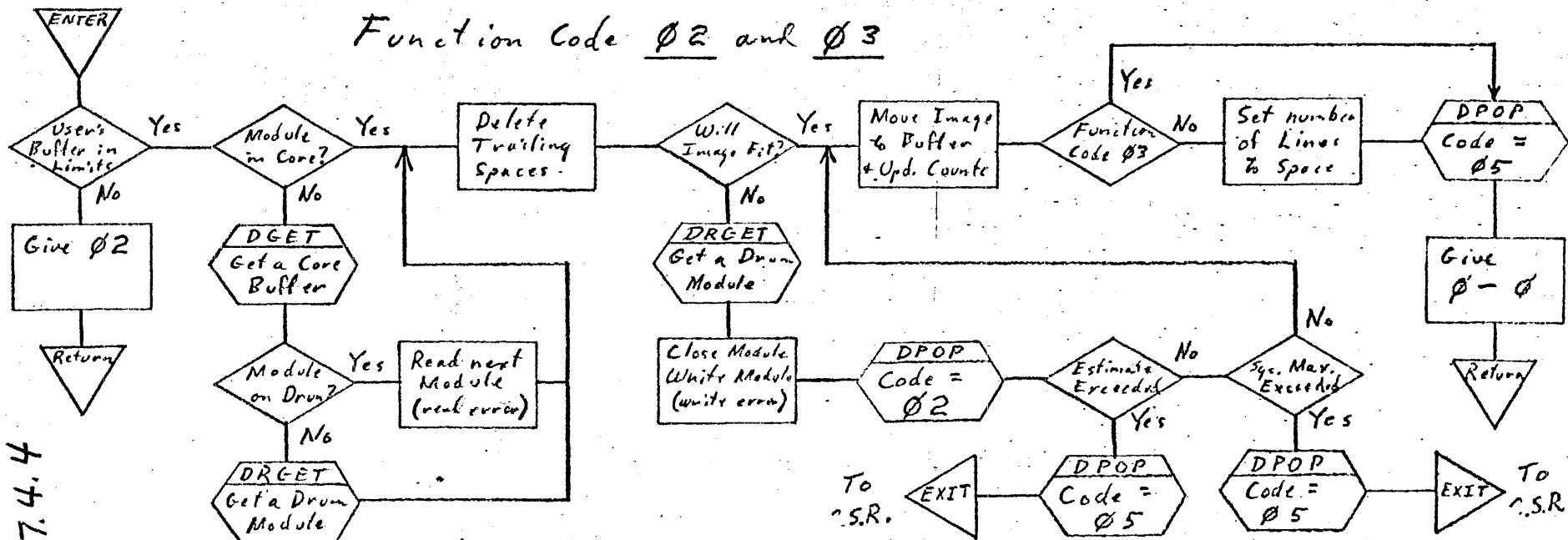
•When a buffer is found, empty it onto the drum (if necessary).

Exit: A register contains address of obtained buffer on completion.

### Function Code $\emptyset\emptyset$ and $\emptyset 1$

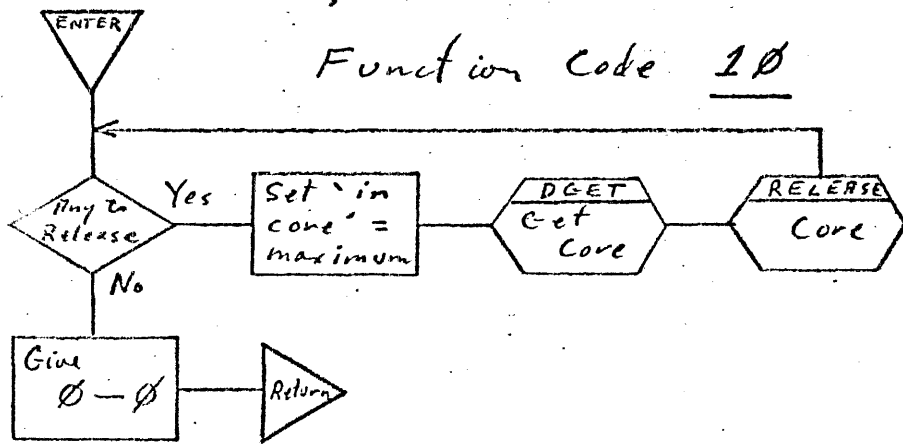
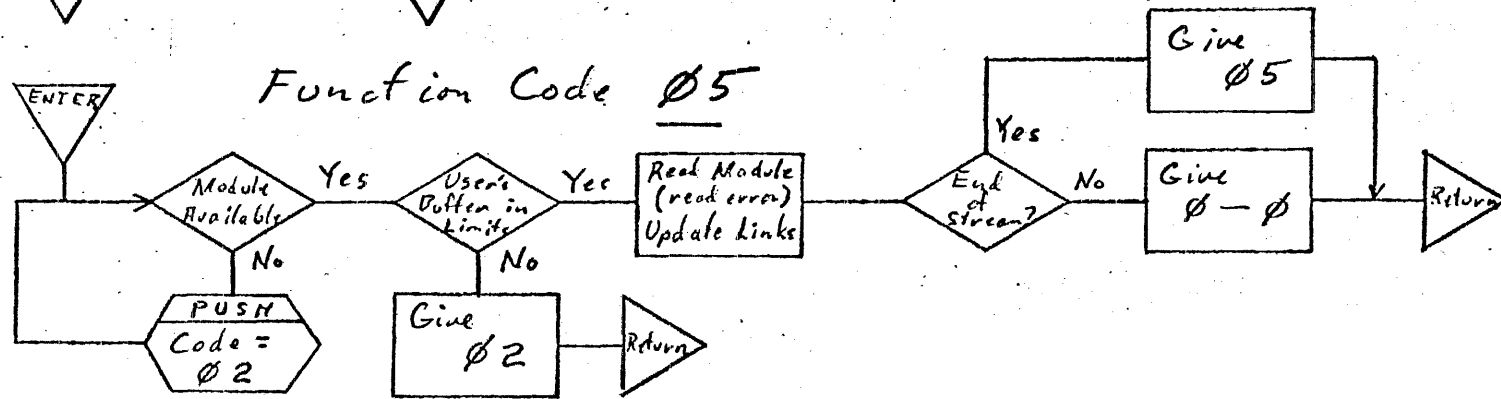
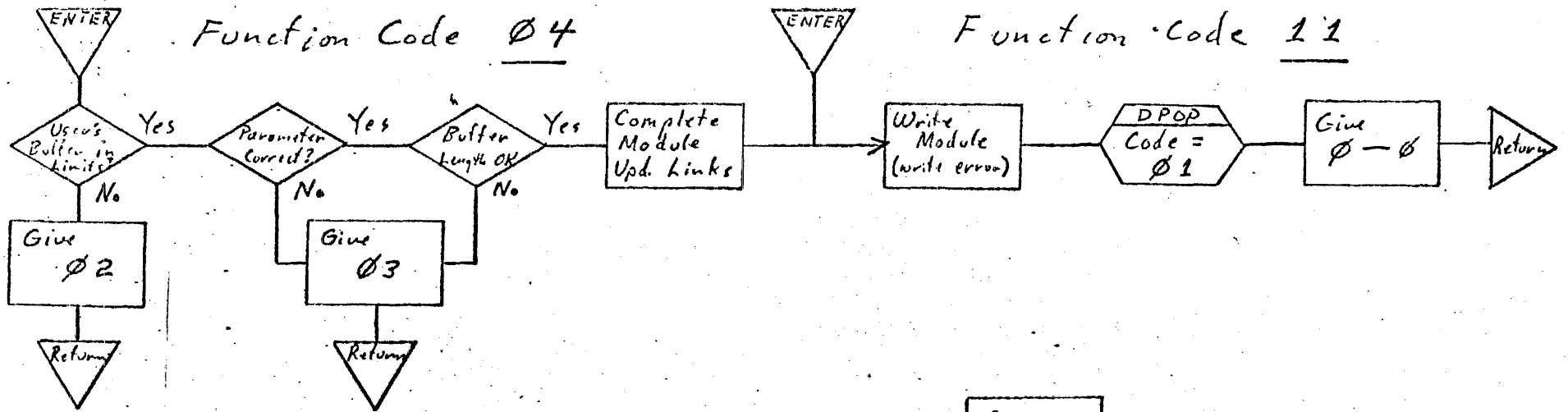


### Function Code $\emptyset 2$ and $\emptyset 3$

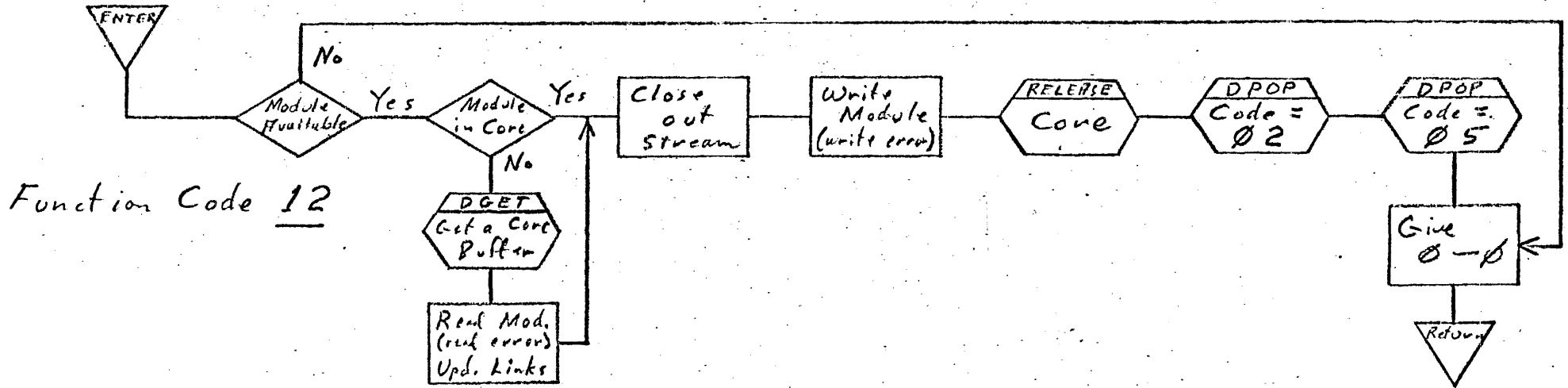
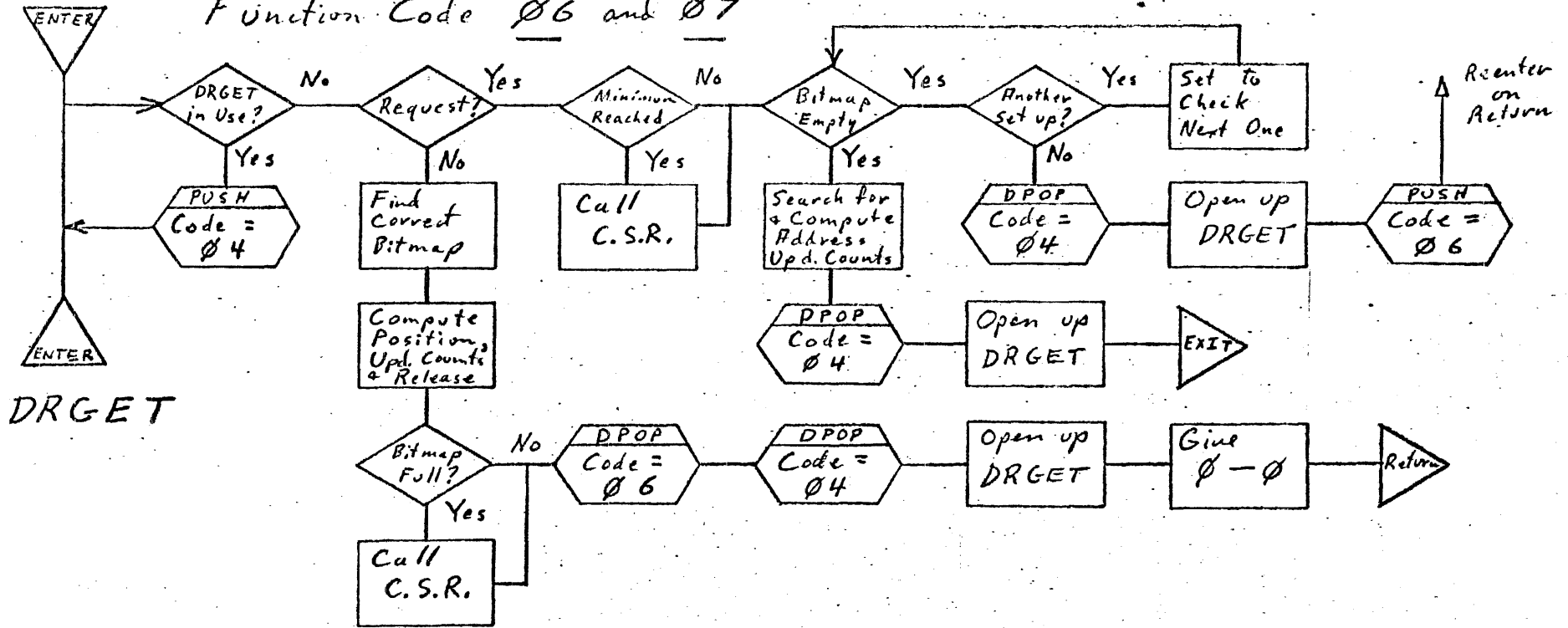


7.4.4

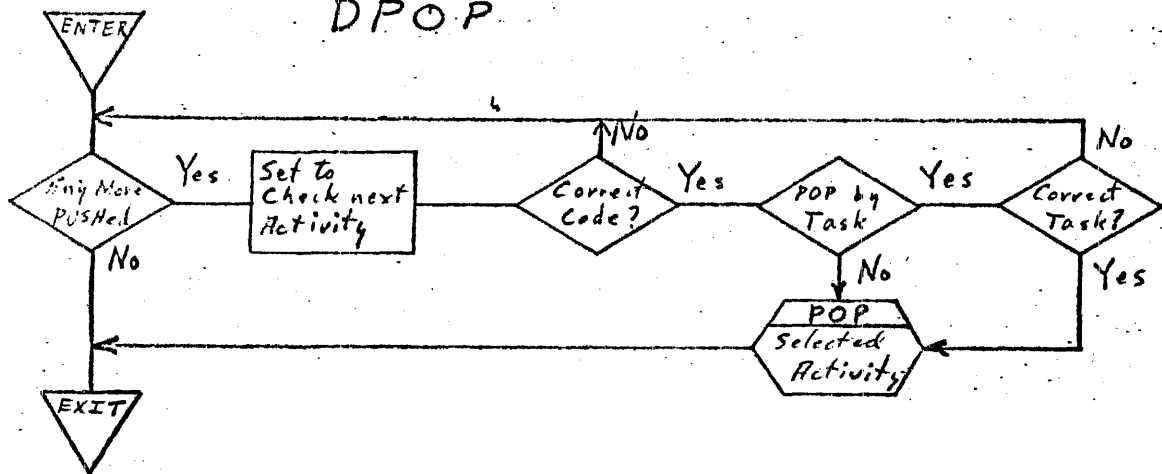
Reenter on Return



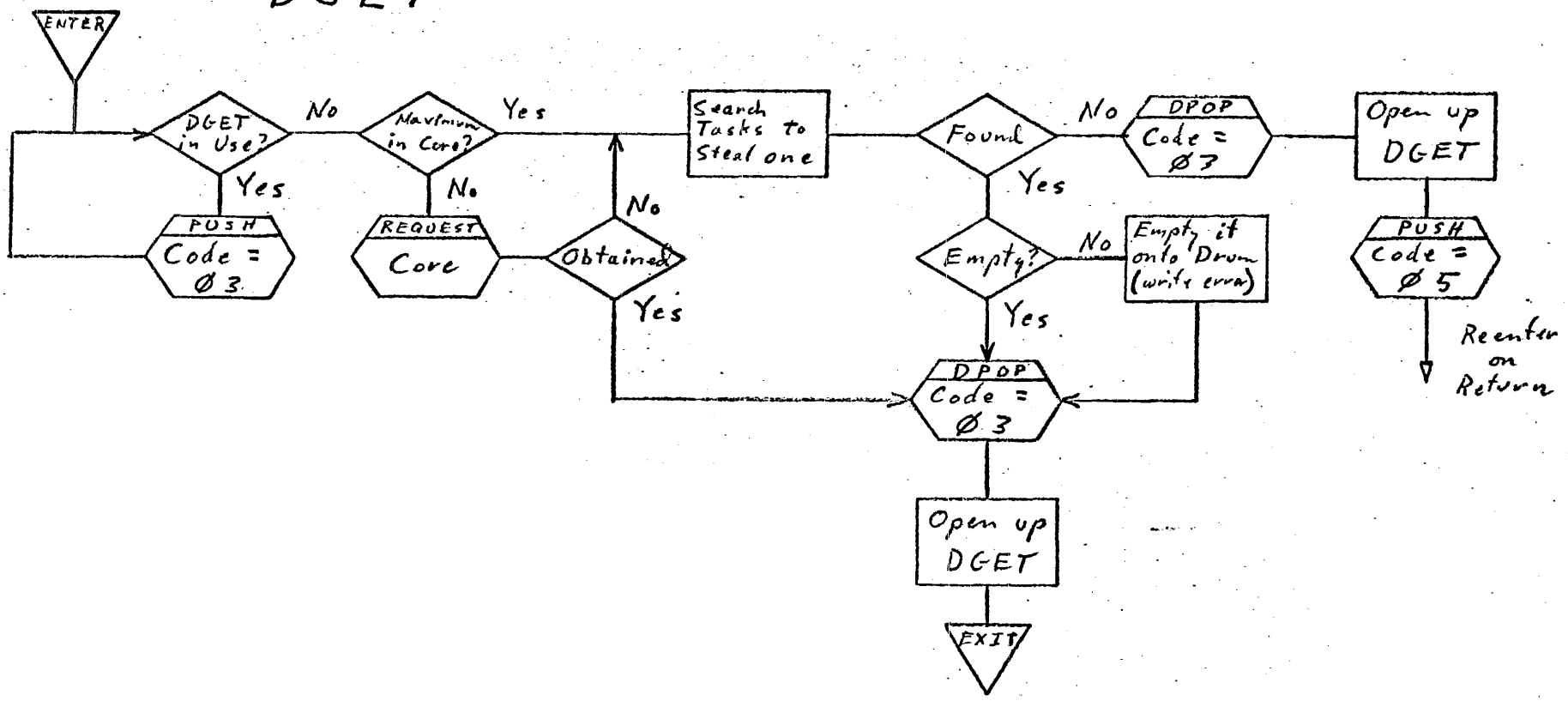
# Function Code 06 and 07



# DPOP



# DGET



## 7.5 Cooperative Maps and Tables

### 7.5.1 Drum Module

Drum modules are used by cooperative control and unit record routines to contain items transferred to or from random access storage.

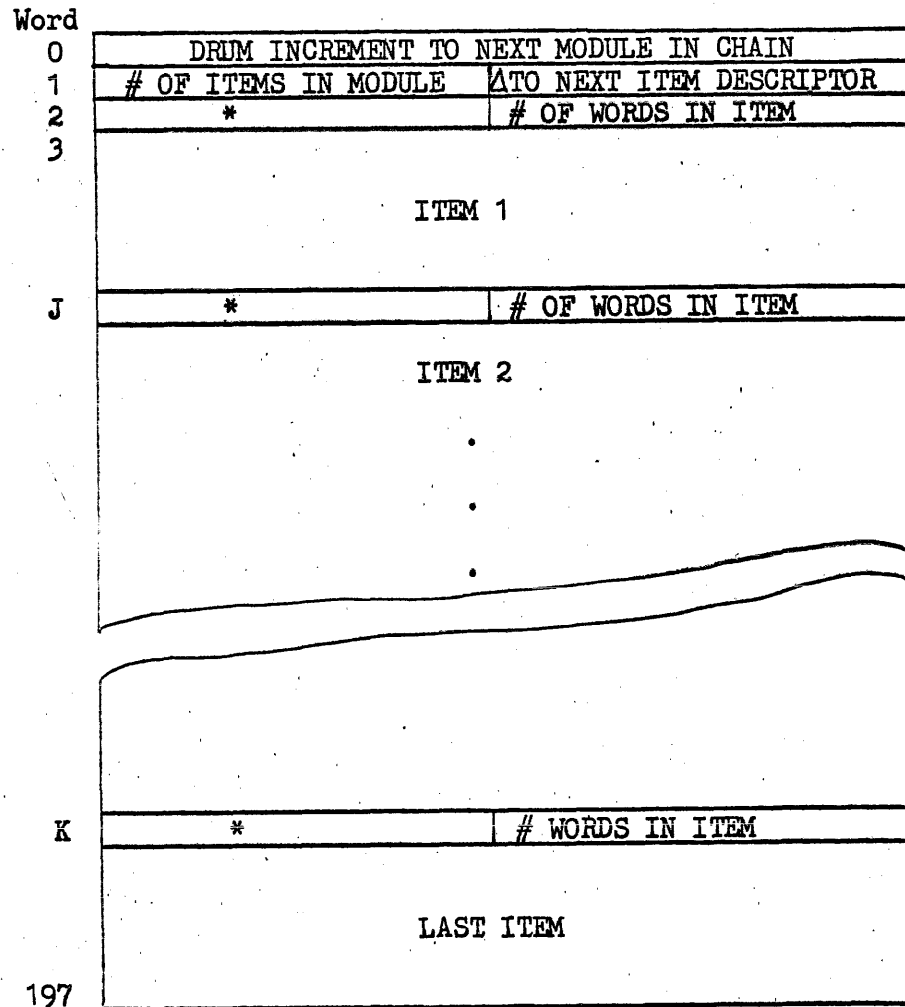


Fig. 7.5-1

Word 0 - contains drum increment of next module in the chain and is controlled by cooperative control.

Word 1 - upper contains # of items currently within the chain. Lower is an increment from word 0 of next item or free area within the module.

Word 2, J & K - are item descriptors required for each item contained within the module. Lower is number of words contained in the item excluding the descriptor. Upper contains number of lines to space before print if item is primary output or contains error status code from unit record routine or drum retrieval is primary input.

Word 3 through J-1, etc. contains primary input, primary output or secondary output images dependent upon which chain drum module is linked.

Module Size Drum module size has been fixed at  $198_{10}$  words as determined by the following constraints.

- The module must be a multiple of a FASTRAND sector to improve input/output efficiency if cooperative library is allocated to FASTRAND.
- Drum modules are of a fixed size due to pooling mechanism and associated program space but contain images from varying device and image lengths. To determine some criteria 16 word card images and 27 word print lines were chosen as normal. The module size should therefore be a multiple of both to reduce wastage of mass storage.
- A third consideration is the cooperative mechanism's routines delete from each item full words of trailing space codes to reduce mass storage requirements. To make this effort meaningfully the drum module size should as be as large as possible without requiring allocation of large core buffers which may be dormant.

The following table illustrates alternate choices of module size. Each module requires two words for module descriptor and is assumed to contain 17 word card images excluding item descriptor or 28 word print lines including descriptor. The table does not take into consideration exclusion of space codes from end of images.

# OF WORDS IN MODULE	# OF CARDS CONTAIN	CARD WASTAGE	# OF PRINT LINES	PRINT WASTAGE
33	1	16	1	3
66	3	13	2	8
99	5	12	3	13
132	7	11	4	18
165	9	10	5	23
*198	11	9	7	0
231	13	8	8	5
264	15	7	9	10
297	17	6	10	15
330	19	5	11	20
363	21	4	12	25
396	23	3	14	2
429	25	2	15	7
462	27	1	16	12
493	29	0	17	17

\* 198 chosen for system. If heavy utilization of cooperative mechanism is made 396 or larger would appear to be a good alternate choice.



### 7.5.2 Drum Module Descriptor

The Drum module descriptor is used as the I/O packet in transferring Drum modules to or from the mass storage. The descriptor is utilized by both unit record routines and I/O cooperative control, however all modification of drum increments is restricted to cooperative control. Descriptors format is as follows:

Word			
0	<table border="1"><tr><td>FILE CODE (ZD)</td><td># OF WORDS TRANSFER 198D</td></tr></table>	FILE CODE (ZD)	# OF WORDS TRANSFER 198D
FILE CODE (ZD)	# OF WORDS TRANSFER 198D		
1	BASE ADDRESS OF DRUM MODULE		
2	DRUM INCREMENT OF CURRENT DRUM MODULE		

Word 0 - Upper contains File Code ZD indicating the I/O transfer is to cooperative library. Lower contains number of continuous words contained or used in the drum module; limited by convention to  $198_{10}$  maximum.

Word 1 - Base core address into which drum module is to be transferred. Address is relative to FLR of requestor.

Word 2 - Contains logical mass storage address, relative to file code. ZD, of drum module being processed.

### 7.5.3 Chain Descriptor

Each stream of data (primary input, primary and secondary output) requires a unique descriptor to maintain its links and other controls. These controls are maintained in the task addendum of the Job and are of the following format:

Word	Field 1	Field 2	Category
0	PRIMARY OUTPUT MAXIMUM	SECONDARY OUTPUT MAXIMUM	PRIMARY FILE
1	DRUM LINK TO CURRENT HEAD OF MODULE CHAIN		
2	DRUM LINK TO CURRENT TAIL OF MODULE CHAIN		
3	I/O	ADDRESS OF CURRENT CORE BUGGER	PRIMARY FILE
4	TOTAL # OF MODULES	# OF MODULES IN SYSTEM	
5	DRUM LINK TO CURRENT HEAD OF MODULE CHAIN		
6	DRUM LINK TO CURRENT TAIL OF MODULE CHAIN		SECONDARY OUTPUT
7	I/O	ADDRESS OF CURRENT CORE BUFFER	
10	TOTAL # OF MODULES	# OF MODULES IN SYSTEM	
11	DRUM LINK TO CURRENT HEAD OF MODULE CHAIN		SECONDARY OUTPUT
12	DRUM LINK TO CURRENT TAIL OF MODULE CHAIN		
13	I/O	ADDRESS OF CURRENT CORE BUFFER	
14	TOTAL # OF MODULES	# OF MODULES IN SYSTEM	

Note - Drum Link set to 7-7 implies end of chain.  $2^{29}$  set to 1 implies partial module stored on drum.

Word 0 - Upper contains maximum number of pages converted to modules of primary output expected for JOB. Lower contains maximum number of cards converted to modules expected for the JOB.

Words 1, 5 & 11 contains logical drum address of head link in chain.

Words 2, 6 & 12 contain logical drum address of tail link in chain which is currently stored on the drum.

Word 3, 7 & 13 are used to contain core address of a buffer containing Drum Module and descriptor when: an image is being transferred to an operating program (Word 3) or a primary or secondary output images is being accepted by cooperative control from an operating program. When interrogating these core cells they have the following meanings:

$2^0-2^{18}$  non-zero implies address of buffer descriptor with drum module immediately following.  $2^0-2^{28}$  set to zero implies no buffer or descriptor currently allocated to stream.

$2^{29}$  set to (1) indicates buffer currently in use and not available for re-assignment, release or use until current process complete.

Words 4, 10 & 14 - Upper is used to maintain a count of all modules allocated for described stream used for accounting and error overflow. Lower contains a count of Drum modules currently contained on mass storage for the stream.

## 7.6 Output Unit Record Routines

Primary and secondary output unit record are normal worker program elements responsible for punch, print, transmit or otherwise process primary or secondary output streams. Each routine is programmed to handle a particular device or type of devices and is called for and activated by the "cooperative service routine" due to one of the following:

- . Particular output stream has reached its maximum length as determined by comparing the number of modules in the system contained in tasks chain descriptor against maximum allowed value contained in "Bit Map Chain Descriptor".
- . Job stream is terminating at which time "TERMINATION" will request activation of output unit record routines.

The sequence of functions performed by each routine is as follows:

- a) Request first module of assigned chain from cooperative control function code 05. First item of module contains an accounting or identification item to be outputted as first image.
- b) Request next module of assigned chain from cooperative control function code 05. Submit each item contained in drum module to assigned device. Buffer area for Drum module and descriptor are contained in the unit record routine. Repeat this step until end-of-stream status code status code is returned for a module request "A" register = 7777740005.
- c) Exit to "cooperative service routine" Function Code 07 primary output or Function Code 11 secondary output. Program control may be returned at the starting location to process a new stream.

Output unit record routines are designed programmed and run under the following constraints:

- . Assignment of output device used to output images is by normal "ASG" control card collected with the unit record routine object code element via the LOADER.
- . All U.R. routines are activated and terminated via "cooperative service routine" and may not FORK or be fragmented.
- . Each routine contains an area for drum module and descriptor used to accept drum modules from I/O cooperative control. All items contained in drum module have had full words of trailing space codes deleted.
- . Each routine is activated at its first location.

## 7.7 EDIT Routines

## 8.0 Real Time and Communication Control

### 8.1 General Description

The Omega communications software is very flexible allowing the user to optimize it to best serve his needs, yet it is highly efficient and capable of handling an unlimited number of devices of all the various types. One user might desire a high level interface between the software and user programs similar to File Control software. Thus allowing him to use simple commands such as "SEND" and "RECEIVE" on a certain file, the software would then handle the details of buffering, translating, packing, unpacking, polling, establishing remote connections, etc., as needed. While another user might prefer to maintain closer supervision over the communication functions. This user as an example can handle his own polling or buffering, etc. These user programs would then of necessity have to be more detailed and precise.

#### 8.1.1 Level 1 Control

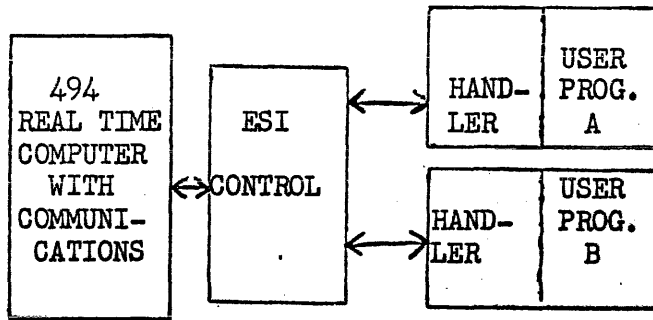
The level 1 OMEGA Communications Software provides the User with a low level, basic routine (ESI CONTROL) which controls the hardware. The User Program essentially has complete control of the hardware by commands which are given to ESI CONTROL. ESI CONTROL will perform the actual input-output hardware instructions as directed by the User Program.

Communications Facility assignment routines are provided in the Level 1 software. These routines allow units which are not in use, to be acquired and assigned to the requesting User Program. A Communications Facility Map will be maintained on Random Access Storage which lists the units, mnemonic names for them, and the units which are in use.

The User Program will perform all editing, translating, packing, unpacking and staging of messages. It will also have to form the poll output messages and monitor the input poll replies. Any special control requirements of the various devices will have to be recognized and provided for in the User Program.

"CTM Control Blocks" and "Unit Control Blocks" are tables in memory which will contain control type information used by both ESI CONTROL and the User Program. These tables are formed as the units are acquired by the Comm. Facility Assignment routines.

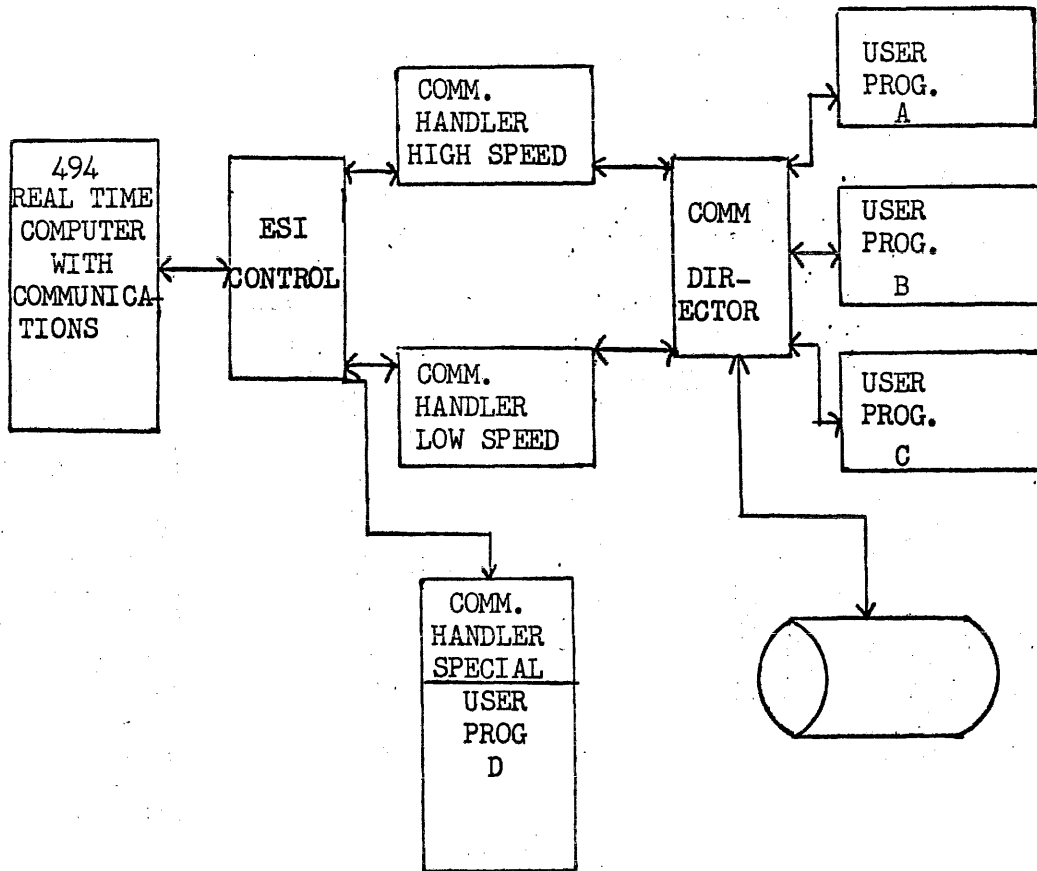
This Software should fill the need of Users with specialized communication systems, which do not require the flexibility of OMEGA Communications Interface Level 2. And some Real Time Programs presently in existence may be able to efficiently utilize the Interface Level 1 Software.



Level 1 Control  
Fig. 8-1

### 8.1.2 Level 2 Control

The lowest level of Software is the "ESI CONTROL" routine. ESI CONTROL executes the hardware functions, interprets the interrupts and provides other basic communication with the hardware. There is only one Communications Function Executor, but it may communicate with any number of "Communications Handlers". The "Communication Handler" is the next higher level routine. A Handler might be specialized for a type of remote device, a particular buffer scanning technique or unique to a particular applications program. The handler might connect directly into some user programs or the "Communications Director". The Communications Director provides a buffering or staging mechanism for the input and output messages, depending upon activity these may be queued on mass storage to preserve space in core memory. It is the Communications Director which interfaces into the high level user program providing him with input messages and accepting his output messages for subsequent transmission to remote communication devices. The Communications Director has other miscellaneous functions, one of which is identifying unsolicited messages.



Level 2 Control

Fig. 8-2

## 8.2 TABLES

In order to control the various communication devices certain information is required concerning the characteristics of the remote units and transmission lines. Current status and control information must also be available. For this reason the following tables have been established:

- . CTM Control Block
- . Unit Control Block
- . Communication Facility Map

### 8.2.1 CTM Control Block

The CTM Control Block is a 30<sub>8</sub> word (Level 2) packet containing information relative to a particular CTM. There is one such Control Block in core for each currently active CTM pair. These Control Blocks are formed and placed in core as a CTM pair becomes active (see Section 8.2.3). The Control Blocks are accessed from a table based on ESI address (see Section 2.1.2.2) The table is referenced by the ESI Interrupt Processor, the Communication Handler, and the Communication Director (Level 2). The CTM Control Block is shown below with a description following.

CTM Control Block

0	STATUS	# WORDS
1	JOB #	ACT #
2	ADDRESS	
3	UCB LINK	
4	CTM TYPE	INPUT BUFFER SIZE
5	ESI (OUTPUT)	MPLX & CTM I.D.
6	CODE TYPE	CHANNEL NO.
7	ACTIVE UCB	
10	INPUT CHAIN COUNT	OUTPUT CHAIN COUNT
11	COMMAND WORD	EXTERNAL FUNCTION
12	INPUT BCW (HEAD OF CHAIN)	
13	OUTPUT BCW (HEAD OF CHAIN)	
14	INTERRUPT CONTROL	
15	INPUT MONITOR CNT	OUTPUT MONITOR CNT
16	INPUT ESI BCW	
17	OUTPUT ESI BCW	
20	ESI EXTERNAL INTERRUPT WORD	
21	SPARE	
22	ESI (DIAL)	MPLX & CTM I.D.
23	POLL PERIOD	TIME OF NEXT POLL
24	HEAD OF DRUM CHAIN	
25	TAIL OF DRUM CHAIN	
26	CORE BUFFER	
27	PUSH/POP LINK	



- Word 0-upper - Status of CTM
1. DOWN-BIT 29 set
  2. INPUT ACTIVE-BIT 28 set
  3. OUTPUT ACTIVE-BIT 27 set
- Word 0-lower - Number of words in this list. The number of words may vary depending on whether Level 1 or Level 2 control is being used.
- Word 1 - The Job and Activity number of the Communication Handler controlling this CTM. The ESI Interrupt Processor references this location when executing a QREF to the handler.
- Word 2 - Address of the handler controlling this CTM. Control is transferred to this address when the QREF above is made.
- Word 3 - Chain cell to first Unit Control Block (see Section 8.2.2). All UCB's associated with this CTM are chained through this cell.
- Word 4-upper - CTM and transmission line characteristics are listed here. Possibilities include:
- . synchronous/asynchronous
  - . simplex/half duplex/full duplex
  - . dial/automatic dial/poll
  - . line speed
- Word 4-lower - Input buffer size. The ESI Interrupt Processor will obtain this size input buffer from the buffer chain when required (see Section 8.3).
- Word 5-upper - ESI for the output CTM. The input ESI will be one greater.
- Word 5-lower - The actual multiplex and CTM I.D. code (used to send external functions to the CTM).
- Word 6-upper - Data Code type (ASCII, X3-3, etc.), code level, etc.
- Word 6-lower - I/O channel number
- Word 7 - Address of UCB which currently has an input or output message in process.
- Word 10 - Input and output buffer chain counts in upper and lower respectively. Each time the ESI Interrupt Processor chains a buffer the appropriate half-word is incremented. May be used by the handler to locate currently active input and output buffers.

- Word 11-13 - ESI Function Executor control words. Described in Section 8.4.2.
- Word 14 - Interrupt Control Word. Specifies ESI Interrupt Processor action at interrupt time. See Section 8.4.1.
- Words 15-20 - Control words used by the ESI Interrupt Answering and Interrupt Processing Routines. See Section 2.1.2.2.
- Word 21 - Space location, may be used by handler for control.
- Word 22-upper - ESI address for the DIAL CTM. Will be  $\emptyset$  if there is no DIAL CTM associated.
- Word 22-lower - The actual Multiplex and DIAL CTM I.D. code used to send external functions. Zero if there is no DIAL CTM.
- Word 23 - Poll period and time of next poll of the remote units on this CTM. Will be zero if not a poll line.
- Words 24-27 - Control information for drum queue of output messages for this CTM (Level 2 only). See section 8.6

### 8.2.2 Unit Control Block

The Unit Control Block is a 14<sub>8</sub> word packet containing information relative to a particular remote device. There is one Unit Control Block in core for each remote unit currently assigned. These Control Blocks are created and placed in core as each unit is assigned. They are accessed through a chain cell originating in the CTM Control Block. They are referenced by the Communication Handlers and the Communication Director (Level 2). The Unit Control Block is shown below with a description following.

Unit Control Block  
(UCB)

0	STATUS	# OF WORDS
1		CTM/UCB LINK
2		CTM CONTROL BLOCK
3	OUTPUT BCW LINK	
4	UNIT I.D.	
5	UNIT I.D.	
6		TA/UCB LINK
7	# MSG'S	# WORDS (MSG)
10		HEAD OF DRUM CHAIN
11		TAIL OF DRUM CHAIN
12		CORE BUFFER
13		PUSH CELL

- Word 0-upper - Status used by the Handler to indicate such things as:
- DOWN REMOTE UNIT
  - REMOTE UNIT RESERVED
  - INPUT ACTIVE
  - OUTPUT ACTIVE
- lower - Number of words in this UNIT CONTROL BLOCK
- Word 1 - CTM/UCB Link. A chain cell linking all units on a CTM to each other and to the CTM. The chain originates in the CTM.
- Word 2 - Address of the CTM Control Block to which this UCB is linked.
- Word 3 - Output BCW Link. A BCW specifying a chain of output communication buffers to be sent to this unit. An output message may be linked through this location by the handler when the CTM is active with another message. This buffer would then be established by the handler when the current message has been completed.
- Words 4,5 - Unit I.D. Information necessary to identify the remote unit. May contain phone number for a dial unit, poll code, transmitter start code, etc.
- Word 6 - TA/UCB Link. A chain cell linking the UCB to the Task Addendum of the program to which it is assigned. All units assigned to a task are linked together through this cell.
- Words 7-13 - Control information for input drum queue of messages from this unit (Level 2 only). See Section 8.6.

### 8.2.3 Communications Facility Map.

The Communications Facility Map is a table, located on Random Access Storage, which has a listing of the units, the mnemonic names they may be called by, the handlers which should be used, Unit Control Block information, Communications terminal Module (CTM) information and a summary of the units which are in use. This map will be set up at Systems Generation and later referenced by the Communications Facility Assignment Software routines as units are acquired and released.

0	Δ TO UNIT DESC LIST			Δ TO HANDLER DESC LIST		
1	SIZE OF USAGE LIST			TO USAGE LIST		
2	COMM UNIT NAME 1					
3	A	B	C	<sup>26</sup> ▽ HANDLER DESC	<sup>15</sup> <sup>14</sup> ▽ UNIT DESC.	0
4	A	B	C	▽ HANDLER DESC	▽ UNIT DESC.	
	}					
	COMM UNIT NAME 2					
X	A	B	C	▽ HANDLER DESC	▽ UNIT DESC.	
X+1	A	B	C	HANDLER DESC	▽ UNIT DESC.	
X+2	A	B	C	HANDLER DESC	▽ UNIT DESC.	
	}					

Word 0 U - The increment from the base of the map to the first unit description.

L - The increment from the base of the map to the first handler description.

Word 1 U - The number of words reserved for the usage list. Each usage entry requires two words.

L - The increment from the base of the map to the first word of the first entry in the usage list.

Word 2 A Communication Unit name by which assignment will be requested. The "ACQUIRE" statements will refer to these names.

Word 3-4  $2^{29}$  (A) = 0 One or more alternate handler - unit descriptor words following for this unit name.

$2^{29}$  (A) = 1 Final handler- unit descriptor word for this unit name.

$2^{28}$  (B) Undefined

$2^{27}$  (C) Undefined

$2^{26}$ - $2^{15}$  The index to the handler description to be used on this assignment. This index is relative to the base of the handler description list.

$2^{14}$ - $2^0$  The index to the unit description to be used for this assignment.

Word X Another unit name which may be used for assignment.

Word X+1- References to handler and unit descriptors  
X+2 for this unit name. Similar to words 3 and 4 above.

Communications Handler Descriptors  
(Initialize and Locate)

The mnemonic names previously defined points to a "Handler Initialize" descriptor, which describes a routine which will be loaded, executed and discarded. This routine will perform initial setup functions as required by the "Handler". The Handler Initialize Descriptor points to a "Handler Locate Descriptor" which describes a handler to be used for the normal handling of communication messages. The handler locate descriptor will indicate if the handler is presently in core, and if so where it is located.

HANDLER INITIALIZE DESCRIPTOR

0	HANDLER INITIALIZE
1	NAME
2	VERSION
3	FILE INCREMENT TO HANDLER INITIALIZE
4	LENGTH (HAN. INIT.)   ▽ HANDLER LOCATE
5	UNDEFINED

- Word 0-2 Contains name/version of communications "Handler initialize" routine.
- Word 3 Contains the drum address of the routine. If 0—0s indicates no initialize necessary, proceed immediately to "Handler Locate" descriptor.
- Word 4 U - Number of words of core required for this Handler Initialize Routine  
  
L - The index to the "Handler Locate" description to be used on this assignment. This index is relative to the base of the handler description list.
- Word 5 Undefined.

HANDLER LOCATE DESCRIPTOR

0	HANDLER	
1	NAME	
2	VERSION	
3	FILE INCREMENT TO HANDLER	
4	HANDLER LENGTH	Ø OR HANDLER CORE ADDR.
5		# USERS

Word 0-2 Contains name/version of communications "handler" routine.

Word 3 Contains the drum address of the handler.

Word 4 U - Number of words of core required for this handler.

L - Address of Handler if in memory or Ø — Øs if not in memory.

Word 5 Number of Communication Units currently using this "Handler". When this count becomes Ø the Handler will be purged from core.

## Communication Unit Descriptors (UNIT and CTM)

The word or words immediately following the mnemonic name point to a "Unit Description" which is a small table containing unit status and other information unique to a unit. A pointer within the unit descriptor points to a "CTM Descriptor" which describes the CTM for this unit. The Unit and CTM Descriptors are interrogated to determine if the requested communication unit may be successfully ACQUIRED. If so, a Unit Control Block will be formed in memory, also a CTM Control Block, unless a previously acquired unit required the same CTM Control Block in which case it would already be in memory.

### UNIT DESCRIPTOR

0	1/0 CTM DESC	# WORDS ON DRUM
1	STATUS	# WORDS IN CORE
2	UNIT I.D. OR POLL CODE	
3		

Word 0  $2^{29}$  If "1" indicates unit is not available for use at this time, it is either down or already in use. If "0" unit is available.

$2^{28}-2^{15}$  The index to the CTM descriptor. This index is relative to the base of the unit description list.

$2^{14}-2^0$  The number of words this descriptor occupies on drum.

Word 1 U - Status information pertinent to this unit.

L - The number of words required in core to form a Unit Control Block from this unit description.

Word 2&3 A unit identification or poll code which may be used by the handler.



CTM DESCRIPTOR

0		# WORDS ON DRUM
1		Ø OR ADDRESS IN CORE
2	STATUS	# WORDS IN CORE
3	CTM TYPE	INPUT BUFFER SIZE
4	ESI (OUTPUT)	MPLX + CTM I.D.
5	CODE TYPE	CHANNEL NUMBER
6	ESI (DIAL)	MPLX + CTM I.D.
7	PERIOD BETWEEN POLLS	

Word 0      Number of words this descriptor occupies on drum.

Word 1      Memory address if a Control Block has been set up for this CTM, otherwise 0—0s.

Word 2-5    Specific CTM information required to form the CTM Control Block in core memory.

Word 6+7    Additional CTM Control Block information, not always required. Dial or pollable CTMs require this information.

### Communications Unit Usage List

A drum stored list is maintained which contains unit identifying information for the units which are currently being used. The Unit Control Block core memory address is paired up with the indexes of the Handler Initialize and Unit Descriptor which were used when this unit was acquired. Thus by searching the usage list and following the indexes, a unit and its associated software components may be thoroughly analyzed.

0	I/O		ADDR OF UCB
1	▽ HANDLER DESC.	▽ UNIT DESC.	
2			
3			
4			
5			
6			
7			

- Word 0       $2^{29} = 1$ . Indicates this word and the following word contains unit identifying information for a unit which is presently being used. If  $\emptyset$ , this two word slot is not being used at the present time.
  
- $2^{17-20}$  The core address of the Unit Control Block.
  
- Word 1      U - The index to the "Handler Initialize Descriptor".
  
- L - The index to the "Unit Descriptor".
  
- Word 2,3  
& on      Same as words 0 and 1 but for a different unit. Each unit in use will have a 2 word entry in this list.

## 8.3 Buffers

Two types of buffers are used in Omega communications: communication buffers and packing buffers. Communication buffers are used to receive and send data at the hardware level while packing buffers are used to exchange data between the Communication Handlers and the Director or the application program.

### 8.3.1 Communication Buffers

Communication buffers are used to transmit and receive data from the Communications Subsystem. The same buffer may be used for both input and output; input entering the upper half word and output exiting from the lower half word. These input and output buffers although occupying the same memory locations are distinct in that they have no connection with each other. The input and output buffers may also be of differing size. The common point of these distinct buffers is the end of the buffer as shown in Figure 8-3.

Each communications buffer contains three control words located at the end of the buffer. These control words allow input and output buffer chains to be developed separately. The "Input BCW Link" is the actual buffer control word of the next buffer of the input chain. The "Output BCW Link" is the buffer control word of the next buffer of the output chain. If either of these two words is zero it indicates that this buffer is the current end of the corresponding chain. The number of words in the third control word is the size of this core area. This value is used by the core chain control routine for allocating or releasing buffers.

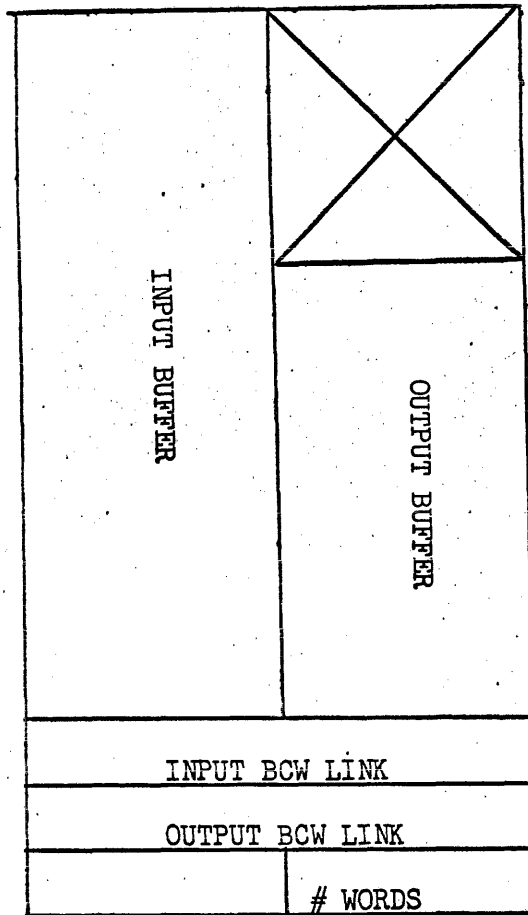


FIG. 8-3

COMMUNICATION BUFFER

These input and output links if present are used at interrupt time by the ESI Interrupt Answering Routine as the new input and output buffers for the terminating ESI as described in Section 2.1.2.2. When the Input BCW Link is zero, the ESI Interrupt Processor obtains a new input buffer and links it to the terminated buffer. An Output BCW Link of zero implies the end of an output message.

### 8.3.2 Communication Buffer Chain Control

An area of core is reserved for communication buffers by the Communication Initialization Routine. This reservation is made through a variable core chain declaration (Section 4.0). The size of the buffer area is established at system generation time.

Three chain numbers have been assigned for communication buffer control - Chain No.'s 3, 4, and 5. Chain No. 3 is the area described in the paragraph above. This chain is taken from free core and set aside for input and output communication buffers. Chain No.'s 4 and 5 are the input buffer and output buffer chains respectively. No chain declarations are made for these two chains - the areas placed in these chains are obtained by the chain control from Chain No. 3. Requests for buffers are made only from chains 4 or 5, as appropriate.

ENTRY - Entry to chains No. 4 and 5 may be made either  
 AND from the ESI Interrupt Processor or from the  
 EXIT Communication Handlers. Entry from the handler is via an Executive Return instruction with parameters specified in the registers:

#### Memory Request

ENT*B7	V <sub>0</sub>
ENT*Q	V <sub>1</sub>
EXRN	4 0 0 0 1

where: V<sub>0</sub> is the chain number referenced.  
 Chain 4 is the input buffer chain and  
 Chain 5 is the output buffer chain.

V<sub>1</sub> is the number of words requested.

Control is returned following the EXEC Return, with the address of the last word of the buffer in the A register.

#### Memory Release

ENT*B7	V <sub>0</sub>
ENT*A	V <sub>1</sub>
EXRN	4 0 0 0 2

where: V<sub>0</sub> is the Chain Number being referenced:  
 Input Chain No. 4; Output Chain No. 5.

V<sub>1</sub> is the address of the last word of the buffer (not including the three control words).

The chain control routine must be readily available to the ESI Interrupt Processor in order to establish input buffers. For this reason the processor enters the chain control with an Enter B and Jump instruction. The same parameters given above apply. Control is returned following the "EBJP" instruction.

**Operation** - A buffer request is made to chain No. 4 or 5 either by the Interrupt Processor or by the handler. If the request cannot be satisfied from the chain referenced, the chain control requests an area of core from chain No. 3 (if chain No. 3 cannot satisfy the request, chain 3 control requests the area from the Free Core Chain). The area obtained is placed in the chain which was not referenced. The length of the area is placed in the third control word.

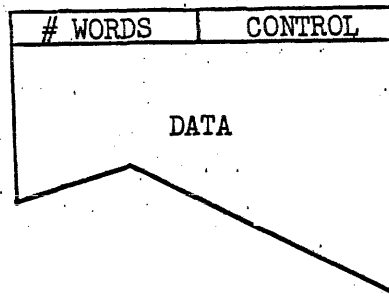
An example will clarify the situation. Initially Chains 4 and 5 are empty, and Chain 3 contains all the core allocated to communication buffers. Suppose the Interrupt Processor requests 60 words from the Chain 4. Chain 4 is empty so chain control requests 64 words (3 words are need for chain and link control and one more is added to make a multiple of 2) from Chain 3. The lower half of this 64 word area is still available to be used as an output buffer; so chain control places this area in Chain 5. When the buffer is released to Chain 4 a check is made to see if the corresponding buffer exists in Chain 5. If so, the area is removed from Chain 5 and the complete area released to Chain 3; if not, the input buffer being released is placed in Chain 4.

### 8.3.3 Packing Buffers

Packing buffers are used to exchange data between the Communication Handler and either the application program (Level 1) or the Communication Director (Level 2).

At Level 1 where the handler passes information directly to the program, any convention may be established concerning the packing buffers (they may be a part of the program). The program could be informed of a full packing buffer either by a QREF, activity registration, jump instruction, or any other means available. At Level 2, however, the staging mechanism of the Communication Director must operate

with a pre-defined format as shown below.



The first word of the packing buffer contains the number of words of data in the buffer and a control indicator such as SOM, EOM, etc. The remainder of the buffer contains data converted and packed to any degree desired by the handler.

On input the handler obtains data from the Communication buffer and places the data in the program's packing buffer (data format in the packing buffer is determined by the handler). On output the handler obtains data from the packing buffer and unstrings the data (if necessary) into a communication buffer for transmission. The acquisition and release of packing buffers is explained in Section 8.5.4.

Any or all of the previous output options may be selected. If both Option 1 and 2 are set, Option 1 will take priority. When the "Output BCW Link" of the terminating output buffer is not zero, the new buffer will always be established.

#### External Interrupt (2<sup>6</sup>-2<sup>8</sup>)

- Option 1 (2<sup>6</sup> set) - Execute a QREF to the handler when an external interrupt occurs.
- Option 2 (2<sup>7</sup> set) - Establish a new input buffer. (either from the "Input BCW Link" or from the input buffer chain)
- Option 3 (2<sup>8</sup> set) - Send a "Look for Sync" function when an external interrupt occurs.

Any or all of the above options may be selected.

#### 8.4.2 ESI Function Executor

In addition to the functions performed at interrupt time by the ESI Interrupt Processor, the handler at times may choose to send certain external function commands to various CTM's, and establish buffer control words in the ESI locations. These functions are handled by the ESI Function Executor. This routine is classified as an "immediate function" routine - it is entered through an Executive Return instruction, the functions are executed in the Executive Mode, and control is returned immediately upon completion. Entry is gained through the following packet:

ENT*Q	VO
ENT*A	0 0 0 1 5
EXRN	0 0 0 0 0

where  $V_0$  is the address of the CTM Control Block defining the functions to be performed.

The functions to be executed have been pre-stored in the CTM Control Block before the above packet is issued. Bit settings in the upper of word 11 of the CTM Control Block define the functions:

- 2<sup>29</sup> = 1      Send "Look for Sync" external function
- 2<sup>28</sup> = 1      Set input BCW. Word 12 of the CTM Control Block will be moved to the input ESI BCW position defined by word 5 of the Control Block.



## 8.4 ESI Control

The Communication Handlers define the method of control to be used for a particular CTM. Two routines are available to the handlers for this purpose: The ESI Interrupt Processor and the ESI Function Executor.

### 8.4.1 ESI Interrupt Processor

Through the ESI Interrupt Processor (See Section 2.1.2.2) the handler controls the sequence of events on the occurrence of an ESI interrupt. This control is accomplished by means of the "Interrupt Control" word (lower of word fourteen) of the CTM Control Block. Through this word a course of action is described for input monitor interrupts, output monitor interrupts, and external interrupts separately as follows:

#### Input Monitor ( $2^0$ - $2^2$ )

- Option 1 ( $2^0$  set) - Execute a QREF to the handler when and input monitor occurs (if  $2^0$  is not set, no QREF will be made).
- Option 2 ( $2^1$  set) - Send a "Look for Sync" function to the CTM.
- Option 3 ( $2^2$  set) - Do not chain a new input buffer when the "Input BCW Link" is zero.

Any or all of the above options may be selected. When an "Input BCW Link" exists in the terminating buffer the new BCW will be established regardless of any of the above bit settings. If the link is zero the Interrupt Processor will obtain and establish a new buffer unless Option 3 has been selected.

#### Output Monitor ( $2^3$ - $2^5$ )

- Option 1 ( $2^3$  set) - Execute a QREF to the handler when an output monitor occurs. If  $2^3$  is not set a QREF will be made only when the "Output BCW Link" of the terminating buffer is zero.
- Option 2 ( $2^4$  set) - Do not QREF the handler when the "Output BCW Link" is zero.
- Option 3 ( $2^5$  set) - Release the terminating buffer to the output buffer chain (Chain No. 5) and update the "head of output chain" (word 13) of the CTM Control Block.

- $2^{27} = 1$  Set output BCW. Word 13 of the CTM Control Block will be moved to the output ESI BCW position defined by word 5 of the Control Block.
- $2^{26} = 1$  Set output BCW for DIAL CTM. Word 13 of the CTM Control Block will be moved to the output ESI BCW position defined by word 22 of the Control Block.
- $2^{25} = 1$  Send external function. The function code contained in bits  $2^0$ - $2^2$  of this word (word 13) will be sent to the CTM defined by word 5.
- $2^{24} = 1$  Send external function to DIAL CTM. The function code contained in bits  $2^0$ - $2^2$  of word 13 will be sent to the CTM defined by word 22.

Any combination of the bits given above will result in the execution of the corresponding functions. A conflict may occur concerning the output and DIAL CTM's; therefore, when  $2^{27}$  is set,  $2^{26}$  will be ignored, and when  $2^{25}$  is set,  $2^{24}$  will be ignored. The bits are examined and the functions executed in the order given above. Any deviation from this established order will necessitate the issuing of two or more successive commands.

The bit settings describing the function to be performed are stored in the upper of word 11 of the CTM Control Block, the function code in the lower of word 11, if applicable, the input buffer in word 12, and the output buffer in word 13. The above packet is then issued. Words 12 and 13 then become the new heads of the input and output buffer chains respectively.

### 8.4.3 Channel Initialization and Termination

The Initialization and Termination of communication channels is accomplished by means of an "immediate function" command. Initialization is performed when the first communication facility is assigned, and termination is performed as the last communication facility is released. The following packet is used:

ENT*Q	VO
ENT*A	0 0 0 2 1
EXRN	0 0 0 0 0

where:  $V_0$  is a code indicating either initialization or termination.

When initialization is indicated the routine will clear all ESI locations and set input and output active on all ESI channels. Clearing the ESI locations will prevent data from entering core and cause a word of all bits (CTM stop code) to be sent to a CTM if it requests output data. Synchronous CTM's come on in the "Look for Sync" mode when power is applied so that the external function need not be issued during initialization.

When a termination operation is indicated, input and output will be terminated on all ESI channels.

## 8.5 Communication Handlers

The Communication Handlers interface between the Interrupt Processor and the application program. Data manipulation between the communication and packing buffers is accomplished by the handlers. The handlers may be written to handle a remote unit in any manner desired by the user. The handlers also may initiate output, initiate polling, specify input and output communication buffers, acknowledge the remote device, reset ESI buffer control words, search input buffers, etc.

Communication handlers are written for a particular type of remote unit; however, they are not restricted to only one CTM -- one handler may communicate with many CTM's. In this case, the handler must be coded such that it is non-re-entrant with respect to an individual CTM and re-entrant with respect to different CTM's.

### 8.5.1 Interrupt Processor Interface

#### Input -

Many options exist in the manner in which the handler communicates with the Interrupt Processor (IP). In fact, the methods which may be employed seems to be limited only by the imagination of the programmer. With respect to the communication buffers, the handlers may obtain buffers in the same manner as does the IP or the handler may let the IP obtain additional input buffers. Using the former method the handler may obtain two (or more) input buffers and link them together with the link of the last buffer pointing to the first buffer. Thus, when the last buffer is filled the Interrupt Answering Routine re-instates the first buffer effecting a re-circulation of buffers. With the above method the handler also has the option of being referenced when an input monitor interrupt occurs so that the buffer may be processed, or the handler may periodically scan the buffer chain for end of transmission codes. Another method the handler could use is to periodically scan and remove data from the communication buffer and reset the buffer control word. This procedure would minimize, if not prevent, input monitor interrupts. If a monitor did occur data would not be lost since the IP would obtain a new input buffer and chain it to the expired buffer. Input communication buffers which have been obtained by the IP should be released to the buffer chain by the handler as soon as the data has been removed. These input buffers may then be obtained and used again by the IP.

The handler instructs the IP on what procedure to follow when a monitor interrupt occurs by setting certain bits within the CTM Control Block of the interrupting ESI. These bits may be varied dynamically by the handler. The options currently available are described in Section 8.4.1.

When these bits indicate that the handler desires control after an interrupt has occurred, the IP executes a QREF to the handler with the address of the CTM Control Block in the A-register, and the terminating buffer in the Q-register. The handler to be referenced by the IP is identified by words 1 and 2 of the CTM Control Block. These words contain the JOB Number, Activity Number and address of the handler. The JOB Number will be that of the program which is to receive the data. The handler operates under the identity of the receiving program, but at a higher priority than the program proper.

The handler may instruct the Function Executor to set up certain input and output buffer control words (ESI's) and issue certain External-Function commands to the appropriate CTM. These functions are performed immediately by the Executor and control is then returned to the handler. Multiple functions concerning the same CLT may be performed with one command. A predetermined order of execution has been set up which should cover most contingencies. Deviation from the execution order will require two or more references to the Executor. See Section 8.4.2 for execution order and location of function and buffer words.

Time dependent control is available to the handlers as well as worker programs. These functions may be used for buffer scanning, acknowledgement remote units, etc. The time dependent functions are described in Section 5.0.

#### Output -

When the complete message has been placed in a communication buffer (or chain of buffers) the handler may check the CTM status in the CTM Control Block to see if output is currently active on the CTM. If not, the handler may initiate output to the unit by instructing the Executor to set up the initial buffer in the ESI address and to send an external function to the CTM. If output is currently active the handler may indicate in the UCB that a complete message is ready, or the handler may possibly chain the message to the currently active output buffer chain.

When an output buffer chain exists the monitor interrupts are handled completely by the Interrupt Answering Routine. If the "Output Buffer Link" in the terminated buffer is not zero it is the Buffer Control Word of the next buffer in the chain and is immediately established. The IP at this time may also return the terminated buffer to the Output Buffer Chain if instructed to do so by the handler. The buffer would not be returned to the chain if the message was being sent to two or more CTM's as in a message switching application. When an output monitor

occurs and the buffer link is zero, a new buffer is not established; the handler has either placed an "end of transmission" code in the buffer, or the handler will allow the I/O hardware to send the EOT when the CTM requests again (the I/O hardware will send a word of all bits and an acknowledge when it finds an ESI word count of zero).

Upon finding an "output buffer link" of zero the IP may QREF the handler to inform the handler that the complete message has been transmitted. At this time the handler may search the Unit Control Blocks associated with the CTM to detect and initiate any additional messages queued for output to that CTM. A reference to the handler when an output monitor occurs is optional; the option is exercised by setting the appropriate bits in the Interrupt Control Word of the CTM Control Block (see Section 8.4.1).

### 8.5.2 Data Handling

Communication Handlers move input data from the communication buffers to the packing buffers. During this operation any data manipulation required may be performed by the handler. For example, the handler may check character and message parity, the handler may convert the incoming message code to any internal code required by the program, and/or the handler may pack the data to any degree in the packing buffer.

To facilitate any or all of the above operations certain locations within the CTM Control Block may be used for control purposes. The upper half of word 4 will contain information identifying the level of the CTM pair and word 10 contains the input code type (ASCII, XS-3, etc.).

Handlers to be used for input to the System (such as the OMEGA Scheduler) will convert the data from the input code to Fielddata code and pack the characters five to a word in the packing buffers.

Upon receiving a packing buffer of output data, the handler requests a communication buffer from the Output Buffer Chain. The handler then unstrings and converts the data as required placing the data in the communication buffer. If the message in the packing buffer was incomplete the handler requests additional data in the same or another packing buffer. The buffer control word of the first buffer may be placed in word 3 of the appropriate UCB. When the additional data is received the handler obtains another communication buffer as above and chains the second buffer to the first by means of the "Output Buffer Link" as described in Section 8.3.1.

### 8.5.3 User Interface (Level 1)

Level 1 control implies a close association between the handler and the user program. Any method of program organization may be employed. The handler may be contained within the user program code or the handler may be contained in one of the libraries (job, group, system). An XREF may then be used to collect the handler with the user program. The handler may be established as a separate activity with higher priority than the user. Time dependent activities for buffer scanning may also be registered.

Packing buffers may be areas within the user program or they may be obtained from a core chain by the handler or the user. Packing buffers and their status may be exchanged between the handler and the user by any means available (i.e. QREF, activity registration, jump instruction, etc.).

The user program may consist of different levels of core. The handler exit, for example, may be to an edit and/or drum queuing routine for retrieval at a later time by a user message processing routine.

### 8.5.4 Communication Director Interface (Level 2)

When operating with level 2 control, the handlers interface with the Communication Director for queuing of messages on the drum (specified CTM's may still operate with level 1 Control while others are controlled at level 2).

In level 2 the handlers originate the commands given to the Communications Director. After performing the command the Director returns to the handler with a status indication. The Director operates as an extension of the handler although possibly, at a different priority.

The following commands are available to the handler:

- . Queue Input Message -- The handler has a packing buffer of date to be queued against a specific remote unit. Control is returned upon completion.  
Parameters required -
  - . Unit Control Block address
  - . Packing buffer address
- . Queue Input Message and release Packing Buffer - The Director will queue the message as above and release the packing buffer.  
Parameters required -
  - . Unit Control Block address
  - . Packing buffer address
  - . Number of words to release
  - . Core chain No.

- Request Output Message -- The handler requests an output message for a specific CTM. A status will be returned indicating whether or not a message is available. If available the appropriate UCB will be indicated.

Parameters required -

.CTM Control Block address

.Packing buffer address and number of words

- Obtain Output Message -- The handler requests an output message for a specific CTM. Control will be returned only when an output message is available.

Parameters required -

.CTM Control Block address

.Packing buffer address and number of words

- Request Output Message and Packing Buffers -- Same as Request Output Message except that the Director obtains a packing buffer from a core chain. Director returns buffer address and number of words.

.CTM Control Block Address

.Core Chain Number

- Obtain Output Message and Packing Buffer - Same as Obtain Output Message except that the Director obtains the packing buffer. Director returns buffer address and number of words.

Parameters required -

.CTM Control Block address

.Core Chain Number

The handler executes the commands by loading the registers with the appropriate parameters and giving an EXECUTIVE RETURN instruction. Control is returned following the EXEC RETURN with the status and other information contained in the registers.



## 8.6 Communication Director

### General Description

The Communications Director is a group of routines which make it possible for the user to write his programs to a higher level interface, freeing him from the communication device dependent details. The user will simply "ACQUIRE" units by a mnemonic name, then issue "SEND" and "RECEIVE" commands. A high degree of flexibility is still provided the user such as "ACQUIRE"ing a group of units, being able to "SEND" to any one or all units of a group and issuing "RECEIVE" commands for a message from a specific unit or from any unit in a group. There are other commands: "DIAL", "HANG UP", "POLL", "RELEASE" and "TRANSFER" which allow optimization and sophistication while still at the high level.

The Communications Director is composed of numerous elements, divided so as to perform specific functions. Only the high usage elements will be in core all of the time, others such as "DIAL" will be in core only while it is being used.

Another major function of the Communications Director is the staging of Input and Output messages on Random Access Storage. Thus allowing the timing of the communication devices and the User Programs to be completely independent.

### Detailed Description

ACQUIRE may be performed by either a control card or by a statement within the User Program. The ACQUIRE causes Communication Facility assignment to secure, if available, the unit or units implied by the NAME or GROUP and assign them to specified File Code. A "DIAL" operation may or may not be implied at this time, thus perhaps insuring the program not only of on site hardware facilities, but also that a connection has been made with the remote device. If "DIAL"ed when "ACQUIRE"d or if this unit was connected by a direct wire, "SEND" and "RECEIVE" commands could be issued immediately.

# HANDLER

# COMMUNICATIONS DIRECTOR

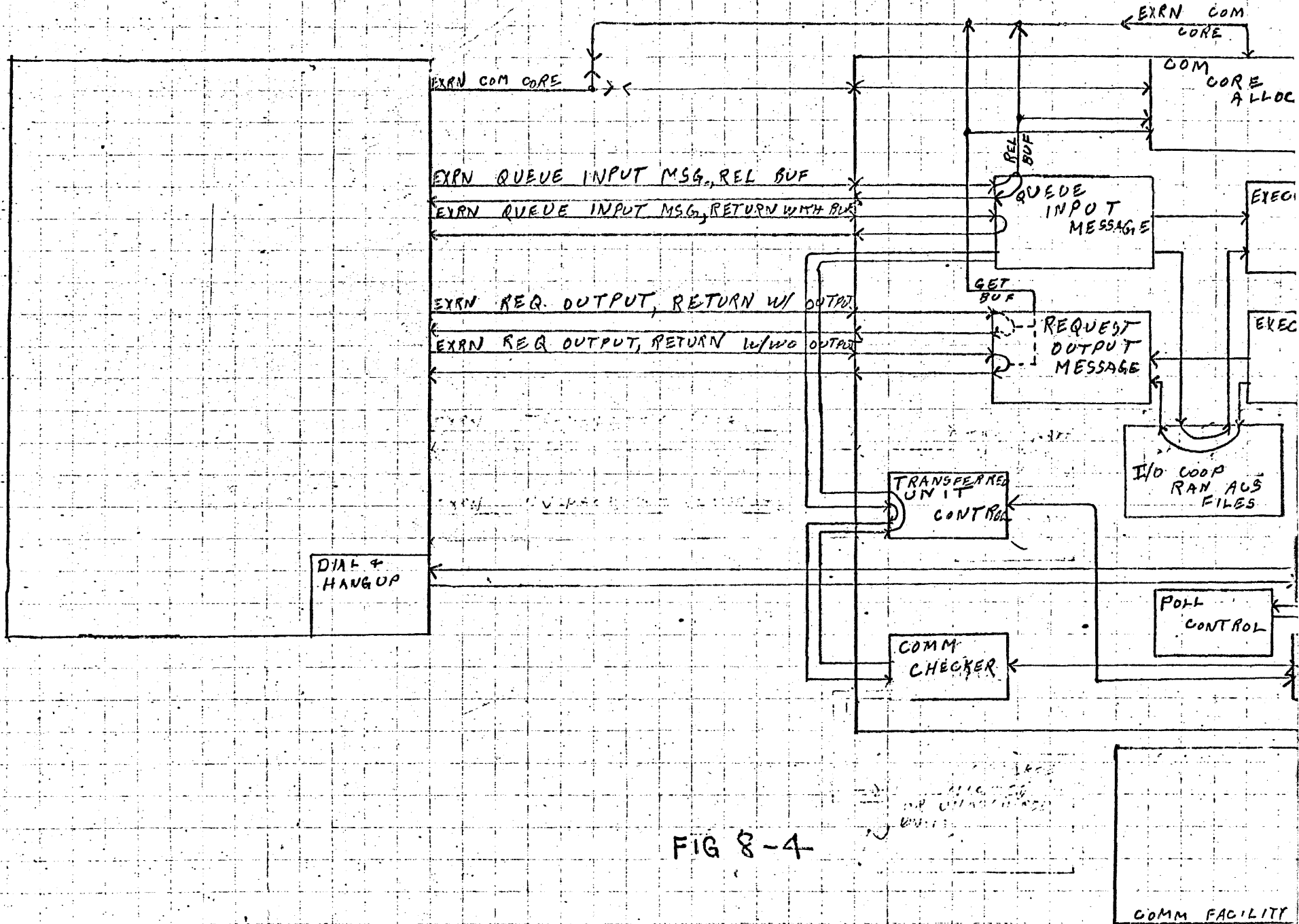


FIG 8-4

COMM FACILITY

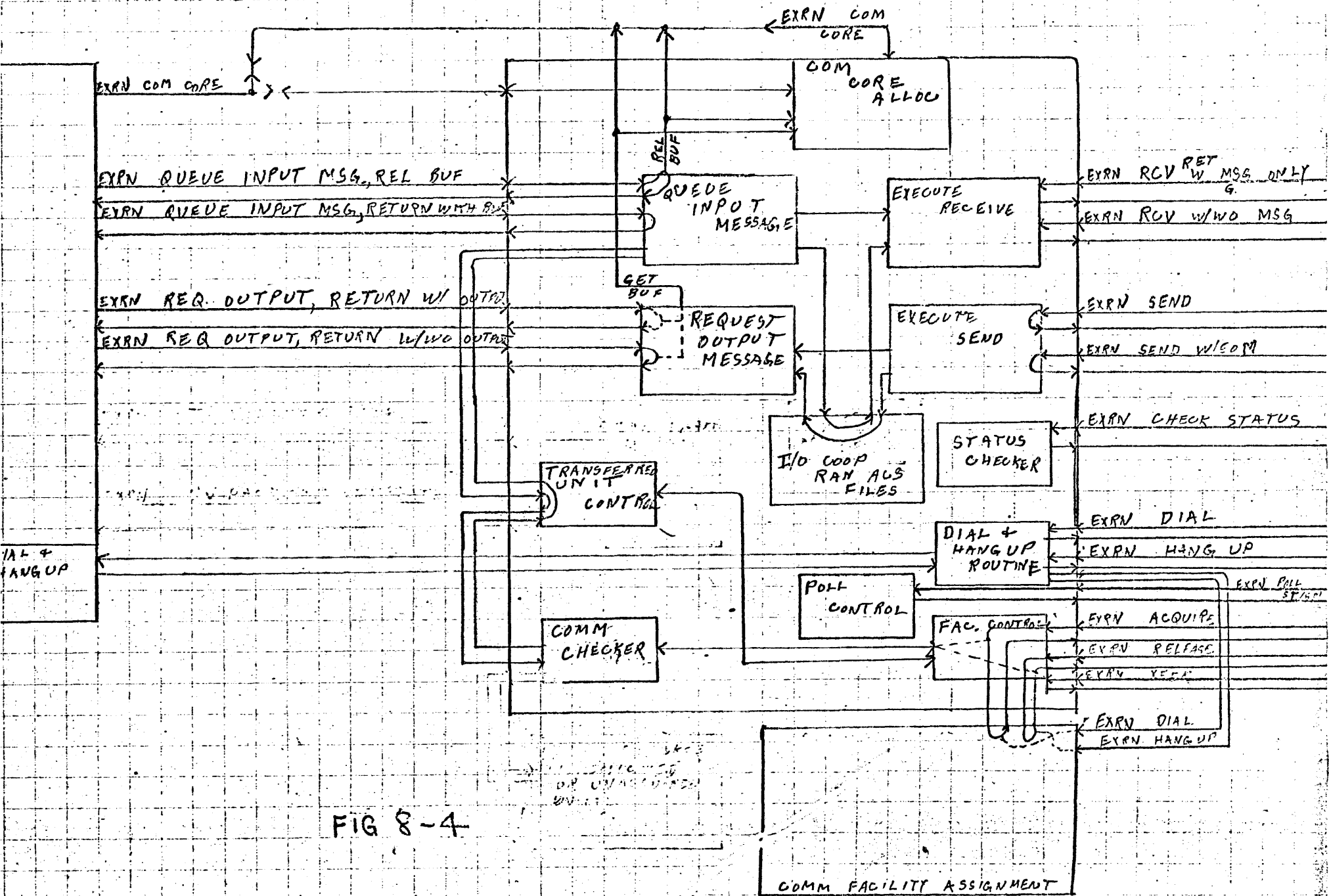


FIG 8-4

RECEIVE is a User Program command which indicates it is ready for an input communications message. The Receive command will be associated with one of the users File Codes. Previously at ACQUIRE time a unit or units were linked to this File Code, so now the "EXECUTE RECEIVE" routine of the Communication Director is able to locate the unit from which input data is requested. The first message or portion of a message received from this unit, which has not been given to the User Program, will be located and transferred to the User Program's buffer, which was furnished with the RECEIVE command.

A GENERAL "RECEIVE" command from the User Program means an input message will be accepted from any one of the units linked to the specified file code. The "Execute Receive" will scan down through the list of units looking for a message. When a message is found, it is transferred to the users buffer. A minor File Code will be given to the user, thus allowing him to reference the same unit again.

"Queue Input Message" is the Communication Director routine which receives the input messages from the Communication Handler. It will form strings of input messages, one string for each communications input unit. These strings will be formed on Random Access Storage. An input message may go directly from the "Queue Input Message" to the "Execute Receive" and to the user if there is an unsatisfied "RECEIVE" command.

SEND is a User Program command which implies output data is to be sent to a remote unit or units associated with a specific File Code of this program. The User Program assembles a complete or partial output message in its buffer, then it executes a "SEND" command, specifying the number of words, buffer base, a File Code and whether this is the end of a message or not. The "Execute Send" portion of the Communication Director will store the message on Random Access Storage forming an output string for each Communication Output CTM. Unit identification and end of message indication will also be stored with the message. The same message and end of message indication with different unit identification will be Random Access stored for each unit that is to receive this output "SEND" message, to satisfy a "SEND" to multiple units. If a Communication Handler had previously requested an output message for a CTM, which was not satisfied, the next "SEND" message for this CTM will be given immediately to the "REQUEST OUTPUT MESSAGE" routine of the Communication Director, thus the drum store need not always take place.

The user with numerous input units on one File code (major) might like to know how many input messages are waiting to be processed. A "CKISTAT" command executed by the User Program will cause a Communication Director routine, "Status Checker" to be loaded into core and executed. This routine will count the number of units assigned to this user File Code and store the count in a buffer like area furnished by the User Program. The number of messages and the number of words of the next message for each unit that has not been processed will also be stored in this buffer area. Thus, the User Program can determine the unit with the largest backlog of input and might like to alter his processing accordingly.

A "Dial and Hang Up Routine" allows a User Program to "ACQUIRE" a unit which is essentially an Input/Output with Dial CLT combination. Then with "DIAL" and "Hang Up" commands a number of remote connections may be made, one at a time. Perhaps a Poll by "DIAL" arrangement. This also allows the user program to "HAND UP" after all of the input data has been received, then when the output is ready to be sent, "DIAL" the number again.

A "Poll Control" routine allows a User Program to give "POLL" STOP or START commands for a unit or all units on a File Code. These commands given to non-pollable units will not affect them. The polling is controlled by the Communications Handler, it will poll or not poll a particular unit as per the last "POLL" command received. The unit description in the Communications Facility Map has provisions for declaring the unit as pollable or non-pollable and whether polling should be initiated when the unit is "ACQUIRE"d.

When a User Program determines there is no more activity for a unit, a "RELEASE" command may be executed. This will cause a "HANG UP" operation if it was a dialable unit and the unit will become available for other "ACQUIRE"s. The Unit Control Block will be cleared from memory. If no other units are using the CTM CB, it too will be released. If this happened to be the last communication unit operating the Communication Director will also be released.

A User Program may also "TRANSFER" a unit to the Communications Director. This indicates the User Program no longer has any need for this unit, but there is the possibility of unsolicited input on this unit. Thus the Communication Director has complete control of the unit as if it had "ACQUIRE"d it.

Unsolicited and unidentified input messages may occur only for those units which are assigned to the Communications Director. The Transfer Unit Control section of the Communication Director will be a small unit control section which will be in core as long as some units remain assigned to the Communication Director. When input is received from one of these units, it will not be known at this time which User Program it is for or it might be an input job stream. The Communication Checker will be loaded and will determine if the input message is the start of a job stream. If so the appropriate Unit Record Routine will be loaded and will ACQUIRE the unit from the Communications Director and proceed in a normal manner. If the message was not a job stream, further attempts will be made to identify it and then inform the proper User Program. If a User Program cannot be found for the message, it will be discarded and perhaps logged as an error situation.

## 8.7 User Program Interface Level 2

This describes the commands and interface which the User Program will use when communicating with the Communications Director. The following are the commands which may be used.

<u>Commands</u>	<u>Explanation</u>
#ACQ	External "ACQUIRE" request for a communications unit or group of units received via a control card.
ACQ\$	Internal "ACQUIRE" request for a communications unit or group of units.
DIAL\$	Direct that a "DIAL" operation be performed.
HANGUP\$	A line disconnect operation will be performed.
RELEASE\$	A previously acquired unit or units will be released.
SEND\$	A "SEND" operation passes an output message from the User Program to the Communications Director.
RECEIVE	A User Program request to the Communications Director for an input message.
CKISTAT\$	Provides the User Program with the status of the input messages received by the Comm. Director which have not been passed onto the User Program.
POLL	Allows a User Program to set a unit or units, so they will be or will not be polled.
TRANSFER	This transfers control of a unit or units from the User Program to the Communications Director or from the Communications Director to the User Program.

### Deatiled Description of the Commands

#### ACQUIRE

An ACQUIRE command may be executed either by a control card or by an internal program statement. The ACQUIRE will be performed in the same manner regardless of how it is initiated.

#ACQ Δ Options Δ NAME, File Code, XX—XX

ACQ\$ Δ Options Δ NAME, File Code, XX—XX

**NAME** If "G" is not specified as an option, this is a 5 character mnemonic NAME of the desired communication unit. The Communications Facility Map will be searched for a match to this mnemonic name.

If "G" is specified as an option, this is then the "NAME" of an element which contains one or more NAME's of comm. units which are listed in the Comm. Facility Map. The search for this GROUP NAME element will be first to the JOB library then the GROUP library then to the SYSTEMS library.

**File Code** Is 1 or 2 alpha characters which represent the File Code this unit is to be connected with. If Comm. units have previously been assigned to this File Code, the newly acquired unit will be added at the end of the chain. The minor file code number will be given to the user. The minor file code is a binary number indicating the relative chain position of a particular unit.

**XX—XX** This is an external number to be used by the X (DIAL EXTERNAL number) option.

#### OPTIONS

#### MEANING

- |   |  |
|---|--|
| G | GROUP. Indicates the NAME parameter is the name of an element. The named element then contains one or more comm. unit mnemonic names.  |
| A | ABSOLUTE. Ignored unless G option is also present. In which case, the "A" indicates all units named in the Group element must be acquired for this ACQUIRE to be successful.   |
| D | DIAL. The number contained in the Unit Descriptor will be dialed. This ACQUIRE will be successful only if the number can be successfully dialed. The dialing will be executed automatically if the automatic dial CTM is available otherwise the operator will be instructed to perform the dialing. |
| X | EXTERNAL. Dial number specified. Dial operation to be executed similar to the "D" option, except the external number will be used.   |



### DIAL

DIAL\$△Field Code, # of words, buffer base.

A DIAL operation will be executed using the number that is contained in the buffer. The dialing will be executed automatically if the automatic dial CTM is available, otherwise the operator will be instructed to perform the dialing. An unsuccessful status will be returned if the dial connection cannot be successfully completed.

### HANGUP

HANGUP\$△File Code

A hang up or disconnect operation will be performed. If the connection was performed by an operator, he will now be instructed to hang up.

### RELEASE

RELEASE\$△File Code/File Code(minor),

A previously acquired unit or units will be released, will become available for use by other programs. All units acquired for this File Code will be released unless a minor File Code is also specified.

The minor file code is a number indicating the relative acquired position of a particular unit. The first unit acquired would have a minor file code of 1, the second unit's minor file code would be 2, etc.

### SEND

SEND\$△File Code/File Code (minor), #of words, buffer base.

The "SEND" command directs the sending of the output message contained in the buffer, to be sent to the unit or units specified by the File Codes. The presence of a minor file code indicates the message is to go to a specific unit. The message will be sent to all units associated with this file code if a minor file was not specified.

### RECEIVE

RCV\$△File Code/File Code(minor), # of words, buffer base,

The RECEIVE command secures a complete message from the unit or one of the units as specified by the file code and places it in the specified User Program buffer. The absence of a

minor file code indicates the first full message located, which had been received from one of the units associated with the major file code will be acceptable.

An unsuccessful status will be returned if a full message isn't available from the designed unit or units.

PRCV\$ $\Delta$ File Code/File Code(minor), # of words, buffer base,

The PARCIAL RECEIVE command is the same as the RCV command except it does not require a complete message to be available prior to passing some of it to the User Program.

An unsuccessful status will be returned only when there is no input data from the applicable unit or units.

RCVW\$ $\Delta$ File Code/File Code(minor), # of words, buffer base,

The RECEIVE WAIT command is the same as the RCV\$ command with one exception. That is, if a full message is not available, control will not be returned to the User Program with an unsuccessful status. But instead control will be held in the Comm. Director waiting for a complete input message, when a message becomes complete, it will be placed in the buffer and control will then be returned to the User Program.

PRCVW\$ $\Delta$ File Code/File Code(minor), # of words, buffer base,

The PARTIAL RECEIVE WAIT command is similar to the PRCV\$ command except it does not return an unsuccessful status when there is no input data. Instead control will be held by the Comm Director until some input data is available.

### CKISTAT

CKISTAT\$ $\Delta$ File Code, # of words, buffer base

A command which allows a User Program to obtain the status of the input messages which have been received by the Comm. Director but not yet requested by the User Program. Thus the User Program might alter its processing technique depending upon the backlog of input messages.

This command will store the major and minor file codes of the last unit acquired for this file code (major), in the first word of the specified buffer. The number of messages and the number of words of the last partial message by unit, which have been received will be stored in the buffer, second word on up. A word will be stored. For each unit, they will be in numerical sequence matching the minor file code.

## POLL

POLLY\$ $\Delta$ File Code/File Code(minor),

This command will set the unit or units specified so they may be polled during the normal Communication Handler poll sequence. If a minor file code is specified, it indicates that only that unit should be set to the "poll yes" condition. If a minor file code is not listed, all units associated with the major file code will be set to the poll condition.

POLIN\$ $\Delta$ File Code/File Code(minor)

This command will set the specified unit or units so they will not be polled by the Communication Handler. If a minor file code is specified, it indicates that only that unit should be set to the "poll no" condition. If a minor file code is not listed, all units associated with the major file code will be set to the non-poll condition.

## TRANFERT

TRANFERT\$ $\Delta$ File Code/File Code(minor),

This allows a User Program to "Transfer To" the Communications Director a unit or units as indicated by the file codes. The Communications Director will retain control and monitor them for unsolicited input messages.

## TRANSFERF

TRANSFERF\$ $\Delta$  File Code

This command enables a User Program to obtain a Unit from the Communications Director which had been previously "TRANFERT"ed. The unit will be obtained and associated with the designated file code.

## 8.8 Communications Facility Assignment

A number of Communications Facility Assignment software routines provide orderly control of the system's communication hardware and some software as well. These Facility Assignment routines provide the following:

- A drum record of the hardware currently available.
- A list of hardware currently in use.
- Satisfying a user program's request for communication unit hardware.
- A means of allowing some hardware to be used by more than one program.
- On other hardware, insuring that only one program uses it at a time.
- The user to mnemonically request communication units.
- The assignment of alternate units if the requested is unavailable.
- The correlation of certain software or user routines with the hardware communication units.
- Automatic loading into core of the above mentioned routines.
- "DIAL" functions if specified on the "ACQ\$" statement.
- "GROUP" assignment of communication units, if requested and if possible.
- Insuring that hardware and software usage conflicts do not develop.

The Communications Facility Assignment routines have to perform the above for both Interface Level 1 and Level 2, since both may be in operation at the same time.

The control of the Facility Map, method of handling the "ACQUIRE" and "RELEASE", and the core memory tables (Unit Control Block and CTM Control Block) are essentially the same for both levels. The major difference is that Interface Level 2 has to tie in additional Software routines such as the Communication Handlers and the various portions of the Communication Director.

## Communications User Interface Level 2 Facility Assignment

The Level 2 Facility Assignment will be explained in a general way, to present an overall picture of the communications control. The Facility Assignment is composed of numerous routines, these will not be distinctly broken down at this time. The tables (Facility Map, UCB and CTM CB) explained in an earlier section are quite closely involved in the Facility Assignment.

Initially at System Generation time a Communication Facility Map is generated. This map is drum stored. It is at this time when 5 character mnemonic names will be linked to a specific communication unit, with possibly some alternate units.

### ACQUIRE

Acquire is obtaining the assignment of a desired unit or units to satisfy a user program request. The user program will execute an "#ACQ" control card or an "ACQ\$" internal program statement, both of these accomplish the same thing. The ACQUIRE specifies either a 5 character mnemonic which refers to a pre-established unit name listed in the facility map or an element name which is composed of one or more 5 character mnemonic names. An option G character in the Acquire statement indicates that an element is named. The G group option is necessary if multiple units are desired for one Acquire statement. When the G option is present, Facility Assignment will perform a search for the named element. First it will search the "JOB" library, then the "GROUP" library and finally the "SYSTEM" library. Note there may be more than one element with this same name, but the first which is located will be the one that is utilized. The 5 character NAME or NAMEs found in this element will now be used, the same as if they had been listed in the ACQUIRE statement.

Now that the actual NAME or NAMEs are known, communication facility assignment will proceed to use them for its first reference to the Facility Map. The map will be searched for a match to the specified name, when it is found, the next word will be used to locate the unit descriptor and the handler initialize descriptor. These descriptors are also located in the Facility Map. The unit descriptor contains available or unavailable indications (unavailable is already in use or down) for the unit. If available, the CTM descriptor (also in Facility Map), specified by the unit descriptor will be referenced. The CTM descriptor contains information which indicates: available or unavailable, is or is not presently being used by another unit, and whether it can or cannot be used by more than one unit at the same time. If the CTM is available and not in use, it will now be marked as in use and a CTM Control Block will be formed in core memory. A Unit Control Block will also be formed in memory at this time and the unit descriptor will be marked as unavailable. As the CTM CB and UCB are formed in memory they will be linked together, each containing the address of the other; also, an entry is now made in the unit usage section of the facility map. This is a two word entry with the first word containing the core address of the UCB and the second word contains the pointer

word which was found and used following the 5 character name, it pointed to the unit descriptor which was just used and to a handler initialize descriptor. Note, if the CTM was in use and could have more than one unit using it at same time, the CTM CB would not be formed at this time, since it is already in memory. The linking (CTM CB - UCB - UCB and UCB - CTM CB) would still take place, also the usage entry would be listed.

Now the CTM CB and UCB have been set up, with everything successfully up to this point, the facility assignment will now proceed with the necessary handler control processing. As mentioned previously, a pointer to a handler initialize descriptor has been found following the 5 character mnemonic name. This Handler Initialize Descriptor (it is in the Facility Map) will be located. If some set up or initializing is necessary this descriptor will contain the drum address of the Handler Initialize routine, it will be loaded and control given to it. When the Handler Initialize is finished, control will be returned to facility assignment. Facility assignment will now locate the "Handler Locate Descriptor" as listed in the Handler Initialize Descriptor. The Handler Locate Descriptor contains information which indicates if the handler is presently in core and how many users it has, also where it may be found on drum. If it is not in core, it will be loaded, its core address and a user count of one will be stored into the Handler Locate Descriptor, if already in core the user counter would simple be increased by one. Now that the handler is in core its address will be stored in the CTM CB.

The above indicated the desired unit was available, it did not conflict with the unit capacity of the CTM, thus the unit was able to be successfully assigned. But, if all of these conditions could not be met, the assignment process would be halted and the previous environment would have to be re-created. Such as if the CTM was previously being used by another unit with a different handler and since a CTM can only be associated with one handler at a time; this assignment of this new unit would have to be halted, the UCB destroyed, the usage entry deleted, and the CTM CB - UCB - UCB chain recoupled. Facility assignment would now go back to the NAME pointer section of the facility and if an alternate pointer was available, it then would try to assign the unit it pointed to (such as same type of unit but a different CTM). If there are no alternates or if none of the alternates can be acquired, an unsuccessful status will be returned to the user program.

If the ACQUIRE requested a group assignment after the facility assignment processing for one unit is successfully completed, the next mnemonic name for the next unit will be obtained and an assignment for it will be processed. In a group assignment with option "A", all units named in the element will have to be acquired successfully or else none of them will be acquired for this ACQUIRE statement. During the facility assignment for this Absolute Group, if some units are successfully assigned and then a unit is found which cannot be assigned; the previous assigned units of this group will be immediately released.

If the ACQUIRE requested a group, but did not specify "A" absolute, any or all units which may be successfully assigned will be acceptable.

Thus facility assignment will alter its processing for group acquires accordingly.

"D" for DIAL is another option of the acquire statement. This means after the unit has been successfully acquired as explained above, facility assignment should initiate and completely process a dial operation using the dial number contained in the Unit Control Block. Note, if this unit is not connected with automatic dialing equipment, the console operator will be instructed to manually dial the number and indicate when it is completed. If the dial connection cannot be successfully completed, it is as if this unit cannot be acquired, thus the original environment will be restored and an attempt for an alternate unit will be made (it perhaps will have a different number).

The "X" option is the same as the "D" option except an external number has been specified. The external number is contained in the ACQUIRE statement.

After facility assignment has successfully assigned the unit or units (with successful dialing, if specified) the UCB or UCBs will be linked out of the designated file code position of the task addendum. If a unit or units were previously assigned to this file code, the newly acquired units will be linked following those previously acquired. It is at this time when a successful status will be returned to the requesting program.

#### RELEASE

RELEASE is the releasing or making available of a previously acquired unit or units. The present user has no further need for a communication unit, so it is released, another user executing an acquire may now have this unit assigned to his program.

Facility assignment routines perform the necessary software functions to execute the RELEASE. Returning control back to the user after the release is complete.

Facility assignment uses the file code to obtain the address of the Unit Control Block which is to be released. The communications unit usage list (in Facility Map) will be searched for a UCB address equal to the one being released. When it is found the other word in this entry of the usage list provides the pointers to the Handler Initialize Descriptor and Unit Descriptor.

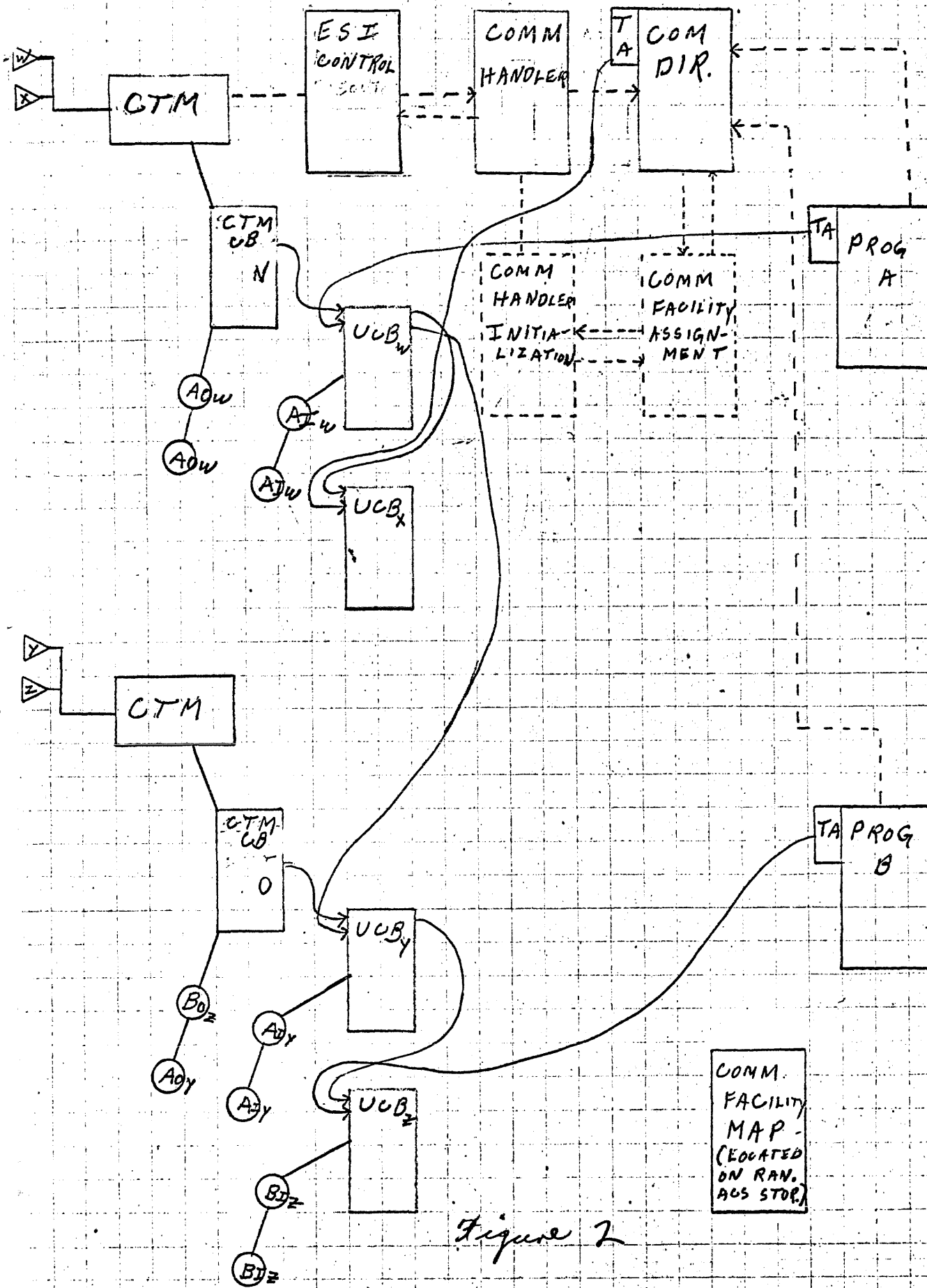
The unit usage entry will be cleared. The Handler Descriptor may be located by following the pointer through the Handler Initialize Descriptor. The Handler Descriptor number of users count is decreased by one. If it is now 0 the handler core address will be cleared in the Handler Locate Descriptor and the core occupied by the handler will be released. Note, this handler will have to be reloaded if some latter "ACQ" requires it. Now the other pointer will point to the Unit Descriptor, the unit will be marked as available and the CTM Descriptor will be located. If this is the last unit using the CTM,

both the UCB and CTM CB will be released, the CTM will be marked as available with no users. If other units are using the CTM, the CTM CB-UCB-UCB chain linking will be reformed to eliminate the UCB which is being released, then the UCB core may be released. Previous to the release of UCB core the file code to UCB chain will be reformed, eliminating the UCB which will be released. This will complete the RELEASE for one unit. If the RELEASE command specified the release of all units on this file code, the next UCB address would be located and this unit would be released. When the requested unit or units have been released control will be returned to the user program.



## 8.9 Illustrative Example Interface Level 2

This section consists of a User Interface Level 2 example. Actual user commands are shown along with a brief description of the software functions performed as a result of these commands. The main purpose here, is to illustrate the message and control paths and at the same time present the overall Level 2 picture.




## Figure 2 Explanation

The hardware, software and User Programs shown in Figure 2 will be used as an example to explain how the Software System functions.

### LEGEND AND LISTING OF COMPONENTS

#### HARDWARE

-  Send-Receive Communication Units connected by direct wire capable of being polled.
- "CTM" - Input/Output Communication Terminal Module

#### SOFTWARE


- "CTM CB" - A table, one per CTM, containing information pertaining to the CTM.
- "UCB" - Unit Control Block. A table, one per Communication Unit, which contains information pertaining to the unit.
- "ESI Control" - A routine which controls the hardware directly. It does this as per instructions received from the Communication Handler.
- "Communication Handler Initialization" - A routine which will perform intializing and setup operations for the Handler which are required only during initialization.
- "Communication Handler" - A routine which drives or controls the CTMs and their communication units. Polling, accepting input, sending output, editing, translating, packing and unpacking the messages, and transferring the messages to and from the Communications Director are its functions.
- "Communications Director" - A collection of routines which together perform the following:
  1. Queuing or staging of the input and output messages on the Random Access Storage.
  2. Interpretation and eventual execution of the advanced commands employed by the User Programs.
  3. Monitoring for unsolicited input messages. Then identifying them and correlating them with their ultimate user.

- "Communication Facility Assignment" - A routine or routines which locates Communication Facilities for a User Program, sets up UCBs and CTM CB and loads some of the software routines.
- "Communication Facility Map" - A table, located on mass storage, which has a listing of the units, the names they may be called by, the handlers which should be used, UCB information, CTM CB information and a summary of the units which are in use.

#### USER PROGRAMS

- "Program A" and "Program B" are shown in this example.

#### MESSAGES

-  - is a message. The A or B is the User Program that it is for or from. The second letter I or O indicates Input or Output message. The third letter identifies the communication unit that it is for or from. Note these messages are stored on drum queues.

#### LINES

- Solid lines indicate chaining or linking of the various components.
- Dotted lines indicate the transferring of Program Control from component to component.

#### TABLE REFERENCE

- The User Programs may not reference anything outside of their own program.
- The Communication Facility map will be referenced by the Communication Facility Assignment routine.
- The Schedule Lists and Unit Control Blocks may be referenced by any of the software routines, but not by the User Programs.

Now let us make some assumptions prior to a step by step explanation of the events.

1. The Communication Units and CTMs are available and ready to operate.
2. Core memory is completely clear of all Communication routines and tables. User Programs A and B have not been loaded as yet.
3. The Communication Facility Map has been set up on Mass Storage and has been initialized. All units are available, pollable, but their descriptions are set to the poll stop position.

- . User Program is loaded and initiated, it executes an internal: command with
- . ACQ\$-GA, GROUP, A - This is an ACQUIRE
  - G = group acquire
  - A = all units are needed.
  - GROUP = the name of an element which consists of the unit W and Y mnemonic
  - A = User Program A File Code this group should be assigned to.

- . "FACILITY CONTROL" of the Comm Dir is loaded.
- . Is ESI Control in memory.
- . NO - Load ESI Control and initialize
- . "COMM FACILITY ASSIGNMENT" is loaded.

- . The "GROUP" is located.
- . The "W" mnemonic is located in the COMM. FACILITY MAP
- . It points to the UCB Descriptor and Handler Descriptor
- . Is unit available? - YES
- . Form "W" Unit Control Block in core
- . List "W" Unit as unavailable in COMM.FAC.SUM.
- . From the Unit Descriptor locate the CTM Descriptor
- . Is it in core?
- . NO - Form the CTM CB N in core
- . List this CTM CB as in core
- . Place address of CTM CB N in UCB-W
- . Place address of UCB-W in CTM CB N
- . Place address of UCB-W in File Code "A" position of User Prog. A
- . Is the Handler in core?
- . NO - Load the Handler
- . Put address of Handler in CTM CB N
- . Load COMM HANDLER INIT and execute
- . Handler executes a "REQUEST OUTPUT W/O RETURN" for CTM CB N
- . Output Section of Com. Dir is loaded.
- . More units to ACQUIRE?
- . YES - "Y"
- . The "Y" mnemonic is located in the COMM. FACILITY MAP.
- . It points to the UCB Descriptor and Handler Descriptor
- . Is the unit available? - YES
- . For "Y" Unit Control Block in core
- . List "Y" as unavailable in COMM. FAC. SUM.
- . From the Unit Descriptor locate the CTM Descriptor
- . Is it in core?
- . NO - Form CTM CB O in core.
- . List this CTM CB as in core.
- . Put address of CTM CB O in UCB-Y.
- . Put address of UCB-Y in CTM CB O.
- . Put address of UCB-Y in UCB-W

\* Note: Two units have now been assigned to File Code A of User Prog A.

- . Is the Handler in core? YES
- . Mark Handler as having another user.
- . Put address of Handler in CTM CB O.
- . Load COMM HANDLER INIT and execute
- . Handler executes a "REQUEST OUTPUT W/O RETURN" for CTM CB O.

- . Move units to ACQUIRE - NO.
- . Return to User Prog A with Successful Code

"SEND\$A A, # of words, buffer base" - A good morning message is sent by prog. A to all units on File Code A.

- . The good morning message for UNIT-W will be passed immediately to the HANDLER.
- . The handler will translate and unpack the message.
- . The handler will direct ESI Control to initiate the transmission of the output message to UNIT-W.
- . The handler will test for units to Poll? - NO
- . The handler will idle.
- The good morning message for UNIT-Y will be passed to the HANDLER.
- . The handler will translate and unpack the message.
- . The handler will direct ESI Control to initiate the transmission of output message to unit-Y.
- . The handler will Test for units to Poll? NO
- . The handler will idle.

"POLLY\$AA," - Start polling for USER PROG. A

- . POLL section Com Director in core? NO - Load it.
- . Set UCB-W to poll on position
- . Will handler get control again?  
YES
- . NO-Q REF Handler for CTM CB N
- Is there output for this - CTM CB?  
YES
- . NO - Is there input for this CTM CB?  
YES
- . NO - Are there units to poll?
- . NO - the handler will IDLE
- YES - POLL unit
- Positive reply from POLL?  
YES
- NO - other units to POLL?  
YES
- . NO - Handler will time delay itself
- After time delay
- YES - Input message is coming.
- ESI Control will pass message to the Handler
- . Handler will translate and pack the message
- . Handler passes input message to Com. Dir.
- . Is the Receive section of Com. Dir. in memory?  
YES
- . NO - Load Receive section
- Queue this input message on drum with it linked out of UCB-W.
- Return to normal handler cycling.
- Send routine will transfer output message from CTM output drum queue to the handler.
- . The handler will translate and unpack the message.

- The handler will direct ESI Control to initiate the transmission of the output message to the unit.
- Return to normal handler cycling.

\* NOTE The above handler cycling will continue as long as input, output or polling operations are needed.

- Set UCB-Y to poll on position.

- Will handler get control again for-CTM CB 0?

YES

- NO - QREF Handler for CTM CB 0

\* The handler cycling explained above will also be initiated for CTM CB 0. The same re-entrant handler coding will be used but different CTM CBs and UCBs will be referenced. Thus multiple handler cycles may be in process concurrently. Even running concurrently with User Programs and some portions of the Comm. Director.

• "RCV\$A,# of words, buffer base," For User Prog. A.

- Receive routine in memory?

- NO - Load it.

- YES - Get a UCB address from the File Code A position of the Task Addendum.

- Was the RECEIVE Command of the GENERAL type?

- YES

- NO - Is the minor File Code number equal to this UCB position in the UCB chain from the Task Addendum (01 = 1st UCB)?

- YES

- NO - Is there another UCB in the TA-UCB chain?

- NO

- YES - GET the address of the next UCB

- Return

- Is there an input message queued against this UCB?

- YES

- NO → Is there another UCB in the TA-UCB chain?

- NO

- YES - Get the address of the next UCB.

- Return

- Return to User Program with Unsuccessful Status

- Is there an input message queued against this UCB?

- NO

- YES

- MOVE the queued message from the UCB queue to the User Prog buffer.

- Return to the User Prog with Successful status

\* Prog. A may now continue on its merry way Sending and Receiving from either or both of its communication units. The polling of these units will also continue.



- . Omega Selection recognizes User Prog. B. Job.
- . Omega Selection may cause some of the previous communication routines which are not now being used to be dropped from core memory.
- . User Prog. B is loaded into memory.
- . ACQ\$- ,NAMEX,B - ACQUIRE Command executed by Prog. B
  - . This Acquires UNIT X similar to the previous with some exceptions. Such as only one unit was requested and it was specified by a UNIT NAME rather than by a GROUP NAME. This eliminated the locating of the GROUP defining element. Also note some of the routines will not have to be loaded since they are already in memory.
- . "ACQ\$- , NAMEZ,B" - ACQUIRE Command executed by Prog B.
  - . Same as the ACQUIRE for UNIT-X except the UCB-Z address is written into UCB-X rather than into the Prog. B task Addendum. This has formed a 2 unit chain. File Code B01 refers to UNIT-X and File Code B02 refers to UNIT-Z.
- . "POLLY\$B/01," USER PROG B.
  - . This will set UCB-X to the poll on condition. The handler will not have to be started since it is already cycling for Prog. A. On the next poll cycle it will poll both units.
- . "TRANFERT\$B/01," - Transfers UNIT-X from Prog B to the Comm. Director.
  - . The UCB-X address is written into a Task Addendum type location in the Comm. Director.
  - . The TA-UCB chain position of UCB-X is cleared to indicate it is now an end of chain unit.
  - . The UCB-Z address is written into the Prog. B. Task Addendum File Code B position.
  - . Note UNIT-Z file code identification has just changed from B02 to B01, its position on the TA-UCB chain has changed.
- . "POLLY\$B/1" User Prog. B.
- . "SEND\$B/1, # of words, buffer base"

Certain timing conditions would have permitted the operations which were explained above. To place the hardware, software and User Programs in the state as shown in Figure 2.

## 9.1 LOADER

### 9.1.1 General Description

The Loader is a processor which provides a flexible and efficient means of collecting independent relative binary (RB) elements to produce an absolute object program for execution as a task under control of Omega. An RB element is an intermediate output code generated by all system compilers as a result of translating a group of source language statements. An RB element is not executable but may contain references (external references) to other RB elements and may itself contain a definition (external definition) which is referenced by another RB element. The Loader may serve to join RB elements generated from source statements expressed in FORTRAN, COBOL, Assembly language, etc. The process of joining RB elements is called collection. The Loader does not actually load a program into memory for execution but constructs the entity which may be read and executed. This organization facilitates compilation and debugging of small parts of a total program and combination of these individual parts for execution with recompiling the entire set of individual parts. An absolute program is an entity with no unresolved references which can be read in and executed. It may be read into any memory area for execution without modification of instructions. Its relocatability is inherent from the relative index register and its device independence with regard to system references. The relationship of the Loader with the source code and absolute code is shown.

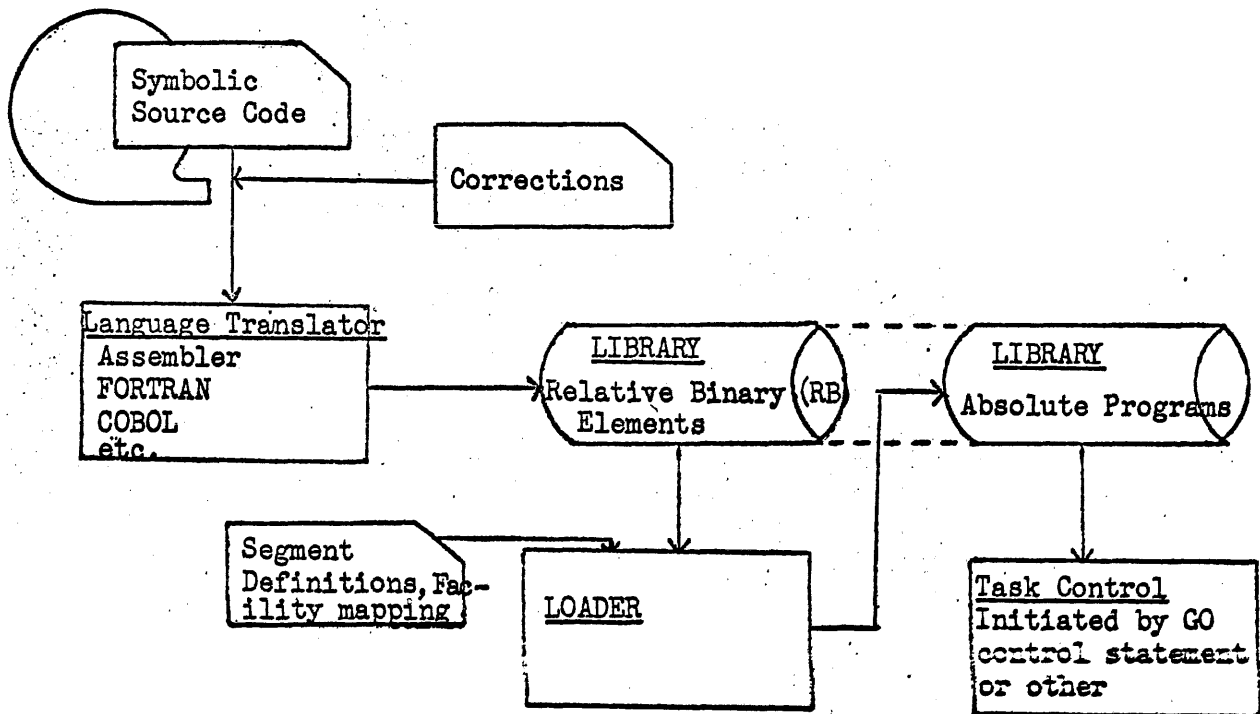


Figure 3-1

Separate elements existing in the job, group, or system libraries are collected in constructing an object program. Elements are collected on the basis of an external reference in one element which can be satisfied by an external definition within a second element. The Loader may be directed to include or exclude specific elements by secondary control statements.

The basic output of the Loader is an absolute object program. The program is entered into the job library under the name specified by the user. Optional output includes a list of labels and tags contained in the program for utilization in testing procedures. Error messages and/or a storage layout listing may be obtained as a hard-copy record of the collection process. The Loader can also transfer the secondary control language as a job library element for subsequent reference.

The order of user specified elements within a segment will be maintained as specifically named by INCLUDE statements. All elements included by a library search are located in the highest level segment from which it can be referenced by all elements.

A starting point is determined during text modification or by an ENTRY control statement. Any element may specify a starting address; the Loader will accept the first address encountered.

#### 9.1.2 Primary Control Statement

The Loader is scheduled and activated in response to a LOAD control statement in a job input stream. Information on the LOAD card is comprehensive enough to direct the collection and loading of most programs.

.LOAD - This statement calls for the collection of an RB element for processing to generate an absolute object program. The format is:

#LOAD  $\Delta$  options  $\Delta$  name/version RB, name/version object, name/version secondary language

Name/version RB of element to be collected. The absolute program is identified as name/version object.

Options are:

- U - Save the secondary language element in the job library. The third field of the specifications list is given as the element name.
- L - Produce a complete listing. This listing is a summary of the memory space used by each element included in the program. The relative location of each external definition will also be listed.
- N - Produce no listing. The N option will be overridden if diagnostic messages are to be produced. If neither an N

or L option is indicated, summary information only will be printed.

Y - Accept the program for execution even though errors were detected during collection or elements marked as in error were included. If the errors prevent the production of an absolute program, the Y option will not be effective.

X - Abort. Do not execute the remainder of the job (i.e., skip to next JOB card) if errors are detected. If neither a Y or X is given, the event of an error will inhibit execution of the program but will perform the remainder of the job.

Z - Inhibit the formation of tabular information to be given to the test system for test procedures.

### 9.1.3 Secondary Control Statements

Construction of segmented programs or particular collections are described by a secondary control language which normally follow the LOAD statement. The Loader or utility package can be directed, however, to place a set of statements into a job library as an element. Subsequent execution of the Loader may utilize this element as directed by the LOAD control statement.

The secondary control language recognized by the Loader allows description for even the most complex programs. The user can enter these control statements with the input stream for each collection or he can specify a library element of control statements to control the collection.

The control statements recognized by the Loader are INCLUDE, EXCLUDE, SEGMENT, ENTRY, and EQUALS. All secondary control language cards are blank in column one to distinguish them from primary control cards.

1. SEGMENT - The Loader provides a straightforward means of constructing overlay segments. For each segment, the user prepares INCLUDE and/or EXCLUDE control statements to specify the relocatable elements to be included within that particular segment. These statements are preceded by a SEGMENT statement specifying the name of the segment and its logical origin.

When a segmented program is called for execution only the main segment is initially loaded. There are two ways by which other segments may be loaded. The direct method is whenever the user makes a direct call to the overlay supervisor specifying the segment to be loaded and the location to which control is to be transferred. The second method is indirect and provides for automatic loading of a segment referenced by a jump type command, whenever the segment is not in core. The mechanics for such loading are set up by the Loader and carried out by the overlay supervisor. The Loader replaces the address portion of the jump command with the address of an entry vector. The entry vector in turn jumps to the location of the externally defined symbol. If the overlay supervisor is entered, it loads

the necessary segment and transfers control the same as the original jump intended. All registers are preserved by the process, and all necessary entry vectors are reset at the loading of any segment.

The SEGMENT statement is used to declare the beginning of a new segment. The format of the SEGMENT statement is:

```
SEGMENT segname1, segname2
```

or

```
SEGMENT segname1,(segname2)
```

The field segname1 is the name of the segment and must be specified. The field segname2 (not enclosed in parentheses) specifies that the segment identified by segname1 is to originate at the same location as does segname2. (Segname2) (enclosed in parentheses) specifies that the segment identified by segname1 starts immediately after segname2. If the second field contains a series of segment names enclosed in parentheses and separated by commas, the Loader starts the segment identified by segname1 immediately following the highest location occupied by any of these. If the field segname2 is void, the segment identified by segname1 is originated immediately following the preceding segment.

#### Example of Segmentation

The following example of segmentation illustrates the control statements and resultant memory layout.

#### Comments

```
#LOAD P, P1
```

The segment A is originated immediately following the preceding segment.

#### Comments

```
INCLUDE A1, A2, A3
```

Identifies specific elements to be collected.

```
SEGMENT B,(A)
```

The segment B is originated immediately following Segment A.

```
INCLUDE B1, B2  
SEGMENT C,B
```

The segment C is originated at the same position as segment B.

```
INCLUDE C1, C2, C3  
SEGMENT D, (B,C)
```

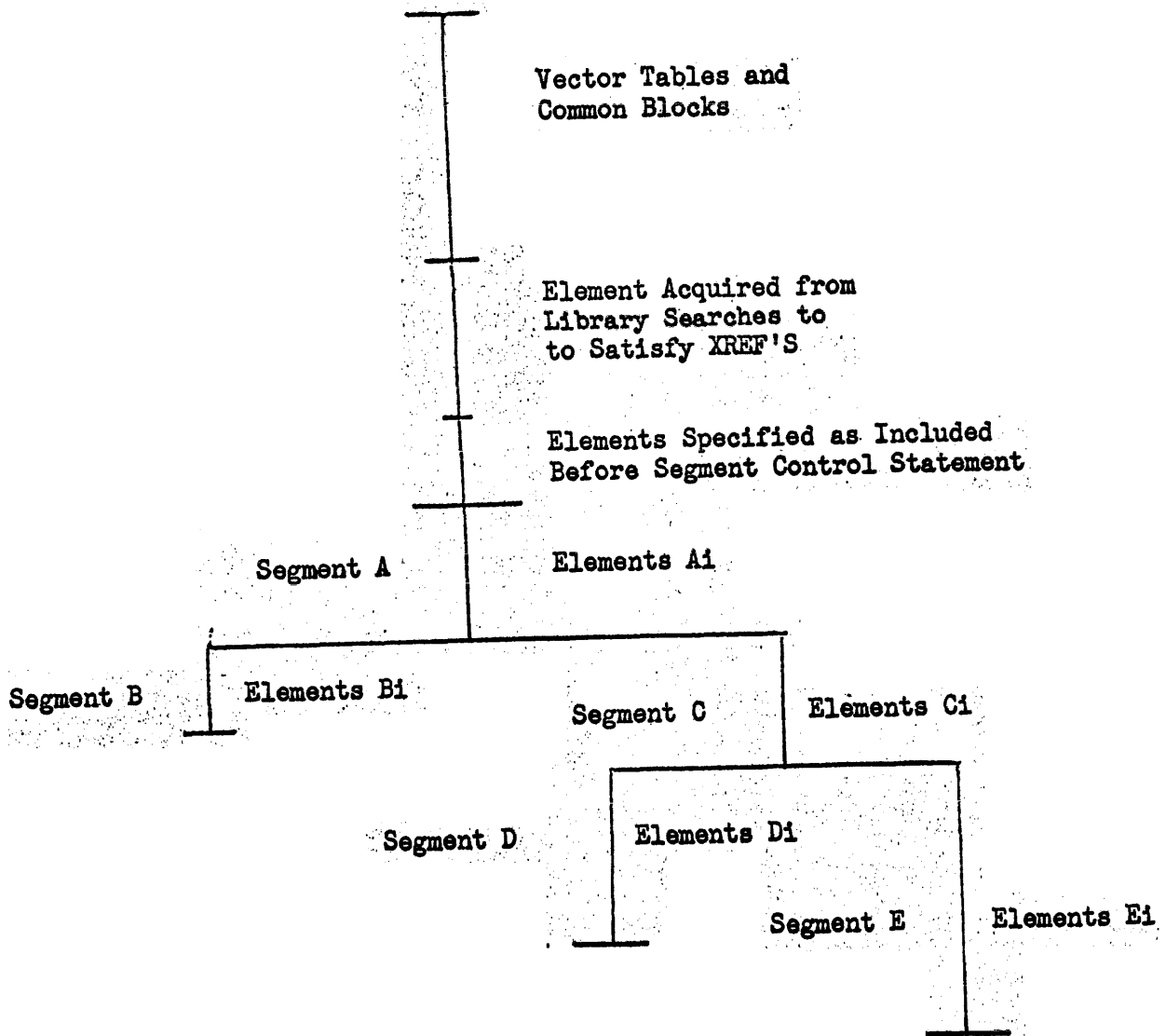
The segment D is originated following the longest of segments B and C.

```
INCLUDE D1, D2, D3  
SEGMENT E, D
```

The segment E. is originated at the same position as segment d.

```
INCLUDE E1,E2,E3,E4  
#GO P1
```

This particular set of control statements would result in the following memory structure.



2. INCLUDE - This statement allows the user to specify which elements will be collected to create a segment.

```
INCLUDE      element/version, element/version . . .
```

Each element/version field specifies an element to be included in the collection.

In order to utilize the FORTRAN "block data" feature the INCLUDE statement can be used to identify "block data" elements (those without external references or external definitions) which have initial content for allocated "common" blocks.

If a "block data" program specifies initial content for two common blocks, one of which is referenced in the program being allocated while the other is not, the latter is simply ignored by the Loader. If more than one "block data" program specifies initial content for a given "common block" the value established by the second will be accepted.

3. EXCLUDE - This statement is essentially the inverse of the INCLUDE. It allows the user to state explicitly which elements are not to be included in a collection. All RB elements implied by the collection other than those excluded are accepted.

```
EXCLUDE      element/version, element/version . . .
```

Each element/version field specifies an element to be excluded from the collection.

4. ENTRY - This statement provides the user the capability of overriding the starting address specified via the assembler, or the entrance to a main program generated by FORTRAN or COBOL.

```
ENTRY      name
```

where name is an externally defined symbol

5. EQUALS - This statement allows the user to define an external reference at the time of collection.

```
EQUALS     name/value, name/value . . .
```

Each name/value field specifies a symbol to be defined and the value to be used. Value may be an octal or decimal integer, or a symbol \* integer. Such symbols must be externally defined by one of the collected elements.

6. Miscellaneous - Primary control statements such as ASG can be submitted as part of the Loader's secondary language. These primary control statements will conform to the normal format except for a blank in column one. The Loader will accept such cards and form them into a block of statements associated with the absolute program. These statements will thereafter be utilized by Omega whenever the absolute program is named for execution. By this means, for example, the ASG statements necessary for efficient selection and execution of the program can be permanently declared and associated with the program.



### 9.1.4 Relative Binary (RB) Code

The following code must be generated by system compilers as an RB element. The basic order and structure is retained in the libraries and on external media. An RB element is composed of a preamble and text sections.

#### Preamble

Preamble is a table of arbitrary length consisting of six distinct lists: Header, Entry Definition List, Externally Referenced Name List, Control Counter List, INFO List and optional Symbol Definitions. The Header is always present and indicates the presence and placement of the remaining lists.

#### 9.1.4.1 Header

1	Error Indicator	0			
N	N	N	N	N	1
N	N	N	N	N	2
V	V	V	V	V	3
Increment to EDEF					4
# of EDEF					5
Increment to XREF					6
# of XREF					7
Increment to CC					8
# of CC					9
Increment to INFO					10
# INFO					11
Increment to SDEF					12
# SDEF					13
Increment to TEXT					14
Length of TEXT					15

where - Word  $\emptyset$  is an RB indicator and an error indicator of compilation.

N - N is a 1-10 character alphanumeric name left justified.

V - V is a 1-5 character alphanumeric version left justified.

Word 4, 6, 8, 10, 12, 14 are increments from the file base to the appropriate list.

Words 5,7,9,11,13, are the number of entries in each list for Entry Definitions (EDEF), External References (XREF), Control Counter (CC), INFO statements, and Symbol Definitions (SDEF).

Word 15 is the length of the text.

Entry Definition . Used to mark points within an element which may be referenced by other elements.

N	N	N	N	N	0
N	N	N	N	N	1
t		CC	Value		2

where N - N is a 1-10 character alphanumeric name left justified.  
 t in bit 2<sup>29</sup>th of Word 2 indicates whether value is relative to a control counter. 2<sup>29</sup>th =  $\emptyset$  if value is relative. 2<sup>29</sup>th = 1 if value does not require relocation.

CC for t =  $\emptyset$  is the number of a control counter, the beginning value of which is to be added to the entry definition value.

Value is the 15 bit value defined for the named enter.

Externally Referenced Name Entry . Contains a list of labels or tags referenced, the definition of which is not contained in the current element. Definition will be made at collection time by inclusion of necessary elements to satisfy references. The number is implied by position of entry in list.

N	N	N	N	N	0
N	N	N	N	N	1

where N - N is a 1-10 character alphanumeric name left justified.

Control Counter Entry . Specifies the number of consecutive words of core required by code operating under control of this counter. During load process control counters are fixed to the starting address of assigned area. Control counter number is implicit by order within list starting with zero.

	# of words
--	------------

Information Entry (INFO) . INFO statements describe areas of common core storage between individual compiled elements.

N	N	N	N	N	0
N	N	N	N	N	1
		GN		CC	2

where N - N is a 1-10 character alphanumeric name of common area left justified.

CC is the number of control counter giving its size.

GN Group number = 0 Named Common

Group number = 1 Blank Common

Symbol Definition Entry . Contains an optional list of labels and and tags within element used for diagnostic purposes.

N	N	N	N	N	0
N	N	N	N	N	1
		CC		Value	2

where N - N is a 1-10 character alphanumeric name of symbol left justified.

CC is control counter number to which this symbol is relative.

Value is relative address within control counter of symbol.

#### 9.1.4.2 Text

Each group of element text is 14 words in length and composed of two sections of variable length: data words, unmodified instructions and/or constants appear from right-to-left from end of image. Modification information extends from left to right in an uninterrupted stream of bits which specify the modification required to load associated indicated data words. The modification information determines the number of data words contained in image.

1 ----- 14 words of text ----- 14



Modification Information . Determines any modification of data words, the loading address at which one or more data words are to be stored and the starting address of an element.

There are four cases of major control in the bit stream from an initial state.

- 00 End of stream: start a new image
- 01ar Address: "r" is a reference number of 2-12 bits and "a" is a 15 bit unadjusted address. If  $r > 511$ , then  $a + cc(r-512)$  is start address of element.
- 1 $\emptyset$  No modification. Load data word
- 11m Modify according to the modification string "m" then start.

The modification string for a specific word is a sequence of the form

i1i1 . . . i $\emptyset$

where each i specifies a single modification. All but the last modification are followed by a "1". The modification string is ended by "0". Thus 00,10 combinations end the modification string immediately.

Each instruction modification is of the form:

f s t r

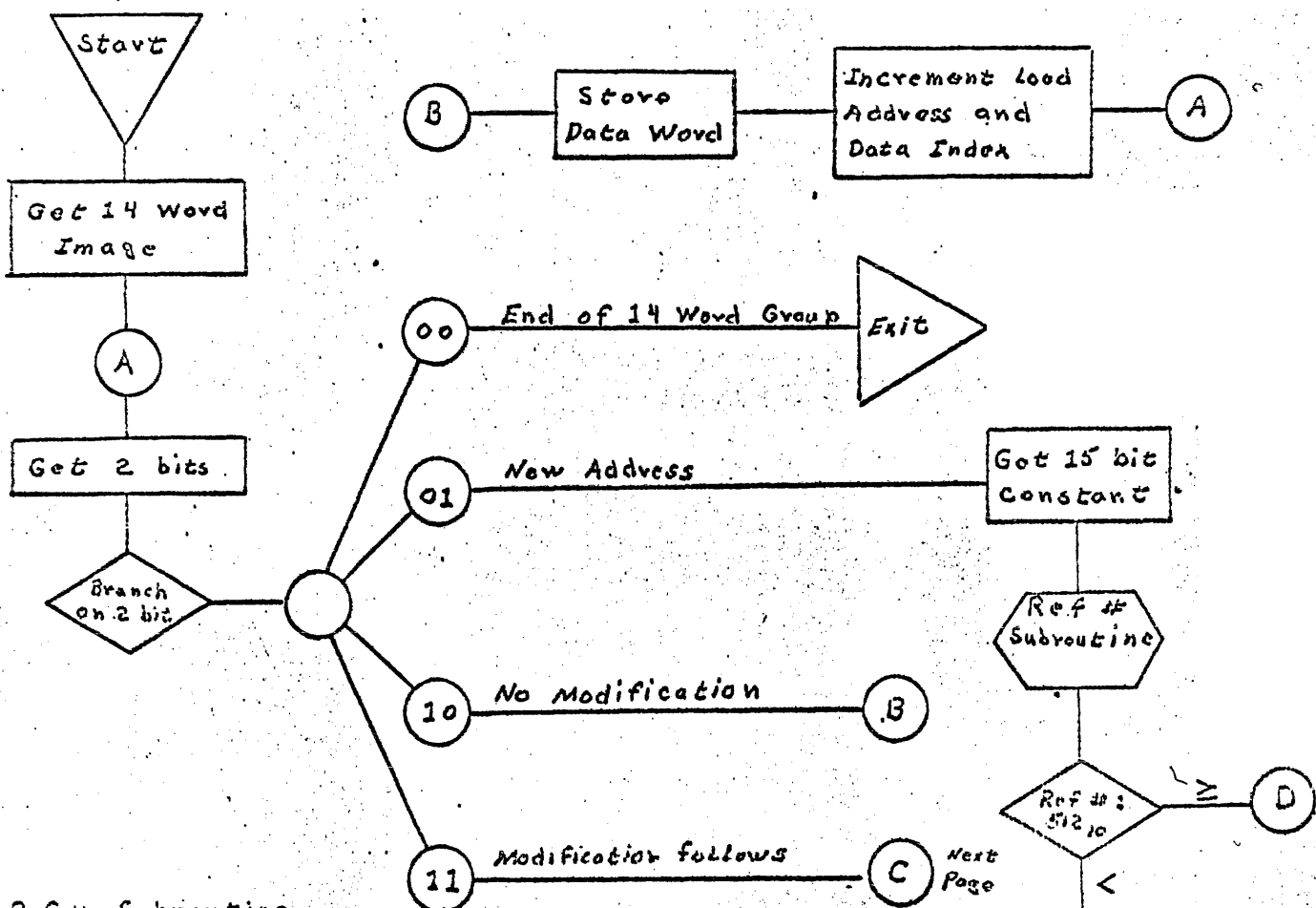
where f defines the field to be modified, and is of variable length. "0" modify bits 14-0, "1" modify bits 29-15, 11xxxxxyyyyy modify bits x-y.

s is sign of modification, "0" for addition, "1" for subtraction.

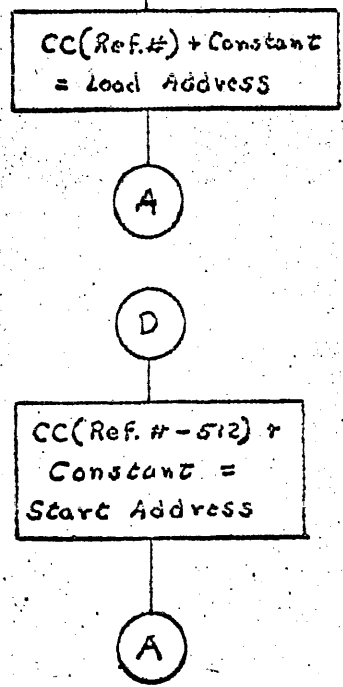
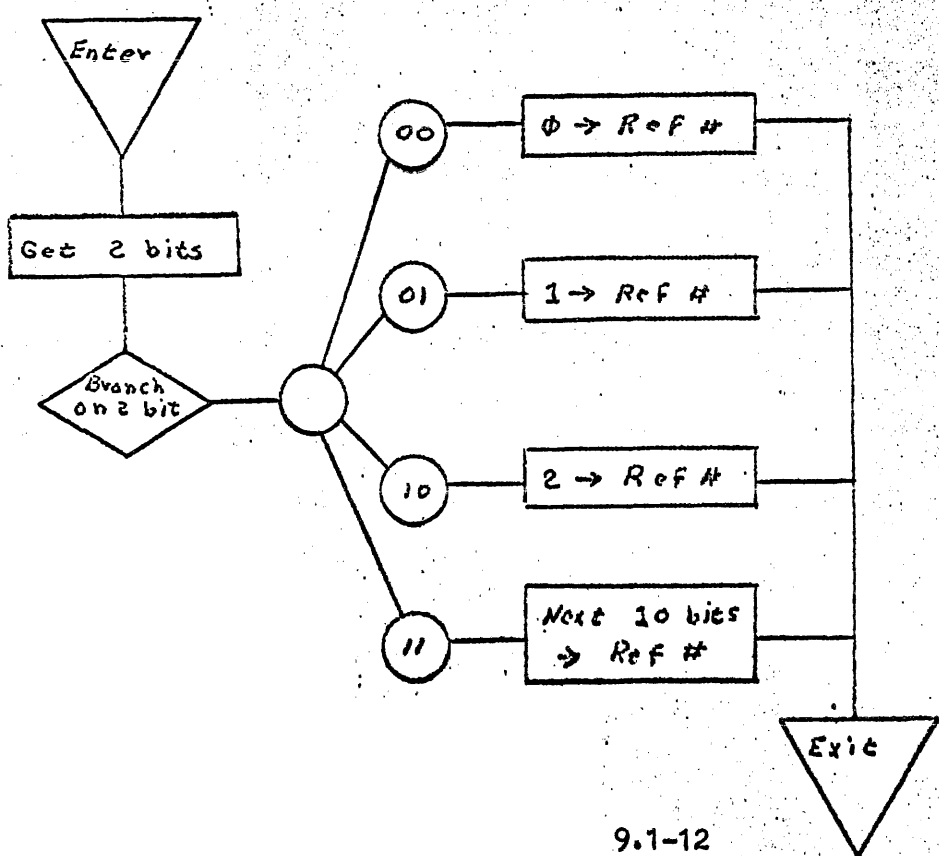
t is the type of modifier, "0" for control counter, "1" for external reference.

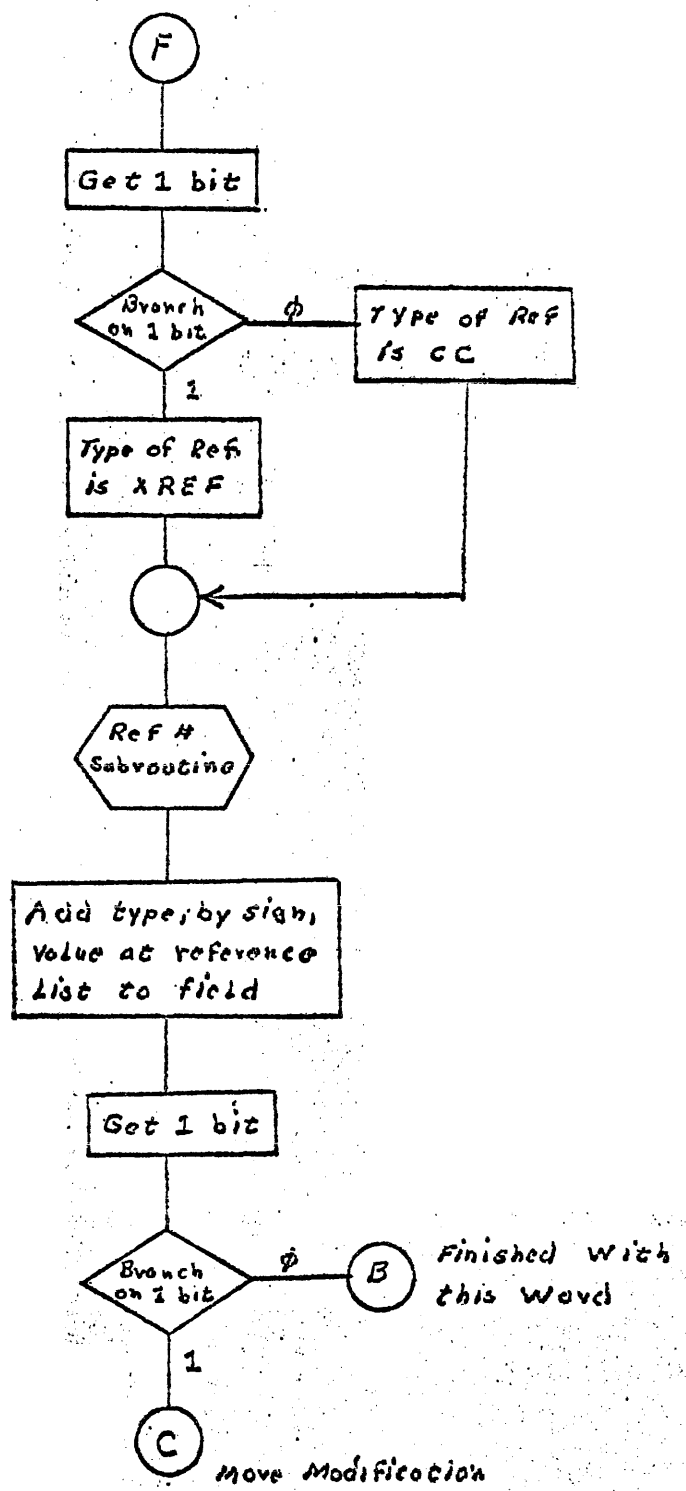
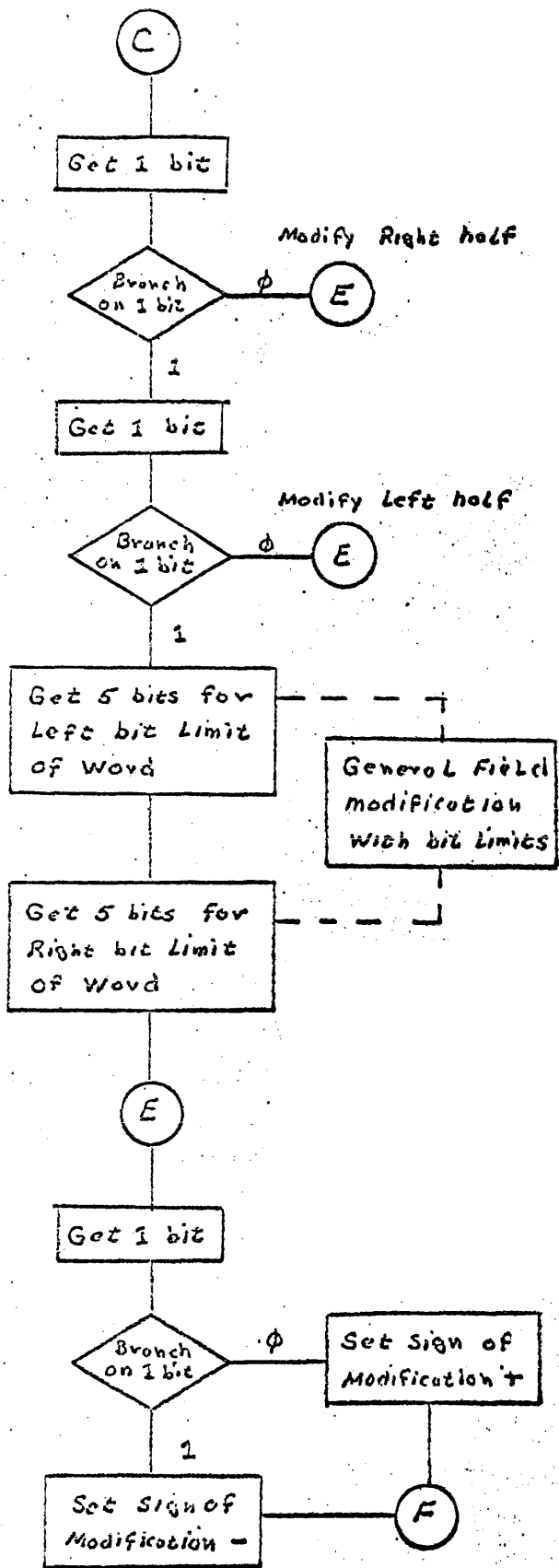
r is a reference number, specifying the number of the control counter or externally referenced name to be used. 00 use 0, 01 use 1, 11x use x which is ten bits.

Modification of data words is illustrated by the accompanying flowchart.



Ref.# Subroutine





### 9.1.5 General Description of Loader Operation

The Loader has access to all elements established in any of the three basic libraries (Job, Group, System). The Loader processes the LOAD statement and assumes that the name/version in the first field is the basic element to be included in the total collection.

All secondary control statements from the primary input stream are then processed, building the necessary lists for the collection process.

After all primary control statements are processed a pass is made on the table of contents (TOC) of all elements in the library complex until all elements stated as being included via the LOAD or INCLUDE statement are located. The search for the included elements is made in a prescribed manner. The elements within the Job library are checked against the elements included. Next the elements within any Group library linked to this job are checked. Finally, if all elements are not located the System library is checked.

As the libraries are being checked a list of parameters necessary to retrieve all the external definitions within the library complex is formed.

If source elements are among those elements stated as being included, the source element is retrieved and the control statements within the element are processed as if they were entered via the primary input stream.

All external definitions from elements in the Job library are collected and placed on the Loaders scratch file, (ZH), since there is no complete list of Job library element external definitions on the job library.

The external definitions for the element stated as being included are retrieved and established as a list on the Loader scratch file. These external definitions will be the first checked in an attempt to satisfy an external reference.

All elements stated as being included are then retrieved and processed one at a time. Processing external references involves the satisfaction of any external reference with the external definitions available. If the element containing the external definition satisfying an external reference is not currently marked for inclusion in the collection it is done at this time. The external definitions are checked in the following order (Included element EDEFS, Job Library EDEFS, Group Library EDEFS, System Library EDEFS). A list of the satisfied external references and their value is maintained on the Loader scratch file.

The processing of common area definitions involves the comparison of a common area name against those already processed. If a name matches, the larger of the two areas defined is included.

A count of the length of all segments is maintained by the accumulation of the value of the control counters contained in each element processed.

An automatic load entry is created by each reference that crosses from one segment to another subsegment.

After all elements have been pre-processed, the bases of all segments are calculated and a base is assigned to each control counter.

If a secondary output was requested the control statements necessary to recreate the collection are put out as a source element.

If the symbol definitions are to be processed, they are read in from each element and modified by the base assigned to the element. They are then placed at a calculated position on the scratch file such that they fall at the end of the absolute element to be created.

The text for each element included is then read in and modified. All automatic load entry addresses are assigned at text modification time. The modified text is buffered out to a pre-calculated area on the scratch file to form the absolute element.

After the text of all elements has been modified the absolute element is transferred to the job library via the Library Service routine in Omega's secondary exec.



#### 9.1.6. Input Element Format

The following diagrams describe the formats of the elements with which the Loader operates to form an absolute element. The element may be on any of the libraries within the library complex (Job, Group, System).

REPRESENTATION OF INDIVIDUAL TOCS (INTERNAL) WITHIN THE LIBRARIES

A	0	3			ERROR IND.	SOURCE
	1	N	N	N	N	
	2	N	N	N	N	
	3	V	V	V	V	
	4	INCREMENT TO ELT BASE				
	5	UNASSIGNED				
	6	UNASSIGNED				
	7	SUB TYPE		# IMAGES IN ELT		
	10	TOTAL ELT LENGTH				
B.	0	2			ERROR IND	ABSOLUTE
	1	N	N	N	N	
	2	N	N	N	N	
	3	V	V	V	V	
	4	INCREMENT TO ELT BASE				
	5	MAX CORE USED		LENGTH OF CONTROL		
	6	# SEGMENTS		# ASG IMAGES		
	7			# SDEFS		
	10	INDEX TO SDEFS				
C	0	1			ERROR IND	RELATIVE BINARY
	1	N	N	N	N	
	2	N	N	N	N	
	3	V	V	V	V	
	4	INCREMENT TO ELT BASE				
	5	INDEX TO XREF		INDEX TO CC		
	6	INDEX TO INFO		INDEX TO SDEF		
	7			INDEX TO TEXT		
	10	LENGTH OF TEXT				

## DESCRIPTION OF TOC CONTENTS

### 1. RB ELEMENT (C)

Word $\emptyset$	A number indicating a RB element. The lower of this word is used for error indication.
Word 1-3	A 10 character name and a 5 character version. All unused portions of these 3 words are space (05) filled.
Word 4	An increment from the base of the job library at which the element can be found.
Word 5	U = an index from the increment in word 4 which relates the start of the XREFS and also relates the length of the EDEFS L = index to start of the cc and relates the length of the XREFS
Word 6	U = index to start of the INFO and the length of the cc L = index to start of the SDEF and the length of the INFO
Word 7	L = index to start of the text and the length of the SDEF
Word 10	Length of the text

### 2. ABSOLUTE ELEMENT (B)

Word $\emptyset$	U = Absolute type number (2) L = Error indicator
Word 1-3	Same as in RB element
Word 4	Same as in RB element
Word 5	U = the maximum amount of core utilized at any one time L = length of control segment
Word 6	U = number of segments L = number of ASG images
Word 7	L = the number of SDEFS
Word 10	Index to SDEFS

### 3. SOURCE ELEMENT (A)

Word $\emptyset$	U = source type number (3) L = Error indicator
Word 1-3	Same as in RB element
Word 4	Same as in RB element

Word 5-6 Unassigned

Word 7 U = a subtype indicating type of source code  
A = normal (20 word card image) B = compressed source  
L = the number of images within the element (compressed or normal)

Word 10 Total element length

PREAMBLE AND TEXT


RB ELEMENT FORMAT

EDEF	N	N	N	N	N	1
	N	N	N	N	N	
	T	CC	VALUE			
XREF	N	N	N	N	N	2
	N	N	N	N	N	
CC	SIZE					3
INFO	N	N	N	N	N	4
	N	N	N	N	N	
SDEF	GN		CC			5
	N	N	N	N	N	
	N	N	N	N	N	
	CC		VALUE			
TEXT						

## RB ELEMENT FORMAT DESCRIPTION

1. EDEF - A three word external definition which defines an entry point within the element or some value defined by the element. It consists of a ten character name and a 15 bit value. If the value is absolute, the sign bit will be set (T) and no control counter (CC) will be present. CC defines the control counter to which this value is relative.
2. XREF - A two word external reference which will be satisfied by an EDEF.
3. CC - A control counter which defines the amount of core occupied by the data under that control counter.
4. INFO - A three word item defining a common area. The name of the common area occupies the first two words. GN is an indicator indicating the type of common, 0 = named common, 1 = blank common. The control counter under which the common is to appear is defined in the lower.
5. SDEF - A three word item defining a position within the element which can be used in debugging procedures. It has the same format as an EDEF.
6. TEXT - Text is grouped into a number of 14 word images which contain modification and instructions.

ENTRY POINT TABLE FOR GROUP AND SYSTEM LIBRARY

N	N	N	N	N
N	N	N	N	N
T		CC	VALUE	
				
77777	IND TO TOC			

ELEMENT EDEFS

REL TO BASE OF TOC. Upon use in the collector loader this value is used to find the toc associated with the particular EDEF.

## 9.1.7 Basic Functions of Loader Phase 1

### A. Inputs

1. JOB, GROUP, SYSTEM LIBRARIES
2. CONTROL CARD AND/OR MAP ELEMENTS

### B. Output Tables and Lists

1. INCT (Include table from INCLUDE STATEMENTS AND LOAD STATEMENTS)
2. SNT (Segment Name Table from SEGMENT STATEMENTS)
3. SMAP (Segment map showing relation of segment bases as defined by Segment statements)
4. EQUT (Equals Table from EQUALS statements)
5. EXCT (Exclude Table from EXCLUDE statements)
6. ENT (Entry definition name from ENTRY statement)
7. ATOC (The table of contents of all items in the INCT table)
8. ED (A list of parameters required to find all EDEFS in the system)
9. Job and Included EDEFS (A list of all EDEFS from included elements and the Job library)
10. Secondary Output (The secondary output consisting of control statements necessary to recreate the collection)

### C. Procedures

The control statements are read from the primary input stream and any source elements included. The above mentioned tables and lists are constructed from these control statements and the elements from the libraries.

The tables and lists built are described below.



LOADER (PHASE 1 INTERNAL TABLES)

SEG#					ORDER COUNT				
N	N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N	N
V	V	V	V	V	V	V	V	V	V
INC TO ELT BASE									
INDEX TO XREF					INDEX TO CC				
INDEX TO INFO					INDEX TO SDEF				
					INDEX TO TEXT				
LENGTH OF TEXT									
TOC DRUM INC (IDENTIFIER)									

(ATOC TABLE)

ATOC is a revised TOC table which is the main output of Phase 1. This table will include an entry for each element to be included in the collection process except those elements included because of an external reference. The segment with which the element is associated is indicated in the upper of word  $\phi$ . The order count is a number representing the order in which this element is to be included.

N	N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N	N
LENGTH					BASE				

(SNT TABLE)

SNT (Segment Name Table) is a list of segment names taken from SEGMENT cards encountered. A 3rd word reserved for the segment length and base is added. This table is related to SMAP in that the procedure for assigning a base is dictated by SMAP. This table is preserved through Phase 3 and a revised version is included in output as information necessary for the load of a given segment.

N	N	N	N	N	N	N	N	N	N
N	N	N	N	N	N	N	N	N	N

(ENT)

ENT is a two word name representing a starting address to be used as the start address of the absolute output. This name will be preserved through Phase 2 where an address will be associated with the name. This address will then be put out as the starting address of the absolute element.

N	N	N	N	N
N	N	N	N	N
V	V	V	V	V
SEG #	CALL ORDER			

N	N	N	N	N
N	N	N	N	V
V	V	V	V	V
SEG #				

N	N	N	N	N
N	N	N	N	N
NE	NE	NE	NE	NE
NE	NE	NE	NE	NE
0 1/4 SEG #	INC			

INCT

INCT (INCLUDE TABLE) is a list of all elements to be retrieved from the library which were specified on INCLUDE control cards. Seg # indicates the segment in which the specified element is to be placed. CALL ORDER indicates the order in which the elements are to be processed. INCT is a temporary table existing only through Phase 1.

EXCT

EXCT (EXCLUDE TABLE) is a table consisting of all elements specified on EXCLUDE control cards. This table exists only during Phase 1. Seg # indicates the segment in which the exclude statement applies.

EQU

EQU (EQUALS TABLE) is derived from the content of EQUALS Card. The first two word name is the name to be equated. The 2nd two word name is the EDEF being used to allocate words 0-1. An indicator is set as these are satisfied. This table exists until Phase 2.

9.1.10 COLLECTOR/LOADER TABLE TRANSITION

BDTOC

4	0	0	0	1	Error Ind	
N	N	N	N	N	N	N
N	N	N	N	N	N	N
V	V	V	V	V	V	V
Increment to ELT Base						
Index to XREF				Index to CC		
Index to INFO				Index to SDEF		
Index to TEXT						
Length of TEXT						

BDTOC is created in Phase 1 during a pass on the system tocs. Its purpose is the processing of block data after the common area bases have been determined. It is preserved to Phase 3 where block data is processed.

The sign bit in word 0 of the toc indicated a block data element. A block data element will contain no EDEFs or XREFs.

Block data is processed by comparing the common names of the block data element with the common names of the collected elements. If a match is made, the data contained under the control counter for that common area will be deposited in the common area defined by the collected element.

If the data length exceeds the common area, the excess data will be discarded.

Block data can only be assigned to named common not blank common.

CODE	SEG #
7-----7	7-----7
CODE	SEG #
7-----7	7-----7

## SMAP

SMAP (Segment MAP) is a table of variable length items describing the relation between segments in regard to base addresses. It is a semi-permanent table that is built as the SEGMENT control card is examined. It is retained until after Phase 2 at which time the length of individual segments is known and a complete picture of the segment allocation can be determined. Referral to SNT is made to obtain the known length of segments. The base is assigned and placed in SNT. Each segment entry is separated from the next by a 7-----7. The code may be any of the following:

- Code 1 Segment base is equal to the base of the segment specified in the lower plus the length of that segment.
- Code 2 Segment base is equal to the base assigned to the segment specified in the lower.
- Code 3 Segment base is equal to the base of the segment specified in the lower plus the length of the longest specified segment.

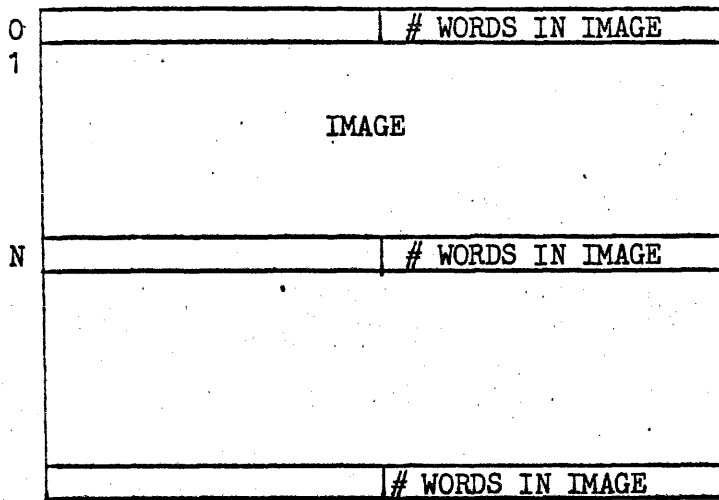
A	FILE CODE	LENGTH
	FILE INCREMENT	
B	FILE CODE	LENGTH
	FILE INCREMENT	
C	FILE CODE	LENGTH
	FILE INCREMENT	
7 7 7 7 7 7 7 7 7		

ED

ED (External Definition parameter list) defines the location and length of all external definitions in the system. Each entry is a two word entry giving the file code under which the external definitions can be found. The length of the data, and the increment to the file base. Normally A would define the parameters necessary to read the included elements external definitions. B would be for the Job library. C would be for a Group library or systems library.

The list is variable in length and is closed out with a word of all 7's.

SOURCE ELEMENT



Source Element Description

The source element is used by the Loader only as an included element containing more secondary control statements necessary for the collection.

Word  $\emptyset$  Defines the size in words of the image following word  $\emptyset$ .

Word 1-N The variable length image.

Word N Same as word  $\emptyset$ .

## 9.1.8 Basic Functions of Loader Phase 2

### A. Inputs

1. Phase 1 Tables and Lists (ATOC, SNT, ENT, SMAP, EQUT, EXCT, ED, collected external definitions for included elements and Job library).
2. Entry Point Lists From Group and System libraries.
3. Lists within included elements (XREFS, CC, INFO, SDEF).

### B. Output

1. ATOC (Updated from Phase 1, with additions and changes)
2. CC (List of control counters with assigned bases for all elements in the collection)
3. SXREF (List of all XREFS from elements along with assigned addresses)
4. INFOT (List of all common area definitions from all included elements along with assigned bases)
5. ALE (List of all cross segment references which will go into the vector table of the absolute element)
6. SDEF (Symbol definitions with address modified and placed on scratch file)
7. XREFs and CC (A modified list of satisfied XREFs and control counters broken down to 1 word each)

### C. Procedure

The items are extracted from ATOC and all external references satisfied by external definitions. If cross segment references are found an ALE item is produced. If the element referenced is not included it is added to the end of ATOC for later processing.

Common area definitions are summarized in INFOT. After all elements in ATOC have been processed, a pass is made on all control counters and satisfied XREFs assigning bases and addresses. The intermediate and output tables and lists are described below.

## Loader Phase 2 Tables and Descriptions

### Internal Tables

ATOC, SNT, ENT, SMAP, EQUT (From Phase 1)

### Intermediate and Outputs Tables

N	N	N	N	N
N	N	N	N	N
T	CC	VALUE		
7				7
TOC DRUM INC (IDENT)				

### EDEF TABLE

The EDEF table is a collection of all EDEFs from all elements within a particular library. EDEFs for use and system libraries are already grouped in an ENTRY POINT TABLE. EDEFs for the JOB are collected during phase 1 while the INCLUDE statements are being processed. If insufficient core storage, the EDEF lists are buffered in and out. The EDEF table is used to satisfy external references of the included elements.



ALE and INFOT FORMAT

N	N	N	N	N	} INFO ITEM
N	N	N	N	N	
1		GN	VALUE		}
N	N	N	N	N	} ALE ITEM
N	N	N	N	N	
		SEG #	ORDER		}

The ALE and INFOT table items are intermixed with the INFOT item having the sign bit of the third word set to distinguish it from an ALE entry. The value in the INFOT item represents the largest common area of the same name. Blank common is represented as having a GN of 1. Any blank common area occupies the same area regardless of name.

The ALE entry will eventually be used to produce the segment jump table. The third word defines the position within the segment jump table for that entry.

SXREF TABLE

	N	N	N	N	N
	N	N	N	N	N
	TU	CC		VALUE	
	ALE LINK			TOC LINK	
CC				VALUE	
	INFO LINK			VALUE	

The SXREF table is created upon the satisfaction of external reference by an external definition. T=1 if value is absolute, U equals 1 if XREF was unsatisfied. The ALE link is a link to an automatic load entry table item. It is present when the reference is such that a segment load is necessary before the transfer can be completed during execution. The TOC link is a link to the ATOC table item of the EDEF satisfying that external reference.

The CCs associated with the element are also processed to some extent. A control counter not associated with a common area is simply moved without charge. A control counter associated with a common area has a link to the INFO item defining this common area.

After each element is processed in phase two the above information is buffered out to mass storage and the TOC (ATOC) table updated with the necessary information.

EXCT

N	N	N	N	N
N	N	N	N	N
V	V	V	V	V
SEG #				

EXCT is built during Phase 1 from names on exclude cards. It is preserved through Phase 2 where the EXCT table is checked before a toc is added to the ATOC table. It indicates what elements are to be excluded from the collection.

A

EQU

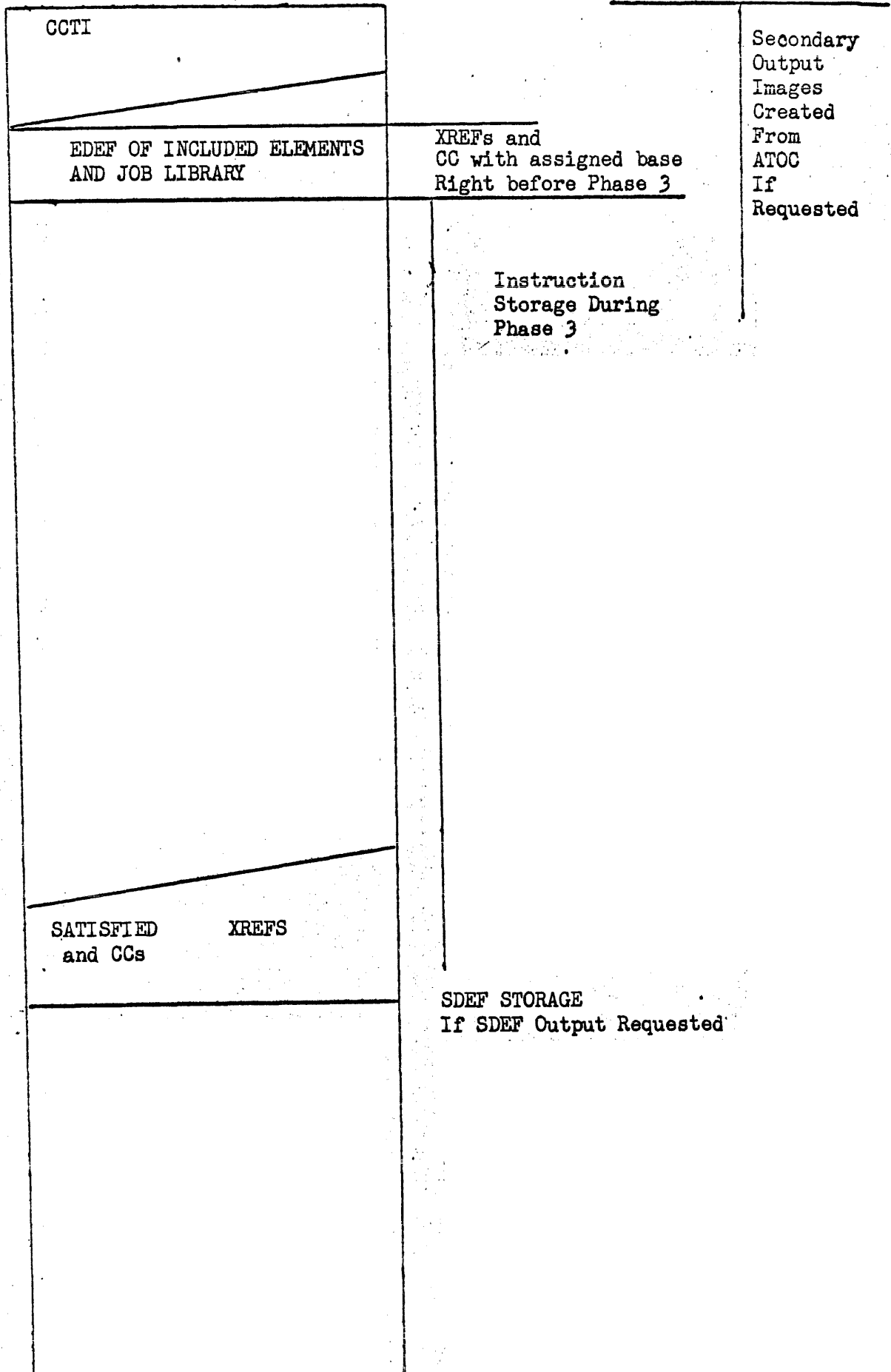
N	N	N	N	N
N	N	N	N	N
NE	NE	NE	NE	NE
NE	NE	NE	NE	NE
0				
4	SEG #		INC	

B

N	N	N	N	N
N	N	N	N	N
O	←			O
ALE	LINK	TOC	LINK	
TV	CC	VALUE		

EQU is built during Phase 1 from information on equals cards. During Phase 2 the reference (NE) is satisfied and the item is modified to appear as in B. This table is then used to satisfy an XREF that cannot be satisfied by an EDEF.

DRUM LAYOUT



### 9.1.9 Basic Functions of Loader Phase 3

#### A. Inputs

1. Phase 1 and 2 Table, Lists (ATOC, XREF AND CC LIST, SNT, INFOT, ALE);
2. TEXT form included elements.

#### B. Outputs

Absolute element on scratch file.

#### C. Procedures

All items are extracted from ATOC in reverse order, (control to last segment), and the text, modified XREFs and CC's for each element are read into core. The text is modified as indicated by the modification and buffered out to the scratch file area reserved for the element.

		A					B					C										
ATOC		SEGMENT #					ORDER COUNT						SEGMENT #					COUNT				
		N	N	N	N	N	N	N	N	N	N		N	N	N	N	N	N	N	N	N	N
		N	N	N	N	N	N	N	N	N	N		N	N	N	N	N	N	N	N	N	N
		V	V	V	V	V	V	V	V	V	V		V	V	V	V	V	V	V	V	V	V
		INCREMENT TO ELT					INCREMENT TO ELT BASE					0	INCREMENT TO ELT BASE									
		INDEX TO XREF			INDEX TO CC		INC TO CC			INC TO END		1	N	N	N	N	N	# XREF			CORE BASE	
		INDEX TO INFO			INDEX TO SDEF		DEC FROM EQTB			INDEX TO SDEF		2	N	N	N	N	N	DEC FROM EQTB			INDEX TO SDEF	
		INDEX TO TEXT					BASE					3	V	V	V	V	V	BASE				
		LENGTH OF TEXT					INDEX TO TEXT					4	INDEX TO TEXT									
		IDENTIFIER (TOC DRUM INC)					LENGTH OF TEXT					5	LENGTH OF TEXT									
		IDENTIFIER					(TOC DRUM INC)					6	# CC					CC BASE				
		IDENTIFIER					(TOC DRUM INC)					7	INDEX TO TEXT									
		IDENTIFIER					(TOC DRUM INC)					10	LENGTH OF TEXT									
		IDENTIFIER					(TOC DRUM INC)					11	# CC					CC BASE				

(A) ATOC is originally produced from the system tocs that correspond to names on include cards. An identifier is tacked onto the end which is used to identify the element from which an EDEF was obtained. If more elements are retrieved to satisfy XREFs the tocs for those elements will be added to ATOC.

After Phase 1, as the XREFs are processed in Phase 2, the ATOC item is modified as in B. The satisfied XREFs and CCs are buffered out to the drum and information necessary to retrieve them is supplied. The information is relative to a file increment known within the loader.

Upon entrance to Phase 3 the ATOC item is in the form defined by C where word 5 has been modified to relate the number of XREFs (1 word) and their location. Word 11 relates the number of CC and their location.

	A (Phase 1)					B (Phase 2)					C (Phase 3)				
SNT	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
						# ENTRY POINTS					BASE		LENGTH		

SNT is initially built during Phase 1 and goes through a series of changes until it eventually obtains the final form described in C, where (length) defines the length of the particular segment and (BASE) defines the relative core base at which segment should be loaded.



ABRIDGED SXREF AND CC LIST

T	U	VALUE
		BASE
INFO	LINK	LENGTH

XREF  
 CC (TEXT)  
 CC (COMMON)

All XREFs for each element are assigned the actual address and are broken down to 1 word. The status bit setting in the upper and the value in the lower. All control counters for text have the assigned base in the lower half word and the upper half is clear. Common area control counters have a link in the upper to the INFOT item where the common area length may be found.

ABSOLUTE ELEMENT FORMAT

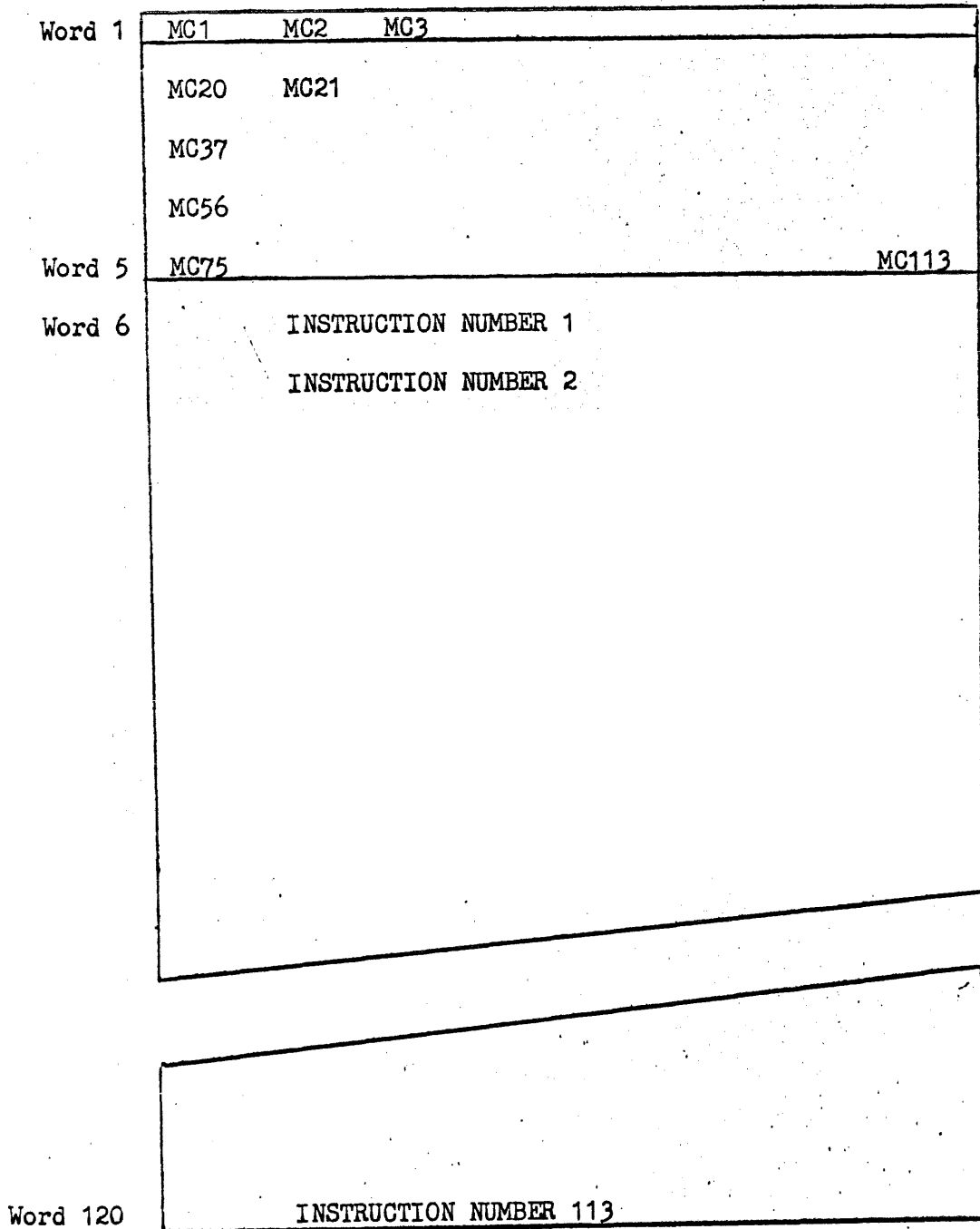
ASSIGN IMAGES		1			
ENTRY INSTRUCTIONS		2			
FC	LENGTH OF SEG	3			
REL CORE BASE OF SEG					
DRUM INC TO SEG					
)					
7 7 7 7 7 7 7 7 7 7	ENTRY ADDR	4			
7 7 7 7 7	OF 7 7 7 7 7	5			
ENTRY ADDR		6			
)					
7 7 7 7 7 7 7 7 7 7		7			
COMMON AND CONTROL		8			
SEGMENTS		9			
N	N	N	N	N	10
N	N	N	N	N	
SEG NO.	VALUE				
)					
9.1-41					

## ABSOLUTE ELEMENT FORMAT DESCRIPTION

1. ASSIGN IMAGES - Images used at load time for facility assignment.
2. ENTRY - The initial word of the element that will be placed in core at load time. The start address of any element will be the base of the element. If the starting point is in a subsegment the entry instruction will be a jump to that entry point in the segment jump table.
3. SEGMENT LOAD PACKET - A segment load packet defining the file, length, base, and increment to the particular segment. There is one, three word packet for each segment in the collected element.
4. 7777777777 - A sentinel to signal the end of the segment load packets.
5. 77777 X - A sentinel signaling the start of the segment jump table. The lower contains a link to the segment load packet for that segment containing the entry points.
6. ENTRY ADDR.- The upper of this word will always contain the entry point address. If that segment is presently in core the lower will contain the same address. If that segment is not in core the lower will contain 77777 which will cause a guard mode interrupt when an attempt to reference it is made. Upon a guard mode interrupt control will be given to the segment loader and the segment referenced will be brought into core. The jump table will then be updated by the segment loader in such a way that any segment destroyed by the new segment will have its entrance address set to 77777.
7. 7\_\_7 - A sentinel indicating the end of the jump table.
8. COMMON AND CONTROL - The part of the element that is resident in core for the life of the element. It contains the control portion plus all common areas defined, plus any routines collected to satisfy undefined XREFs.
9. SEGMENTS - Segment storage
10. SDEF - Symbol definition list with assigned values.

# INSTRUCTION MODULE FORMAT for 490 COLLECTOR OUTPUT

The instruction modules that the loader produces consist of five modification words and 113 instructions. The modification appears, left adjusted, in the first five words of the instruction module. Each of the 113 instructions require two bits of modification code, MC. Instructions start in word six of the module.



## 9.2 Library Maintenance Functional Description

Library maintenance consists of four basic operations; IN, OUT, PRINT, and LINK. IN's function is placing an element onto the Job Library. The operations are initiated by a control statement. Upon receiving a job number and a control image, IN scans and extracts the different fields. The option field is master bitted and then used as a switch. The Name/Version fields are extracted according to valid control characters and checked for their length. After extraction the job directory entries (p. ) are accessed from the job library. The previous job numbers in the directory are compared with the current number. A successful comparison enables the linking to any prior TOC modules. The TOC (Table of Contents) contains a residing element's name/version, logical drum address, and indexes. The current name/version from the statement is cross referenced against the TOC. With a mismatch, IN starts buffering in a new element from either the primary or secondary source. As the IN function continues, there is a buffering out of the element onto mass storage. The storage has been assigned to the element through the use of the master storage directory (p. ). The incoming TOC is updated by entering a logical drum address. The TOC and its element are now linked together. The new TOC is then moved to a new or prior TOC module. The final operation in placing an element onto the library is updating the job directory and TOC module. With each successive element, the pre-described events are repeated. Upon exhaustion of either the primary or secondary input streams, the function returns control back to the system.

The OUT function is initiated in the same manner as IN. OUT scans and extracts the different fields from the control image. One mandatory field is the library name (JOB, GROUP/LIB # or SYS.). Without this field, the function would be unable to determine in which library a specific element resides. Also, the field is necessary for the interpretation of the specified format of each library (p. ). TOC modules are accessed from one of these libraries. Comparison of name/version from the statement with the related TOC establishes which element is to be processed.

From analyzing the option field, a switch is set determining whether a secondary or primary output is required. With the type of output and TOC known, the function chooses one of three avenues in the processing of a specific element type. The avenues are associated with the element type: relative binary, absolute or source (p. ). If the switch is set for primary output, the function transfers each element in 20<sub>8</sub> word card images. With the exception of source, two

words of any 20<sub>8</sub> word image will have a sequence name and number (p. ). For tape output OUT sets up a header, data, and end sentinel block (p. ). The header block contains the TOC. The remainder of the element resides in the data blocks. Each data block contains specific number of 16<sub>8</sub> word images. Images will not have a sequence name and number. Each element established on a tape contains these basic blocks.

An element's TOC and preamble contains vital information. PRINT's task is to submit that information to primary output printer. PRINT has OUT's method of retrieving a specific element from a library. With the retrieved information PRINT checks for a (T) option. If present the function prints the TOC. Otherwise, the TOC and preamble are both printed. The next logical event to occur is determining a element format. Once established, the TOC and preamble are printed in the preceding forms.

#### RELATIVE BINARY ELEMENT

#### TABLE OF CONTENTS

	BINARY	ELEMENT	NAME-LMO		VERSION-ONE	
	EDEF	XREF	CC	INFO	SDEF	TEXT
INDEX	00000	00003	00056	00063	00077	00077
NUMBER	00000	00027	00005	00004	00000	01526

The preamble of this binary element consists of entry definitions, external references, control counter and symbol definitions. The entry definition (EDEF) is a point within an element referenced by other elements. The EDEF has 1 to 10 character left justified alphanumeric name and has a value associated to the name. In the below representation, the CC is the number of a control counter which contains a base increment. The base increment plus "VALUE" of the EDEF is the entry or referenced point. If there are three or more EDEF the form is repeated as indicated.

#### ENTRY DEFINITIONS

NAME	CC	VALUE	NAME	CC	VALUE	NAME	CC	VALUE
GJD	00000	00101	SCO	00002	00010	SCOP	00000	00031
MAD	00004	00765	...					

The preamble contains a list of labels or tags not referenced in the element (XREF). At collection time the external references are satisfied by including the necessary referenced element. The name has 1 to 10 alphanumeric characters and is left justified. The NR (number) is implied by position of entry in the list.

EXTERNAL REFERENCES

NR	NAME	NR	NAME	NR	NAME	NR	NAME
00000		00001	SPICK	00002	IDUS	00003	KROK

The control counter specifies the number of consecutive words of core required by code operating under control of this counter. At load time the counters have affixed starting address of assigned relative area. The NR (number) is implied by a counters position.

CONTROL COUNTERS

NR	VALUE	NR	VALUE	NR	VALUE	NR	VALUE
00000	00100	00001	00110	00002	00200	00003	00210
00004	00240						

Areas common between different compiled elements are described by information entries. The INFO has 1 to 10 character alphanumeric name. The name is left justified. The CC is a number of a counter giving its size.

INFORMATION ENTRIES

NAME	CC	NAME	CC	NAME	CC	NAME	CC
MOCK	00000	AKKE	00001	SOSE	00002	ZIDE	00003
POSH	00004	LOCA	00005	...			

Symbol definition entry (SDEF) is a label or tag used within an element for testing purposes. The base value represented by the CC is modified by a relative increment (VALUE) at load time. The obtain relative address is an entry or referenced point used for testing. The name is 1 to 10 character alphanumeric and is left justified.

ABSOLUTE ELEMENT

TABLE OF CONTENTS

ABSOLUTE ELEMENT	NAME-LM01	VERSION-ONE
MAXIMUM CORE USED-		00200
NUMBER OF ASG IMAGES-		00004
NUMBER OF SDEF-		00016

ASG IMAGES

△ ASG △ H △ TAPE,A, ,WORK UNIT

. . .  
ASG images are used at load time for facility assignment.  
SDEFs are described p ( ).

SOURCE ELEMENT

TABLE OF CONTENTS

SOURCE ELEMENT	NAME-LM01	VERSION-TWO
NUMBER OF IMAGES IN ELEMENT-		00102

SOURCE ELEMENT

00000 LMO PROGRAM  
00001 CD COMMON\*100



## IN OPERATION

### A. Format of Statement

# IN OPTIONS FC, NAME/VERSION, ETC.

### B. Valid Options

C - Elements follow in primary input streams

R - Rewind tape before processing

X - Abort job if errors occur

Y - Continue even if a non-fatal error occurs

### C. Element Specifications

If no NAME/VERSION is specified all elements contained in the medium specified will be included. The order in which the specified element will be processed is dependent on the order in which the elements are inputted and not on the order of specification on the IN image.

FC - File code is present only when no C option

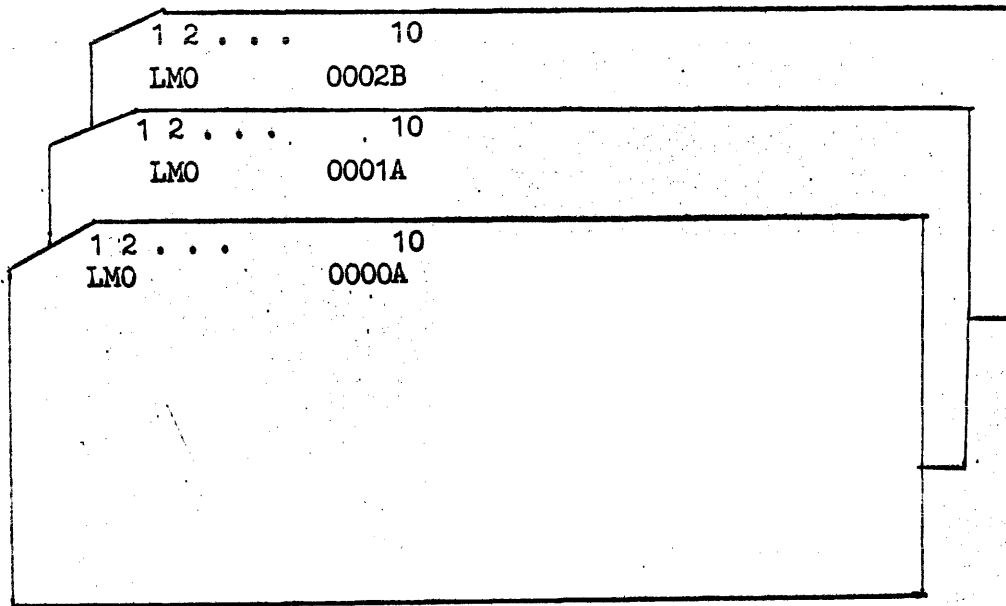
### D. Functional Description

1. The initial duty of Library Maintenance upon receipt of an IN statement will be to attempt to locate the job within the Job Directory maintained as part of the job library. If the job is not already established in the directory an entry is made.
2. The options are then master bitted and any element names unstrung.
3. The header of the elements are located and a branch is made on the type of element encountered. The proper format is then determined, a request for the amount of mass storage needed for the storage of the element is made, and the element is then buffered out to the job library.

## CARD IMAGES

### 1) Relative Binary Element

Col. 1 to Col. 10 - Sequence number and card type



A) Card Type (A) consist of a TOC and preamble

B) Card Type (B) is Test

2) Absolute Element - In addition to (A) and (B) type the absolute element has a third type, (C). The (C) type image contains symbol definitions.

3) Source Element - Source has no sequence number or card type.

## OUT OPERATION

### A. Format of Statement

# Out options FC, JOB, NAME/VERSION, ETC.

# Out options FC, SYS, NAME/VERSION, ETC.

# Out options FC, GROUP/LIB#, NAME/VERSION, ETC.

### B. Valid Options

C - Elements to follow primary input stream

F - Write hardware end of file on tape output

X - Abort job if errors occur

Y - Continue if non-fatal error occurs

### C. Element Specification

Elements will be outputted in the order in which they are located in the library not in the order in which they were specified on card. If an element NAME/VERSION is blank, all elements within the specified library will be outputted.

FC - (file code) is present when a C option is absent.

JOB;SYS;GROUP/LIB# - is a fixed entry which defines the particular library from which the element(s) are to be taken from.

### D. Functional Description

1. The job is located within the job directory.
2. The TOCS are located for elements to be put out.
3. The data is formatted in the proper format.
4. The data is put out to the prescribed media.

Tape and Drum Formats

0	7 3 7 3 7 3 7 3 7 3
1	ELEMENT NAME
2	AND
3	VERSION
4	DATE
5	REEL #
6	TIME
7	BLOCK SIZE
10	ITEM SIZE
0	TOC
1	
2	
3	
4	
5	
6	
7	
10	
30 <sub>8</sub>	7 3 7 3 7 3 7 3 7 3

The initial block of each element contains the library number assigned to the element. The block size and item size will normally be 306 and 14 respectively. Each item will normally be a 14 word card image.

Tape and drum formats are identical. If all tape and drum processing is handled as drum processing, the processing will be identical.

## Data Block Layout for Elements

# ITEM	BLOCK SIZE
CXSUM	

The lower of the first word contains the actual number of data words present in the block. The upper word has number of equal sized items contained in the data block. If upper half is negative, there is no logical item size. The last word of the data block is the CXSUM.

On tape and drum, any desk ID, sequence number and type are dropped and only 14 word images are used except in the case of source code. There is no separation of the individual parts of the elements as the length of each part can be calculated from the toc.

## PRINT OPERATION

### A. Format of Statement

# PRINT options JOB, NAME/VERSION, ETC.

# PRINT options SYS, NAME/VERSION, ETC.

# PRINT options GROUP/LIB#, NAME/VERSION, ETC.

### B. Valid Options

T - list TOCS only

O - Print card images contained in named files normally used for control streams source code.

X - Same as #IN

Y - Same as #IN

### C. Element Specification

Elements will be outputted in the order in which they are located in the library not in the order in which they were specified on card. If an element NAME/VERSION is blank, all elements within the specified library will be outputted.

FC - (file code) is present when a C option is absent.

JOB;SYS; GROUP/LIB# - is a fixed entry which defines the particular library from which the element(s) are to be taken from.

### D. Functional Description

1. The job is located within the job directory.
2. The TOCS are located for elements to be put out.
3. The data is formatted in the proper format.
4. The data is put out to the prescribed media.

## LINK OPERATION

### A. Format of Statement

# LINK options LIB#, FC, Ptype

### B. Valid Options

Same as on an #ASG card

### C. Library Specification

LIB# is the file name of the desired group library

### D. Functions Description

Library Maintenance will request a file check with MFDR using the name/version of the user library. If already present, an indication is given and the link increment returned. If not present, Library Maintenance will perform a service request for the assignment of media which the requested library resides. The initial block of the library is read and a registration request for file extensions is made along with the amount of mass storage required. The elements are then processed and place on mass storage.





The Job Directory

The initial job directory occupies logical addresses 102-203 of the job library file. It serves the dual purpose of linking job numbers to their associated elements and of linking group libraries with individual jobs. The job entry is one word in length. The last word of the module contains the remaining number of cells within the module and a link address to the next module. The library link entry has the sign bit set to distinguish it from directory entries. Upon a reference to a group library, a search is made of the user library links and entries with corresponding job numbers will apply. The group library number is an identifier to distinguish between group libraries.

0	JOB 1	INC TO LAST TOCM
	JOB 2	INC TO LAST TOCM
	JOB 3	INC TO GROUP LIB.
	GROUP LIBRARY NUMBER	
101	# ENTRY	Ø OR LINK NEXT MOD.

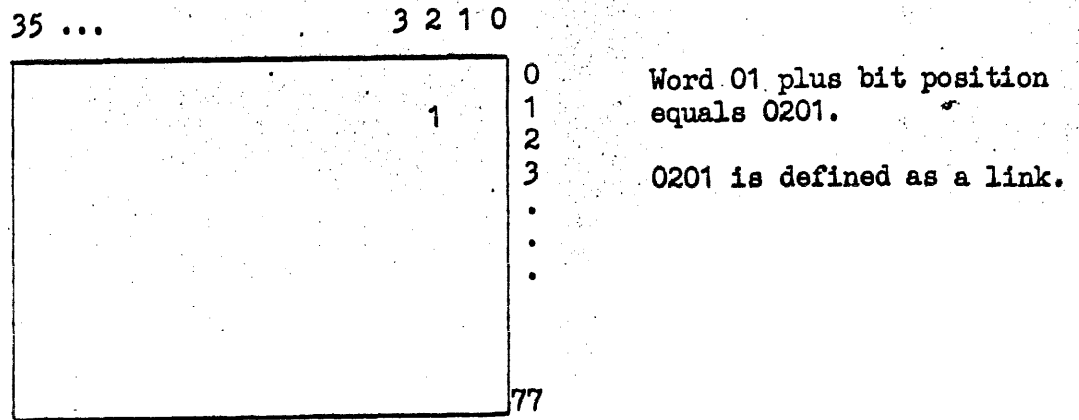
## LIBRARY MAINTENANCE STORAGE DESCRIPTION

A basic area of prime mass storage is used to pool data involved with a number of jobs. Any extension required due to the overflow of the basic area will be restricted to the job associated with the request causing the overflow. Any group library established will cause the extension of the job library file. If possible the extension for a group library will be made on non-prime mass storage. Upon the termination of a job, the area within the basic mass storage will be released to library maintenance. Any extensions associated with the job will be released to the system. A group library extension will remain accessible as long as an active job is associated with it. A non-active group library area will be released to the system if the area is needed.

The extension of the job library are made through a Master File Directory Routine (MFDR). The MFDR will maintain a directory with an entry for each file along with logical and absolute addresses of the file. A count of the number of jobs using the group library extension is maintained.

## MASTER STORAGE DIRECTORY OR BIT MAP

The bit map occupies the initial 102 words of the job library. The first 100 words are taken as a 100 by 36 grid of lists. Each bit within this grid symbolizes a 102 word mass storage area. The file increment of any 102 word module within the basic job library can be determined by the position of its representative bit in the map. The file increment is equal to the horizontal position, which forms the upper two digits of the bit map increment, plus the vertical position, which represents the last two digits of the bit map increment, times the module size (102).



The list in the example above would represent a 102 word module at file increment  $0201 \times 102 = 20501$ .

As the storage is used the bits representing the area used are set in the bit map. To find a hole for storage the bit map is checked vertically from right to left.

A group library is not placed in the basic job library, but an extension to the job library is requested to hold the group library. If there is insufficient storage in the basic job library for the storage of an element an extension will be created similar to that created for a group library.

The bit map is read with lock to prevent the manipulation of the bit map by different routines at the same time.

Upon termination all element storage associated with the job will be released.

The routines used to manipulate the bit map in library maintenance are EM, which is used to request storage and EMD which is used to release storage to the system.

GROUP AND SYSTEM LIBRARY FORMAT WITHIN LIBRARY COMPLEX

LIBRARY NUMBER	
INCREMENT TO TOC	
INC. TO EP TBL	
LENGTH TOTAL LIB	
LENGTH ABS TOC	LENGTH RB TOC
LENGTH OF SOURCE TOC	LENGTH OF Edef.
<p style="text-align: center;">             L              I              B              R              A              R              Y                E              L              E              M              E              N              T              S           </p>	
ENTRY POINTS FOR RB ELEMENTS	
T	ABS TOCS
O	RBS TOCS
C	SOURCE TOCS

Entry Point Table for Group and System Library

7	7	7	7	7	TOC NUMBER
N	N	N	N	N	
N	N	N	N	N	
T	CC	VALUE			

ELEMENT EDEFS

For the group or system library all EDEFS are collected from the individual relative binary elements. Each EDEF is logical placed in the table in a direct relationship to its associated TOC. The TOCS are stored in descending order. Therefore, the last entry in the EDEF table has a TOC NUMBER equal to zero. Upon use, the collector loader is enabled to correlate EDEF and TOC with a degree of efficiency.

Representation Of A TOC Module And Individual Internal TOC

	# ENTRIES	Ø OR LINK NEXT MOD.			
Ø	(SOURCE ELT.) 3	ERROR IND.			
1	N	N	N	N	N
2	N	N	N	N	N
3	V	V	V	V	V
4	INCREMENT TO ELT. BASE				
5	UNASSIGNED				
6	UNASSIGNED				
7	(SUB.CODE) B	# IMAGES IN ELT.			
10	TOTAL ELEMENT LENGTH				
0	(ABSOLUTE ELT.) 2	ERROR IND.			
1	N	N	N	N	N
2	N	N	N	N	N
3	V	V	V	V	V
4	INCREMENT TO ELT. BASE				
5	MAX. CORE USED	LENGTH OF CONTROL			
6	# SEGMENTS	# ASG IMAGES			
7		# SDEF			
10		INDEX TO SDEF			
0	(REL. BINARY) 1	ERROR IND.			
1	N	N	N	N	N
2	N	N	N	N	N
3	V	V	V	V	V
4	INCREMENT TO ELT. BASE				
5	INDEX TO XREF	INDEX TO CC			
6	INDEX TO INFO	INDEX TO SDEF			
7		INDEX TO TEXT			
10	LENGTH OF TEXT				

## DESCRIPTION OF TOG CONTENTS

### 1. RB ELEMENT

Word $\emptyset$	A number indicating an RB element. The lower of this word is used for error indication.
Word 1-3	A 10 character name and a 5 character version. All unused portions of these 3 words are space (05) filled.
Word 4	An increment from the base of the job library at which the element can be found.
Word 5	U = an index from the increment in word 4 which relates the start of the XREFS and also relates the length of the EDEFS L = index to start of the cc and relates the length of the XREFS
Word 6	U = index to start of the INFO and the length of the cc L = index to start of the SDEF and the length of the INFO
Word 7	L = index to start of the text and the length of the SDEF
Word 10	Length of the text

### 2. ABSOLUTE ELEMENT

Word $\emptyset$	U = Absolute type number (2) L = Error indicator
Word 1-3	Same as in RB element
Word 4	Same as in RB element
Word 5	U = the maximum amount of core utilized at any one time L = spare
Word 6	L = number of ASG images
Word 7	L = the number of SDEFS
Word 10	Index to SDEFS

### 3. SOURCE ELEMENT

Word $\emptyset$	U = source type number (3) L = Error indicator
Word 1-3	Same as in RB element
Word 4	Same as in RB element

Word 5 Unassigned

Word 6 Unassigned

Word 7 U = subtype (B) indicating an internally compressed source while subtype (A) indicates normal source from primary input stream.  
L = the number of images of compressed source type (B)  
The total length of type (B), compressed source, or the total number of type (A) non-compressed source in field data code.



Source Element

INDEX TO NEXT SOURCE IMAGE
INDEX TO NEXT SOURCE IMAGE

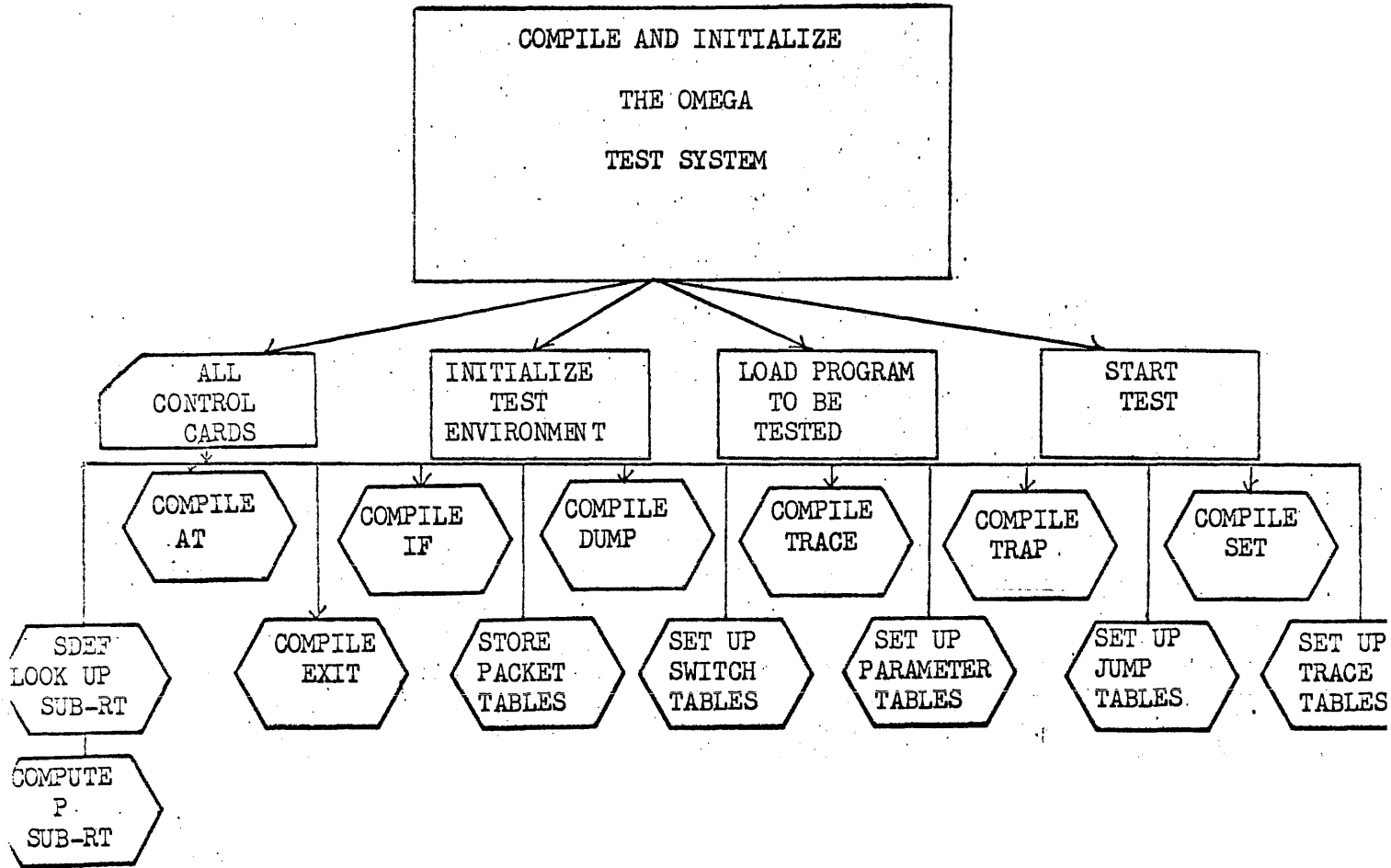
A source image is received from the primary input stream with trailing spaces removed. Hence, the term (compressed source) is defined. As each source image is encountered, an index to the next image is attached. The index and the compressed image is buffered onto the job library. With each successive compression occurring the Job Library saves prime storage.

# Library Tape Format

0	7 2 7 2 7 2 7 2 7 2
1	7 2 7 2 7 2 7 2 7 2
2	FILE
3	
4	IDENTIFIER
5	DATE
6	
7	TIME
10	
11	# ELEMENTS
12	# RB ELEMENTS
13	# ABS ELT
14	# SOURCE ELT
15	DRUM REQUIREMENT FOR STORAGE
	7 2 7 2 7 2 7 2 7 2
30 <sub>8</sub>	7 2 7 2 7 2 7 2 7 2

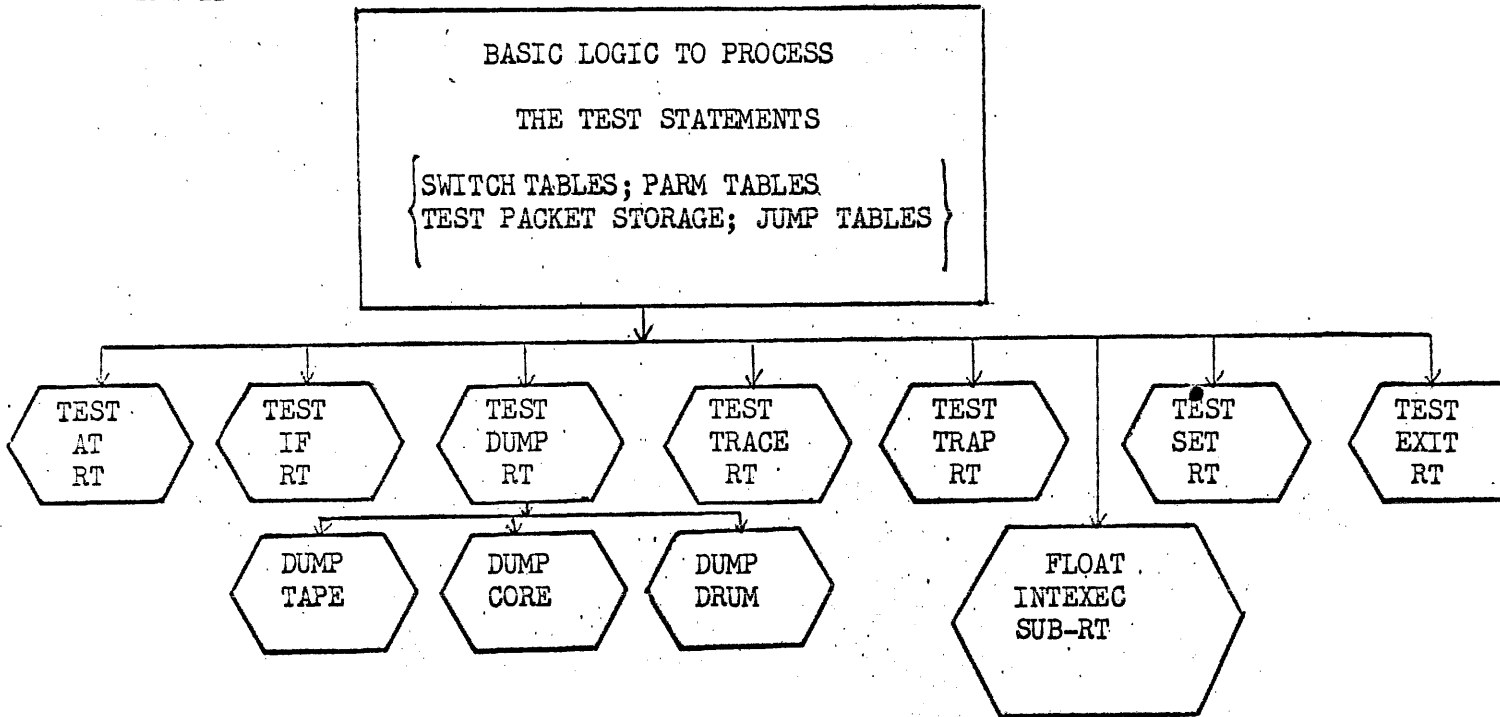
File Header - This record will be the initial block on a library output tape. A rewrite of this block is performed by ELM in order to insert the drum requirements of the library and the number of elements within the library. The header size is 30<sub>8</sub> words.

SEG I

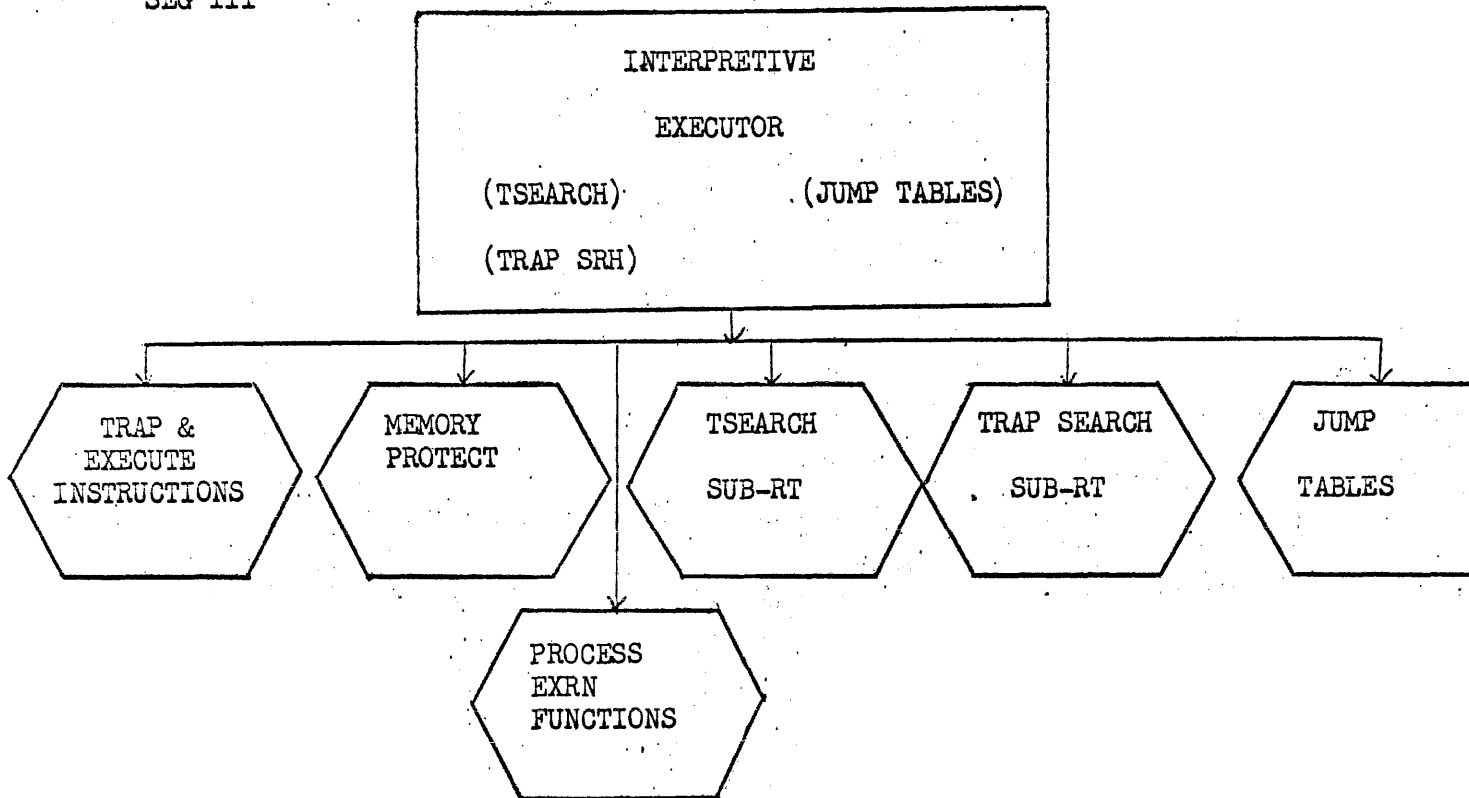


TEST SYSTEM

SEG II



SEG III



OPERATING TEST SYSTEM IN CORE

SEG II

BASIC LOGIC TO PROCESS  
TEST STATEMENTS

SWITCH TABLES: PARM TABLES

TEST PACKETS STORAGE

RIR = 00000 PLR - LOWER

PROGRAM.

UNDER TEST

RIR = 00000

SEG III

INTERPRETIVE  
EXECUTOR

TSEARCH: DUMP;SET;AT;IF;EXIT;TRACE\*

TRAPSRH; TRAP\*

PLR - UPPER

\*NOTE: Whenever a hit is made control is given to SEG. II for processing.

## Phase I Test Package COMPILE

The Compile phase of the test Package is called into memory by means of the standard job card. The initialization phase consists of the following:

- (a) Set all B Registers to zero
- (b) Clear work areas
- (c) Get necessary core set RIR and PLR
- (d) Locate SDEF tables of program to be placed under test.
- (e) Load SDEF and Vector tables of the program under test.
- (f) Set all control switches to off.

### Card Read Section

The cards are read and processed by this section and all follow the same chain of logic:

- (a) The first 5 field data characters are used and a search is made of a table of all valid secondary control statements. If no hit is found the card is rejected and printed as an error card. Control is then returned to the card read segment and step (a) is repeated. Note error cards may not stop the test package. The test package will continue until it reaches an end condition.
- (b) Whenever a hit is found the card image is first submitted to the unstringer and then control is transferred to the proper subroutine to process that statement. All statements are processed by their own separate subroutine.

Search Table for all Valid Secondary Control Statements

- |    |                  |
|----|------------------|
| 1. | -AT--<br>--P-REG |
| 2. | -IF--<br>--P-REG |
| 3. | -DUMP<br>--P-REG |
| 4. | -TRAC<br>E-P-REG |
| 5. | -TRAP<br>--P-REG |
| 6. | -SET-<br>--P-REG |
| 7. | -EXIT<br>--P-REG |
| 8. | -END-<br>--P-REG |



COMPILE-AT-

Example: -AT--P/A,V<sub>0</sub>/V<sub>1</sub>,B

1. Since the card is already unstrung P is computed by a return jump to the Compute P subroutine.
2. Save Segment Number and P value
3. Test to see if switch A is given. If yes. Then set switch off and store location in packet. If no. Then set location for switch #27\*.
4. Set counter in packet equal to zero.
5. Convert V<sub>0</sub> to Binary and store in packet. If V<sub>0</sub> is equal to spaces set V<sub>0</sub>=1 and V<sub>1</sub>=7777 and skip to (7)
6. Convert V<sub>1</sub> to Binary and store in packet if spaces set to all seven.
7. Test for switch B. If given set up logic and store in packet. If it is not given then set for switch #27\*.
8. Increment B1, B2 and exit.

Packet

SWITCHB	SWITCHA
V0	V1
Counter	SEG #

\*Note. Switch #27 is a dummy switch that is always turned on.

Compile IF

Example: -IF--P/A,  $\underbrace{(\quad) \text{ // } (\quad)}_{\text{Relation}} * \text{ or } * \# \text{ next expression}$   
and  
 $\underbrace{\quad}_{\text{operator}}$  continuation symbol  
 $\underbrace{\quad}_{\text{connector}}$

1. The P value is gotten by a Return Jump to the Compute P subroutine and along with the segment # is stored in the Packet.
2. Test for switch one. If it is spaces set it equal to switch #27. If a switch is specified set up the proper logic and store the switch in the packet.
3. Compute the relational expression in the following manner:
  - a. Get the K value and save, if no K was specified then the value within the open and closed parenthesis is assumed to be an arithmetic constant and is converted to binary if necessary and stored at TIFPKT40 / B4 for future reference. This address is then submitted for the A operand. Go to (d).
  - b. If K was found then the value within the parenthesis is assumed to be an address and is computed by going to the compute P subroutine.
  - c. The A operand address complete with designator is saved.
  - d. The Test portion of the expression is saved for later reference.
  - e. Steps (a) through (b) are repeated to get the next portion of the expression and it is treated as the B operand.
  - f. The operator is tested for AND, OR. The routine branches on the AND or OR.
  - g. Test for B-A condition. If present then switch operands (B for A).
  - h. Branch on the test (EQ, GE, etc.)
  - i. Get K and Y operand for the ENTER A instruction. ENT\*A\*A address.  
Get K and Y operand for subtract A instruction.  
Sub\*A\*B Address with proper J designator.

example: ENT\*A\* operand

SUB\*A\*B operand \*J

JP\* continue

Note: The jump is set to accept if the operator was an OR, and to reject if the operator was an AND. The J is set to skip on a miss for an OR operator and to skip on a Hit for the AND operator. The end results is that the same 3 instructions are used for all relational expressions.

- j. A check is made after each operator for a continuation card. This is the only place where a request for a continuation card can be made. Any number of continuation cards may be given.
- k. Repeat steps (a) through (j) as many times as there are relational expressions.
- l. When the comma is detected after an operator the IF statement is closed out. The switch is set up to be turned on or off depending on the acceptance or rejection of the IF statement.

Sample IF statement:

```
-IF--START /A,L (SUMX/5) /EQ/ L(YSUM) *OR*#  
W(XLIST) /LT/ (666) *OR*L (DOG) /GT/ (60) *AND*,B
```

The total Packed compiled for this would be:

The table is to be  
tested  
When P=START  
of the operating program

TABLE

SEG#	TSWITCH	= A 00
TSWITCH	= B 01	
ENT*A*L (SUMX / 5)		
SUB*A*L (YSUM) *ANOT		OR Test 1
JP*TABLE / 011		
ENT*A*W (XLIST)		
SUB*A*W (TIFPKT40 / B4) *APOS		OR Test 2
JP*TABLE / 011		
JP* <u>TREJECT</u>		
ENT*A*W (TIFPKT40 / B4)		
SUB*A*L (DOG) *ANEG		AND Test 3
JP* <u>TREJECT</u>		
JP*TACCEPT		

These instructions would be generated and stored in the test bed. Each time the P value was reached in the program under test, and the switch A is on, then control would be given to TABLE 2.

AND 1. EQ  $A=B$  ; J = 4 ; AZERO ; A-B  
2. GT  $A>B$  ; J = 7 ; ANEG ; B-A  
3. LT  $A<B$  ; J = 7 ; ANEG ; A-B  
4. NE  $A\neq B$  ; J = 5 ; ANOT ; A-B  
5. GE  $A\geq B$  ; J = 6 ; APOS ; A-B  
6. LE  $A\leq B$  ; J = 6 ; APOS ; B-A

OR 1. EQ  $A=B$  ; J = 5 ; ANOT ; A-B  
2. GT  $A>B$  ; J = 6 ; APOS ; B-A  
3. LT  $A<B$  ; J = 6 ; APOS ; A-B  
4. NE  $A\neq B$  ; J = 4 ; AZERO ; A-B  
5. GE  $A\geq B$  ; J = 7 ; ANEG ; A-B  
6. LE  $A\leq B$  ; J = 7 ; ANEG ; B-A

Restrictions: (a) There cannot be more than 20(g) consecutive OR operators in any one IF statement.

(b) There cannot be more than 25(g) constants used in all the IF statements.

Compile the DUMP statement.

Example: -DUMP-OPTIONS-P/SWITCH, V<sub>0</sub>,V<sub>1</sub>,V<sub>2</sub>

1. P is computed by a return to the compute P subroutine and along with the Segment Number is stored in the DUMP Packet.
2. Set the options. If none are given set for octal.
3. The switch is tested. If spaces then the Dummy switch is set. If it is not spaces then the switch logic is set up.
4. Convert V<sub>0</sub> and store.
5. Convert V<sub>1</sub> and store.
6. Convert V<sub>2</sub> and store, if it is present. If V<sub>2</sub> is not present then it will be treated as Zeros.

DUMP PACKET

Base V <sub>0</sub> address	# of V <sub>1</sub> words
OPTION	File V <sub>2</sub> Code
SWITCH	SEG #

Compile TRACE statement

Example: -TRACE-OPTIONS-SWITCH,START,END

1. Test for and set switch. If the switch is left blank then set the Dummy switch.
2. The start location is gotten by doing a return jump to the Compute P subroutine. Both the P setting and Segment Number are saved.
3. The end location is gotten by a return jump to the Compute P subroutine. Both the P setting and the Segment Number are saved.
4. The Option is checked for a complete trace or a logical trace only. If a complete trace is requested then zeros are stored in the packet. If it is to be a logical trace only, then 5 sevens are stored in the packet.

TRACE Tab Packet

0	# Tests	} Packet 1
SEG #	SWITCH	
START	END	
77777	SEG #	
"	"	} Packet n
"	"	
"	"	
"	"	
SEG #	SWITCH	} Packet n
START	END	
0	SEG #	

### Compile the SET Statement

Example: -SET--P/SWITCH, Store Address, Value<sub>1</sub>/Value<sub>2</sub>,--,#

Store Address, \_/\_ , \_/\_ , \_/\_ ,

1. The P is gotten by a return jump to the Compute P subroutine. Store the Segment Number and the P setting.
2. Test and set switch. If spaces, set Dummy switch. If a switch is given then set up the proper logic.
3. The store address is gotten by doing a return jump to the Compute P subroutine. The Segment Number is compared to see if it is in the same segment as the P of step (1). If it is a different Segment Number then an error print occurs and the routine exits.
4. If valid the store address is saved in the Packet store area.
5. The first half of the value is gotten by a return jump to the ComputeP subroutine.
6. Step (5) is repeated for the second half. The two halves are combined and the 30 bit word is stored in the packet storage.
7. A check is made for a continuation card and if present the card is read and control is returned to step (3).
8. A check is made for a end of data symbol if present the packet is stored and the routine exits.
9. If neither (7) or (8) is true then Control is returned to step (5).

Set Packet

SWITCH	SEG #
P-Setting	# Words
-----	-----
-----	-----
P-Setting	# Words
-----	-----
-----	-----
77777	77777

Compile the TRAP Statement

Example: -TRAP-OPTIONS-P/SWITCH

1. P is gotten by a return jump to the Compute P subroutine. The P setting and the Segment Number are stored in the test package.
2. The switch is tested for and, if spaces then is set to the dummy switch, if given then the proper logic is set up.
3. The Option is stored if given. If no Options are given the octal conversion is assumed.

TRAP Packet

SEG #	OPTIONS
	SWITCH



Compile the EXIT Statement

Example: -EXIT--P/SWITCH

1. P is gotten by a return jump to the Compute P subroutine.
2. The switch is tested for, and if present then the proper logic is set, if it is omitted then the Dummy switch is set.
3. EXIT

Packet

SEG #	SWITCH
-------	--------

Test Package SEG II Test and Perform

Test-AT-Statement and Perform

1. Control comes from the TSEARCH ROUTINE and B7 holds the address of the Test Packet.
2. Test switch on:  
No: Exit  
Yes: Continue
3. Test if the Routine is in the correct segment  
No: Exit  
Yes: Continue
4. Perform AT Statement
  - a. Test  $V_1$  equal to 0. If Yes then EXIT
  - b. Increment counter and test if equal to  $V_0$ . If Yes. Turn the switch on clear the counter and subtract one from  $V_1$  then exit.

### Test-IF-Statement and Perform

1. Control comes from the TSEARCH Routine and B7 holds the address of the test packet.
2. Test if the switch conditional to the performance of the IF Statement is on. If it is off exit.
3. Test if this is the correct segment. If not then EXIT.
4. Go Perform the IF Statements logical tests. If the decision is to accept then turn the indicated switch on and EXIT. If the decision is to reject then turn the indicated switch off and EXIT.

## Test-DUMP-Statement and Perform

1. Control comes from the TSEARCH Routine and B7 holds the address of the test packet.
2. Test if the conditional switch is on. If off then EXIT.
3. Test if this is the correct segment. If it is not then EXIT.
4. Submit packet to the DUMP ROUTINE.
5. EXIT when control is returned.
6. Perform-DUMP
  - a. Clear the initial print line
  - b. Branch on the OPTION Code
    1. A OCTAL
    2. B Field Data
    3. C Binary to decimal
    4. D Double precision Floating point to its decimal representation.
7. OCTAL: Convert binary number to its field data equivalent and print the base address of the print line and 8 words.
8. Cycle on Step 7 until print packet is exhausted then EXIT.
9. FIELD DATA  
Move the data directly to the print line and print.
10. BINARY TO DECIMAL  
Convert the data to its decimal representation. A maximum of ten characters per 30 bit word.
11. Floating Point Descale Routine  
Convert the 60 bit word and shift it to the print line.

### Test-TRACE-and Perform

1. Control comes from the TSEARCH Routine and B7 holds the address of the Test Packet.
2. Test for the correct segment number range. If it is not then EXIT.
3. Test if this location is within the proper limits. If it is then go to step 5.
4. Continue search of TRACE TABLE by incrementing to the next test packet. If the last packet has been checked then EXIT. If it has not then go to step 2.
5. Test to see if this is a logical trace only. If it is then go to step 7.
6. Go Perform the printing trace and then EXIT.
7. Test if the instruction being interpretively executed is either a jump or a return jump. If it is go to step 5. If it is not then EXIT.

Note 1: If this is a complete printing trace option and the instruction being interpretively executed is the Repeat instruction then the instruction being repeated will be printed without the registers settings.

Note 2: Those registers remaining the same as in the previous print, either trace or trap, will not be printed again until their values change.

### Test-TRAP-and Perform

1. Control comes from the TRAPSRH Routine and B7 holds the address of the test packet.
2. The logical switch is tested. If it is off the Routine exits.
3. The segment number is tested. If this is the wrong segment then the routine exits.
4. Perform the TRAP Operation
  - a. Convert the address of the trapped location to its Octal representation.
  - b. Branch on the option
    - 1) Octal: Convert the 30 Bit word to its octal representation, and move it to the Trap buffer. Go to step 5).
    - 2) Field Data: Transfer the field data word to the Trap buffer and go to 5).
    - 3) Decimal: Convert word to its decimal equivalent and move to the Trap buffer. Go to step 5).
    - 4) Floating Point: Convert the 60 Bit value to its decimal equivalent and move to the Trap buffer.
    - 5) Print the Trap Buffer line
    - 6) RJP to the printing trace and print the instruction that is making the change, and the registers, the same as in the printing trace.
    - 7) EXIT

### Test-SET-and Perform

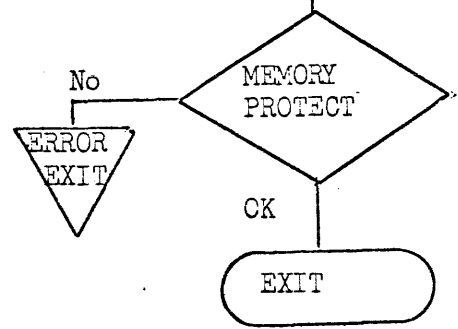
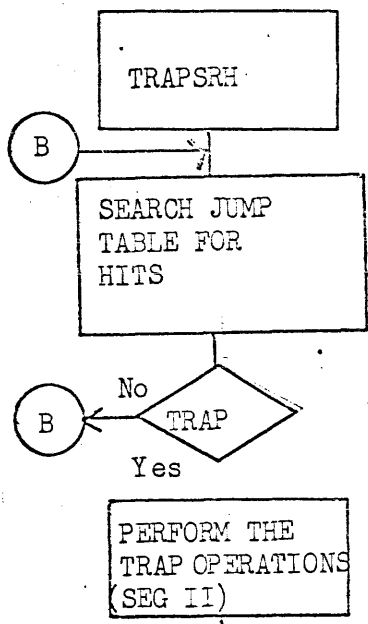
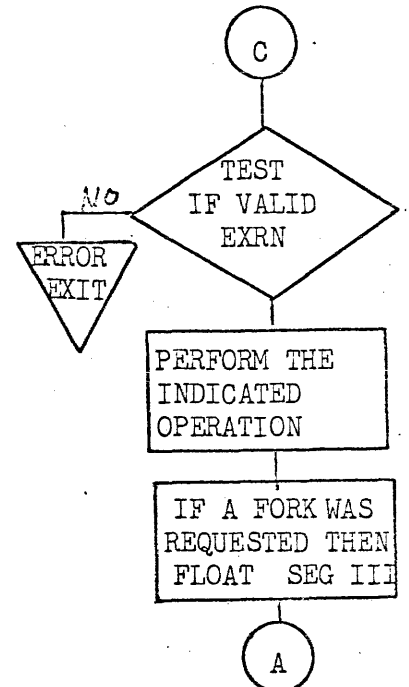
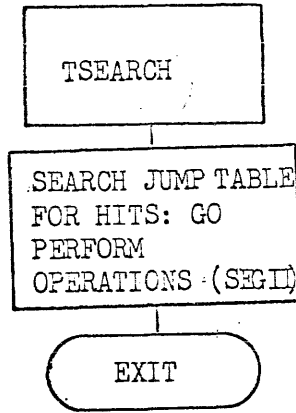
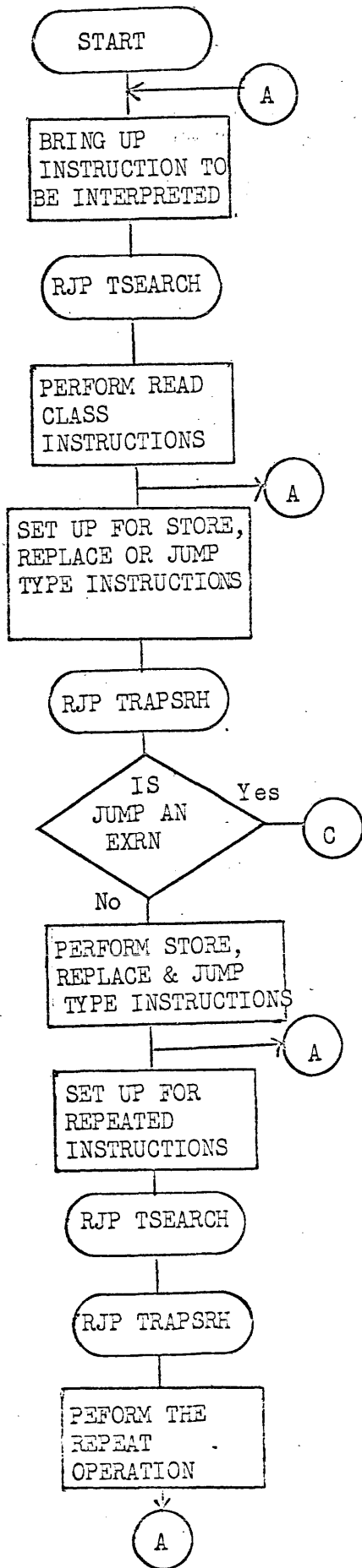
1. Control comes from the TSEARCH Routine and B7 holds the address of the test packet.
2. Test logical switch. If it is off EXIT.
3. Test segment #. If it is not the correct one then EXIT.
4. Perform Set:
  - a. Pick up the P setting and number of words, if they are equal to all sevens then EXIT.
  - b. Transfer the number of words, specified in the lower of the first word, to the store address specified by the upper of the first word.
  - c. Increment to the next table and go to step a.

### Test-EXIT-and Perform

1. Control comes from the TSEARCH Routine and B7 holds the location of the test packet.
2. Test the logical switch. If it is off then EXIT.
3. Test the segment number. If it is incorrect then EXIT.
4. Set up the ending logic and terminate the activity under test.



INTERPRETIVE EXECUTOR SEG III



## Table Look Up Routine

### TSEARCH

1. The function of this routine is to search through the JUMPTABLE for all Debugging Statements and then give control to the proper subroutine.
2. Control comes from the interpretive executor by means of a return jump.
3. The total number of tests is entered into B7 and the search is begun from the bottom of the JUMPTABLE. B7 is used to carry the position in the jump table. B7 is also used to locate the corresponding position of the Parameter table.
4. Whenever a HIT is found B7 is stored in the upper of TSEARCH and Control is given to the proper Debug Statement subroutine.
5. When Control is returned from the Debug Statement, B7 is entered with the upper of TSEARCH and the Search continues until the entire table has been searched. Then exit.

## Table Look Up Routine

### TRAPSRH

1. The function of this routine is to search through the JUMPTABLE. The method is the same as in the TSEARCH routine. It differs only in that whenever a hit is detected a further check is made to see if this is a request for a Trap Operation. If it is not then B7 is decremented and the JUMPTABLE Search is continued.
2. Control comes to this routine from the interpretive executor and only whenever the program under test tries to store to memory.
3. The routine exits only whenever the complete JUMPTABLE has been searched.

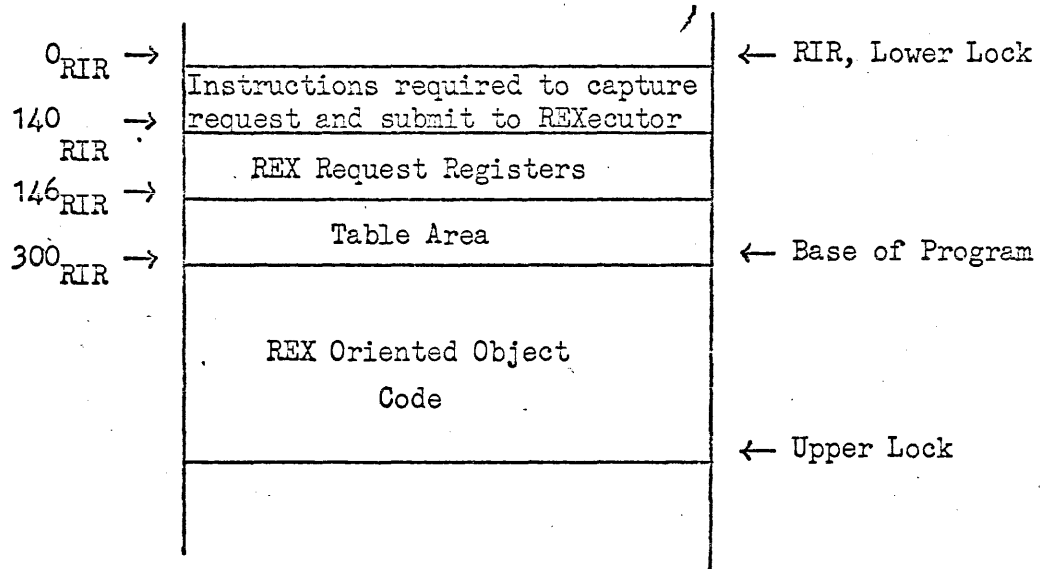
## 9.6 REXecutor

The 494 REXecutor is composed of a pair of secondary Exec routines developed to load and control the execution of batch type programs coded for use in a REX environment on the UNIVAC 490. This routine provides the user an ability to run one or more REX oriented programs concurrently with OMEGA oriented programs on the UNIVAC 494.

### 9.6.1 Method of Operation

Programs loaded and controlled by the REXecutor are activated and described via primary and secondary control statements embedded within a Job deck or contained on the drum in the form of a source element. As the program is loaded an execution area is allocated as part of the program to capture REX requests and contain parameters required to execute the program.

Upon completion of load, an allocation process, the described program is activated and allowed to run in a normal manner until it executes a return jump to the REX entry table (Locations 140-146). At this point the REXecutor will capture the request, reformat to OMEGA conventions and execute. Upon completion of the request program control will be returned by REX conventions with REX status codes. The following figure illustrates core layout which makes this possible.



## 9.6.2 The REX Control Statement

The REX Control Statement signals the start of a task which, selected as any other OMEGA recognized task, will activate the load and execution of REX oriented batch program(s). All peripherals and mass storage assignment used by REXecuted program must have been previously assigned to the task via ASG\$ control statements.

### REX Statement

#REX Δ Options Δ Minimum/Maximum, Name/Version, Library

#### Options are:

- X - Abort. Do not execute remainder of Job stream if errors encountered during load or execution of described program(s).
- Y - Continue REXecutor Phase 2 even through errors were detected during load and interpretation of control cards.

Absence of X or Y options implies terminate task and skip to next task within Job stream if errors are detected during load or interpretative execution of REX service requests.

#### Specifications are:

Minimum/Maximum is the amount of core storage, in units of 100g words, required to load complex and/or simple relative program(s) to be REXecuted. Selection will automatically increase specified amount of 400g words to compensate for execution area. Maximum core including execution area may not exceed 32K.

Name/Version (optional) contains name and version of a source element on random storage containing secondary REXecutor language statements required to process the task.

Library Field (optional) identifies the library in which the source element is contained on random access storage when name/version option is used. Any one of the following specifications are valid:

SYSTEM specifies the named source element is contained in systems library.

JOB specifies the named source element is contained in the Users Job library.

GROUP/Library Number specifies the named source element is contained in the named group library previously linked to the job.

In absence of library specification Job Library is assumed by the REXecutor.

### Errata Cards

Errata cards contain corrections to REXecuted programs and must follow PROG statement for which they pertain. They follow the identical format used by REX.

<u>Card Column</u>	<u>Content</u>
1	"A" identifies card as errata.
2-3	Segment number (blank implies control segment).
4-5	Unused
6-10	Address relative to execution area; 400g plus relative address within program, in which first errata word is to be stored.
11-80	Seven 10 character fields used to contain errata values. Errata words will be loaded at consecutive addresses starting with address given in columns 6-10. A blank correction word will terminate the card.

### Parameter Cards

Parameter cards contain data to be stored within program at address specified by Executive Information Region (E.I.R.) of the loaded program. Parameter cards must follow PROG statement for which they pertain. Two types of parameter cards are recognized, numeric and alphanumeric.

#### Numeric Parameters:

<u>Card Column</u>	<u>Content</u>
1	"B" identifies card as containing numeric parameters.
2-3	Number of 30-bit words to be filled from content of this card. Blank implies 7, the maximum.
4-10	Unused
11-80	Data storage - Seven 10 octal character fields to be stored in octal form. Blank columns will be entered as zeros. Words will be unstrung and stored left to right until specified number of words have been stored.

#### Alphanumeric Parameters:

<u>Card Column</u>	<u>Content</u>
1	"C" identifies card as containing alphanumeric parameters.
2-3	Octal number specifying the number of computer words to be filled from this card. Blank implies 16 octal, the maximum.
4-10	Unused
11-80	Alphanumeric data words - 16, 5 character fields to be stored in their fielddata form. Words will be stored left to right until specified number of words have been stored.

### 9.6.3 Secondary Control Statements

Secondary Control Statements immediately follow REX control card or are contained on random storage; and, describe program(s) to be loaded, REX parameters or errata, and map peripheral and mass storage assignments from OMEGA to REX conventions. Secondary control statements are terminated by PS (Program Start) control statement.

#### The PROG Control Statement

The PROGRAM control statement is used to describe a simple or complex relative (SPURT outputs 321 or 322 respectively) program which is to be loaded under control of the REXecutor. Any number of PROG cards can be submitted for any one execution to effect subroutine load. The only constraints placed upon complex or simple relative programs is that total amount core required by programs plus the REXecutor execution area does not exceed 32K and that only one of the programs is segmented. The PROG control statement is as follows:

△PROG△Options△File Code, Program ID/Name, Base, Logical Address

Options are:

- P Described program is the main code capable of being segmented and receiving REX parameters. Only one object code program may contain "P" option for any one REXecution.
- L Rewind load servo before search of the program.
- R Release load medium assignment after program load.

Specifications are:

File Code the alphabetic file code of logical unit on which described object code resides normal magnetic. If assignment is random access storage "logical address" is required to define starting point of code.

Program ID/Name identifies the simple or complex relative program to be located and modified to running form. Program ID is the ten character searchword of the code e.g. 74747 lib #. Program name is ten character name assigned to the program, this field is optional if program ID is unique.

Base, optional, used to specify a specific relative address on which the described program is to be loaded. If blank REXecutor will load program beginning at the first available cell following execution area previously loaded programs. When specified, value must be greater than 400<sub>8</sub>. Normally used to load additional subroutines.

Logical Address required only when random access storage is assigned as the source medium of the object program. Logical address is relative to assigned "file code" and is the beginning address of the object code.

#### The MEANS Control Statement

All peripheral and random access storage assignments used by the program(s) placed under control of the REXecutor must have been previously defined via the ASG control cards (see 5-1 of Programmer's Reference Manual) and described to the REXecutor via the MEANS statements. This feature not only insures proper execution of the program but allows a degree of flexibility in altering peripheral type. For example, a program written for UNISERVO IIA tape units can be changed to UNISERVO VIC tape units without reprogramming or reassembly of object program.

The only exception to the facility assignment procedure is in the use of card and high speed printer subsystems. The REXecutor will resubmit REX card and print packets to OMEGA's primary and/or secondary elements unless otherwise assigned.

The MEANS control statement is as follows:

△MEANS△Options△File Code, Channel/Unit, peripheral type, Drum  
Address/Length

Where options are:

A - Described unit is to be used to satisfy internal REX facility request when submitted by the program running in the REXecutor mode. Channel/unit may contain any unique channel/unit number not previously used by object code.

Specifications are:

File Code is the alphabetic file code to which the assignment has been previously assigned via the ASG control statement.

Channel/Unit is the numeric channel/unit as defined by the SPURT MEANS and ASSIGN statements. In the case of hand coded packets the channel/unit contained in the packet must be used. Channel/unit will be used at execution time of an I/O packet to map request back to OMEGA file code and not as a physical ch/unit.



Peripheral Type is the mnemonic REX peripheral type for which the original REX I/O packet or internal facility request was coded for. The following are the valid entries.

<u>Mnemonic</u>	<u>Subsystem</u>
FH880	FH880 Drum
FAST	FASTRAND
UN2A	UNISERVO IIA
UN3C	UNISERVO IIIIC/IVC/VIC/VIIIC
UN3A	UNISERVO IIIA
CRIN	Card Reader
CROUT	Card Punch
PRINT	High Speed Printer
PTIN	Paper Tape Reader
PTOUT	Paper Tape Punch
PIN	Submit request to primary input
POUT	Submit request to primary output
SOUT	Submit request to secondary output

Drum Address/Length is required only if assignment is for random access storage and existing program uses absolute drum addresses to reference file. For each distinct file assigned on the UNIVAC 494 and used by the program under execution the absolute base address and file length assumed by the 490 program must be specified. File length is conveyed as number of words or sectors contained in current file. These values are used to effect the transition from absolute drum reference to logical increment and to determine which of several files reference is intended.

Lack of specification implies program will use zero as the file base and that only one random access file is assigned.

Relocatable mass storage declared by complex relative (322) must be assigned to File code Y and mapped by MEANS statement. REXecutor will set up E.I.R. of REXecuted program to contain expressed value in the normal manner.

#### Segment Storage

The REXecutor allocates random access storage for segments and errata for a complex relative program. File code ZG is used for this purpose.

## The Program Start Statement

The Program Start Statement is used to terminate load process and activate execution of program.

△PS△Options△P, A, Q, B1/B2/B3/B4/B5/B6/B7

Where options are:

- D - perform octal dump of execution area upon completion of REXecuted program regardless of reason normal or abnormal.
- E - Perform octal dump of execution area upon completion only if error occurs during execution.
- P - Perform octal dump of execution area prior to execution of program.

Specifications are:

P contains alternate program start address relative to base of program area. Lack of specification implies use start location contained in E.I.R. of program.

A, Q & B's optional used to specify register setting at time of execution.

### 9.6.4 REXecutor Elements

The REXecutor itself is composed of three distinct elements: REXecutor Phase 1, REXecutor Phase 2, and Execution area.

Phase 1 is a non-reentrant secondary Exec routine responsible for the following functions related to load and control of REX oriented programs.

- . Interpret 2 unstrung control cards used to describe program(s) to be processed.
- . Load SPURT produced simple relative (321 output) or complex relative (322 output) programs and establish execution area.
- . Initialize and activated REXecuted program.
- . Submit all Error messages to primary output stream
- . Process STOPRUN and TERMRUN service requests.
- . Process REX Loader service requests other than segment load.
- . Load segment errata when present.

Phase 2 is a reentrant secondary Exec routine used to process normal REX Input/Output requests and their associated controls.

- . Order standard REX Input/Output service requests
- . Process REX Input/Output service requests
- . Process CKSTAT service request.
- . Process TAKEOVER service request.
- . Process Segment Loads
- . Return program control to REXecuted program.
- . Call Phase 1 for Error conditions and all non-processed Loader functions.

Execution Area is allocated at during Load phase to contain the following:

- . REX object code being executed.
- . Maps and tables showing peripheral assignments and segments description.
- . Pseudo REX entry locations and a limited set of instructions used to call Phase 1 or 2 upon each REX service request.
- . Work area used to submit reformatted I/O requests to OMEGA.

#### 9.6.5 REXecutor Phase 1

REXecutor Phase 1 library number XXX is a non reentrant secondary Exec routine used to load, activate and execute REX service request which require exception processing for reasons of core buffers or elaborate interpretation.

#### CALLS

Phase 1 is activated via EXRN\*2XXFC from Phase 2, or Execution area to perform some function with regard to load and control of REXecuted program. The following is a summary of function codes their caller and brief description of each followed by a functional description of each.

<u>Function Code</u>	<u>Caller</u>	<u>Description</u>
01	Execution	Load and interpret control cards describing REXecuted program.
02	Phase 2	Interpretively execute REX Loader service request.

<u>Function Code</u>	<u>Caller</u>	<u>Description</u>
03	Execution	Process console TYPEEC service requests
04	Phase 2	Process STOPRUN, TERMRUN or console functions.
05	Execution	Invalid REX service request
06	Phase 2	Submit diagnostic message
07	Execution	Program Fault
10	Execution	Memory Guard violation
11	Execution	Illegal instruction execution
12	Phase 2	Locate and load segment errata
13-37	Un-assigned	

Function Code 01 is activated as the result of selection encountering a REX primary control statement in the Job control stream and selecting it as the next task to be entered into the system. Upon activation selection has: assigned core area specified on REX control statement plus execution area, stored REX control statement in parameter storage with a "SEND\$" operator, loaded initial execution area element and given program control to execution area.

Initial instructions contained in execution area establish two queue process activity addendums: Activity 1 which is used to execute the program object code. Activity 2 which is used to execute input/output concurrently with instruction execution. Upon establishment of control threads a QREF is executed to Activity 1 and "RETURN\$" is executed.

**Purpose:** Load and establish REX oriented batch programs for execution within the OMEGA environment. This is accomplished by requiring and processing REXexutors secondary control statements.

**Caller:** Execution area as the result of QREF\$ to Activity addendum 1.

**Parameters:** Task addendum contains core area assigned to the Task and link to REX control statement which caused activation. REX control statement can be retrieved through use of the "RECEIVE\$" operator (See 9.6.2 for "REX" control statement").

- Functions:
- a) Retrieve REX control statement, unstring and set up necessary linkages to retrieve REXecutor secondary control statements from random storage or primary input upon request from Step (b).
  - b) Retrieve secondary control statement, submit to primary output for subsequent printing and jump to appropriate step dependent upon statement type. MEANS statement go to Step (c), PROG statement go to Step (d), parameter or errata go to Step E, PS statement go to Step (f). If not one of the above submit the following diagnostic number 1 and repeat Step (b).
  - c) Unstring and process MEANS statement setting on appropriate entry in Table 1A and/or Table 1. If table overflow or invalid parameter submit appropriate diagnostic and repeat Step (b).
  - d) Unstring and process PROG statement by locating and modifying to running form complex or simple relative program described on PROG card. If "P" option given program is loaded and E.I.R. is set up describing core bounds and any random access storage assigned to File Code "Y" and mapped by previous MEANS statement. If execution area overflow or errors detected submit appropriate diagnostic and repeat Step (b).

During Load process no modifications of ch/unit positions in REX I/O packets, T-tags or D-tags is performed. Therefore, original channel/unit numbers specified by "MEANS", FACIL, and ASSIGN SPURT directives will be contained in the packets at execution time and must conform to channel/unit numbers used on MEANS secondary control statement.

- e) Unstring and store REX parameter or errata cards by REX conventions. If segment errata, store errata on mass storage following program segments and set appropriate segment description bit in execution area indicating errata. Repeat Step (b).
- f) Unstring Program Start card, store register settings and starting address if given in the allocated storage module for this activity. Close any remaining initialization of execution area and return program control.

Exits: Normal return through "Content Supervisor" if REXecuted program(s) were successfully loaded or Y option on REX card given.

ERROR or ABORT return through "Content Supervisor" if errors detected during Load process. ABORT if X option on REX card, ERROR exit if X or Y option on REX card are absent.

Function Code 02 is activated as the result of a call from Phase 2 to process a service request normal directed to the REX Loader.

**Purpose:** Interpretively execute REX Loader service request and return program control to requestor.

**Parameters:** B4 set to address of SMOD making Phase 1 request.  
B4 of requesting SMOD set to address of SMOD containing REExecuted programs worker "B" registers.  
B5 of requesting SMOD set to address of execution area of program.  
B6 of requesting SMOD set to address of REXMOD allocated.  
"P" position of REXMOD contains address of requesting REX packet relative to execution area.

**Function:**

- . Retrieve REX Loader packet and process as summarized by Table 9-1.
- . Set Normal and Abnormal return address in REXMOD to "P" plus packet length. Set B7 of REXMOD to  $\emptyset$  if no errors were encountered during process of request. Set B7 non-zero if errors.

**Exit:** Return control to Phase 2.

Function Code	REX Service Request	Packet Length	Processed	Responsible Phase	E.A.S. Given	REXecutor Action Taken
00-10	Un-assigned	-	-	-	-	REXecuted program is terminated.
11	Site Utility	?	NO	-	1	REXecuted program is terminated.
12	Un-assigned	-	-	-	-	REXecuted program is terminated.
13	Real Time initialization	12	NO	1	NO	Skip and mark done.
14	Start Slave	2	NO	1	YES	Skip and mark done.
15	Rerun Dump	V	NO	1	NO	Skip and mark done.
16	Print Drum	4	YES	1	YES	Process & submit images to primary output
17	Print core	4	YES	1	YES	" " " " " "
20	Subroutine load or R.T. extension	4	NO	1	YES	Skip and mark done.
21	Internal Load request	3	NO	1	YES	Skip and mark done.
22	Allocation request	2,4	YES	1	YES	Locate previously reserved unit.
23	Segment Load	2	YES	2	NO	Load segment and segment errata.
24	Peripheral unit release	2	YES	1	NO	Release mapped unit to OMEGA.
25	Core release	2	NO	1	NO	Skip and mark done.
26	Random storage release	3	YES	1	NO	Release mapped storage to OMEGA.
27-7777	Un-assigned	-	-	-	-	REXecuted program is terminated.

Table 9-1

- ACCEPT - submit following request to OMEGA console handler.

ENT\*B7\*Message Address  
EXRN\*1Ø1Ø2

Where message contains zero characters to print. Upon return of control store A and Q registers in ACCEPT specified buffer and exit to Phase 2. ACCEPT requests will be limited to 10 characters excluding stop code.

- STOP and TERMRUN - Submit appropriate termination message to primary output stream, set "done" indicator to execution area and exit to Phase 2.

Exit: Return control to Phase 2 with normal and abnormal return address of REXMOD set to end of packet.. B7 of REXMOD should be clear to indicate successful completion.

Function Code 05 - Activated when an invalid REX service request is made.

Caller: Execution area under Activity 1 of REXecuted program when it performs one of the following calls through REX pseudo interrupt registers. U and L of 136, 137, 143, 145. U of 144, 146; L of 142.

Parameters: B7 of calling SMOD contains "P" of request relative to execution area.

Function: Submit following diagnostic to primary output stream; set ABORT indicator in execution area and abort control thread.

ILLEGAL REX CALL

Exit: Exit to Content Supervisor with a abort control thread option.



Function Code 06 - Used by all routines to submit diagnostic messages.

Caller: Phase 2, Execution area or an internal request from Phase 1 under either Activity 1 or 2 of REXecuted program.

Parameters: B3 contains message number.

Function: . Retrieve and submit to primary diagnostic message indicated by B3.

. Set indicators Done or Abort in execution area dependent upon message number (see Table 9-2).

Exit: . Return control to requestor or abort control thread dependent upon message options.

Function Code 07 - Used to process Program Fault(s) executed by REXecuted program.

Caller: Execution area under Activity 1.

Parameters: Set to address of fault location.

Function: . Return program control to fault recovery address specified in REXecuted programs E.I.R. If fault recovery address not provided submit diagnostic and terminate task.

Exit: . Return program control to fault recovery address if given.

. Terminate task if no recovery point given.

Function Code 10 - Activated as the result of memory guard violation

Caller: Execution area under Activity 1.

Function: Submit diagnostic message 27.

Exit: Abort control thread through Content Supervisor.

Function Code 11 - Activated as the result of executing a privileged instruction (function code 776X-777X, except 7771 and 7775, and I/O instructions).

Caller: Execution area under Activity 1.

## Function Code 11 (Continued)

**Function:** Return program control to fault recovery address specified in REXecuted programs E.I.R. If fault recovery address not provided submit diagnostic message 30.

**Exit:**

- . Return program control to fault recovery address if specified.
- . Abort control thread if no recovery address given.

## Function Code 12 - Load segment errata.

**Caller:** Phase 2 as the result of a segment load.

**Parameters:** B2 of storage module contains segment number loaded.

**Function:** Locate and load segment errata for given segment.

**Exit:** Return control to Phase 2.

### 9.6.6 Diagnostic Messages

An element of Phase 1 within the REXecutor is responsible for submission of all diagnostic messages conveyed to primary output stream by the REXecutor. Upon activation of diagnostic routine B3 contains binary number of message to be submitted. B2 may contain supplementary information; normally a binary address to be strung into a five character numeric field and submitted along with the message.

Message number is used to retrieve a one word Enter within a table which describes message and action to be taken upon completion of submission. Format is as follows (see Table 9-2 for messages):

<u>Bit Position</u>	<u>Content</u>
29	Set to "1" implies additional data contained in B2 which is to be masked into last position of message before submission.
28	Set to "1" request a check of "P" option from Program Start card if given dump Execution area.
27	Set to "1" request a check of "E" option from Program Start card if given dump Execution area.
26	Set to "1" - Set "abort" indicator in Execution area U (word X).
25	Set to "1" - Set "Done" indicator in Execution area U (word X).

Bit PositionContent

24	Set to "1" - return control to requestor.
23	Set to "1" - set SMOD to ABORT\$ exit if "X" option given on "REX" card.
22	Set to "1" - return control to requestor if "Y" option given.
21	Set to "1" - set SMOD to ERROR\$ exit.
20-19	Unused
15-18	Length of diagnostic message.
0-14	Address of diagnostic message.

TABLE 9-2

MSG #	29	28	27	26	25	24	23	22	21	20	19	18	15	14	0	MESSAGE LITERAL
00	1	1	1	1	0	0	1	0	1				6			ERR0 INVALID REXECUTOR CALL XXXXX
01	0	0	0	0	0	0	1	0	1				7			ERR1 REXECUTOR CONTROL STATEMENT ERROR
02	0	0	0	0	0	0	1	0	1				4			ERR2 MEANS TABLE OVERFLOW
03	0	0	0	0	0	0	1	0	1				4			ERR3 PROGRAM NOT FOUND
04	0	0	0	0	0	0	1	0	1				5			ERR4 CODE MODIFICATION ERROR
05	0	0	0	0	0	0	1	0	1				5			ERR5 CORE STORAGE OVERFLOW
06	0	0	0	0	0	0	1	0	1				5			ERR6 PROGRAM CODE UNREADABLE
07	0	0	0	0	0	0	1	0	1				6			ERR7 PROGRAM LOAD MEDIA UN-ASSIGNED
10	0	0	0	0	0	0	1	0	1				5			ERR10 SEGMENT STORAGE OVERFLOW
11	0	0	0	0	0	0	1	0	1				5			ERR11 INVALID ERRATA ADDRESS
12	0	0	0	0	0	0	1	0	1				5			ERR12 INVALID PARAMETER STORAGE
13	1	1	1	0	0	0	1	0	1				7			ERR13 INVALID PROGRAM START GIVEN XXXXX
14	1	1	1	1	0	0	1	0	1				5			ERR14 ILLEGAL REX CALL XXXXX
15	1	1	1	1	0	0	1	0	1				7			ERR15 INVALID REX LOADER REQUEST XXXXX
16	0	1	1	1	0	0	1	1	1				6			ERR16 NON DESCRIBED CHANNEL AND UNIT
17	0	1	1	1	0	0	1	1	1				6			ERR17 NON DESCRIBED DRUM ADDRESS
20	0	1	1	1	0	0	1	1	1				4			ERR20 FACILITY UN-ASSIGNED
21	0	1	1	1	0	0	1	1	1				6			ERR21 UN-TRANSLATABLE REX PARAMETER
22	0	1	1	1	0	0	1	1	1				7			ERR22 INVALID INPUT/OUTPUT FUNCTION CODE
23	0	1	1	1	0	0	1	0	1				3			ERR23 REXMOD OVERFLOW
24	1	1	0	0	1	1	0	0	0				3			ERR24 REX TERMRUN
25	1	1	0	0	1	1	0	0	0				3			ERR25 REX STOPRUN
26	1	1	1	1	0	0	1	0	1				4			ERR26 PROGRAM FAULT XXXXX
27	0	1	1	1	0	0	1	0	1				5			ERR27 PROGRAM LOCK VIOLATION
30	0	1	1	1	0	0	1	0	1				7			ERR30 PRIVILEGED INSTRUCTION VIOLATION
31	0	0	0	0	0	0	0	1	0				10			ERR31 REXECUTOR CONTROL CARD UN-RECOGNIZABLE
32	0	1	1	1	0	0	1	1	1				6			ERR32 I/O ERROR NO ERROR ADDRESS

9.6.7 REXecutor Phase 2 - Library Number XXX is a re-entrant Secondary Exec routine responsible for reformatting and executing standard REX I/O and associated controls: CKSTAT and TAKEOVER. Phase 2 is activated by the following function codes.

<u>Function Code</u>	<u>Call</u>	<u>Description</u>
01	Internal	Reformat and execute I/O request
02	Internal	Return program control to eligible lost control point
03	U(140)	REX input/output call
04	L(140)	REX CKSTAT call
05	U(141)	REX TAKEOVER call
06	L(144)	REX LOADER call
07	U(142)	REX console request termrun or stoprun
10-37	Unassigned	

Function Code 01 - Used to translate and execute REX input/output request serial for any REXecuted program. See Table 9-3 for REX functions available.

Caller: REXecutor, Phase 2, as the result of a QREF to activity addendum 2 of REXecuted program.

Parameters: . B4 - Set to address of storage module containing B register settings at time of request. "A" register position of storage module contains address of assigned REXMOD.

. B5 - Set to address of execution area of REXecuted program.

Function: . Clear execution packet and retrieve REX call packet. Locate equivalent OMEGA assignment from Table 9-1 or 1A through use of unit and/or channel number contained in REX packet.

. If peripheral type is FH880 or FASTRAND translate drum address to logical increment and store in OMEGA packet.

. If peripheral type is 13, 14 or 15 branch to routine to set up and execute appropriate request to Cooperative Control.

. With peripheral type retrieve function code from REX packet. Retrieve "schedule number" from Table 9-4 by referencing word based on REX function code and location within word based on peripheral type.

. Access correct location of Table 9-5 based on "schedule number" obtained from Table 9-4 and perform the following:

If  $2^{29}$  set, retrieve buffer control word from REX packet and store as base address and number of words in OMEGA packet (base address is stored as relative to lower lock of the Phase 2 routine).

If  $2^{28}$  set retrieve search word from REX packet and store in OMEGA packet.

Enter OMEGA function code in B1 and give control to the execution routine indicated by Table 9-5.

. The execution routines perform or simulate (Table 9-6) REX I/O requests by issuing I/O requests to OMEGA. Certain REX requests may require multiple OMEGA functions: (e.g. the execution routine responsible for BACKFILE would issue READBS\$ functions until an end-of-file is encountered). The following parameters

are required by the execution routines:

- B1 - OMEGA function code
- B4 - Storage module address
- B5 - Execution area address
- B6 - REX I/O packet address

At the completion of their operations the execution routines pass control to the Status routine.

- The Status routine converts the OMEGA status code to REX code and stores the appropriate code and indicators in the REXMOD. The parameters required for operation are as follows:

- B4 - Storage module address
- B5 - Execution area address
- A - OMEGA status code
- Q - Number of words transferred.

The status routine performs the following functions:

Stores the "number of words transferred" in A upper of the REXMOD.

Using peripheral type and OMEGA status determine REX status code. OMEGA status codes 0, 7 and 11 are converted directly to REX status codes 01, 10 and 06 respectively. OMEGA status code 10 will cause an abort condition (message number 16). The REX status codes corresponding to the remaining OMEGA codes vary with the peripheral type, and are obtained by referencing Table 9-7. A REX status code of 00 obtained from Table 9-7 will cause an abort condition.

The REX status is placed in A-lower of the REXMOD. (Note: Abnormal frame counts on magnetic tape units will be indicated as in REX: A-lower equals 01, A-upper equals number of words transferred with  $2^{29} = 1$ , and the magnitude of frame count error indicated in the lower six bits of the last buffer word).

B7 of the REXMOD is set to indicate normal or abnormal completion. "0" indicates normal, "1" indicates abnormal. If abnormal, the address of the REX I/O request is placed in Q-upper and the normal completion address in Q-lower of the REXMOD.

Exit:

- If REX function performed, control will be given "EXIT" routine (See Function code 02).

- If REX function cannot be performed program's "ABORT" indicator will be set and a direct switch to REXecutor Phase 1, Function code 06; with B3 set to one of the following diagnostic messages.

B3 = 16 NON DESCRIBED REX CHANNEL AND UNIT

B3 = 17 NON DESCRIBED REX DRUM ADDRESS

B3 = 21 UNTRANSLATABLE REX PARAMETERS

B3 = 20 FACILITY UNASSIGNED

B3 = 22 INVALID INPUT/OUTPUT FUNCTION CODE



Function Code 02 - responsible for returning program control to REXecuted code upon completion of Input/Output, Loader, etc., service requests.

Caller: Phase 2 internally or Phase 1 by direct call under either Activity 1 or 2.

Parameters: Phase 2 set B5 to address of execution area.

Function: A) Search for REXMOD with a 6 status code indicating some type of request without E.A.S. was given and is complete. If find made perform Step C.

B) If TAKEOVER is not set exit. If TAKEOVER indicator is set search for REXMOD with 3 status code indicating some type of request with E.A.S. was given and is complete. If find made clear TAKEOVER indicator and perform Step C.

C) Set up REXMOD for return of control.

- If B7 of REXMOD = 0 set B1 to normal return.
- If B7 of REXMOD  $\neq$  0 Set B1 to abnormal return.
- Set REXMOD status to 77777.
- Set B2 to REXMOD address relative to execution area.
- Enter A and Q from REXMOD
- QREF Activity 1 and Exit

Exit: Set P and B3 of calling SMOD to execute RETURN\$ upon exit from Content Supervisor.

If abnormal return indicated and error address of REXMOD = 1, Set B3=32 and exit to Phase 1, Function Code 06.

Function Code 03 - Used to queue requests for normal I/O.

Caller: Activity 1 REXecuted program when it performs a call to the U(140).

Parameters: B7 set to address, relative to RIR, of request.

Function: • Set up REXMOD, P,1  $\rightarrow$  status  
• QREF Activity 2 for execution of Function code 01 with B1 through B6 set to those of requestor. "A" set to address of REXMOD.

Exit: • Phase 1 of REXecutor if REXMOD overflow.  
• Return program control to end of REX I/O packet.

Function code 04 - Used to process a CKSTAT request

Caller: Activity 1 of REXecuted program when it performs a call to L(140).

Parameters: B7 set to the address, relative to RIR, of the CKSTAT request.

Function: .Search for REXMOD which contains addressed I/O request. If find is made set "Normal and Error" addresses and store worker B-registers in REXMOD. If find is not made locate available REXMOD, set "Normal and Error" addresses, store B-registers, set B7 and A of REXMOD to 1 and  $\emptyset$  respectively, and set status to 2.

.If EAS =  $\emptyset$  increment status by 4. If EAS = 1 (TAKEOVER) increment status by 1 and set TAKEOVER indicator. IF EAS  $\neq$   $\emptyset$  or 1, increment status by 1.

Exit: .Phase 1 if REXMOD overflow occurred.  
.Return control to REXecuted program when EAS $\neq$  $\emptyset$  or 1.  
.Go to "EXIT" routine when EAS =  $\emptyset$  or 1.

Function code 05 - Responsible for processing REX TAKEOVER.

Caller: Execution area upon SILRJP\*U(141) under activity addendum 1.

Parameters: Phase 2 set B5 to address of execution area.

Function: Set TAKEOVER indicator in execution area.

Exit: Give program control to exit routine, See Function code 02.

Function code 06 - Responsible for initial set up for processing of REX and/or REX loader service requests. (See table 9-1 for detailed description of each)

Caller: Execution area upon SILRJP\*L(144) under activity addendum 1.

Parameters: B4 set to address of SMOD containing B1-B6, A and Q at time of request. B7 of SMOD contains packet address relative to execution area. Phase 2 sets B5 to address of execution area.

Function: A) Retrieve REXMOD, set status = 2, store worker B1-B6, and "P" of packet. If request for segment

load perform Step C.

- B) For non-segment loads call Phase 1 Function code 02 for processing. Upon return of control check E.A.S. position. If zero E.A.S., update status by 4 and go to exit routine (See Function code 02, Phase 2). If E.A.S. = 1, set TAKEOVER indicator, update status position by 1, and go to exit routine. If E.A.S. specifies an address, update status by 1 and return control to specified E.A.S.
- C) Load indicated segment and if segment errata indicated, call Phase 1 Function code 12 for errata placement in segment. Upon completion of load process update REXMOD status code by 4, update normal and abnormal return addresses and give program control to exit routine.

- Exit:
- If request contained E.A.S., address update REXMOD for return of control, set SMOD P address to E.A.S. and return control through Content Supervisor.
  - If no E.A.S. or an E.A.S. of 1, update REXMOD for return of program and go to "exit" routine.

Function code 07 - responsible for initiating console requests or terminating the REXecuted task.

Caller: Execution area upon SIIRJP\*U(142) under activity addendum 1.

Parameters: B4 set to address of SMOD containing B1-B6, A and Q registers at time of request. B7 of SMOD contains packet address relative to execution area. Phase 2 sets B5 to address of execution area.

Function: Retrieve REX service requests function code and process according to type. CONSOLE HOLD or RELEASE update "P" of SMOD and return control to program (ignored). If REX STOPRUN, TERMRUN or TYPET, call Phase 1 function 04 for processing. If ACCEPT request, call Phase 1 function 04 and process E.A.S. as described in Step A and B of Function code 06, Phase 2.

- Exit:
- CONSOLE HOLD, RELEASE, TYPET or ACCEPT with E.A.S. address specified, return program control to REXecuted program.
  - TERMRUN or STOPRUN abort activity through Content Supervisor.
  - ACCEPT without E.A.S. or with E.A.S. = 1, update REXMOD for return of control and go to exit routine.

9.6.7-8

Function	Omega FH880 Drum	FASTRAND	UNISERVO IIA	UNISERVO IIC	UNISERVO IIA Card Subsystem	MSF Subsystem	Paper Tape
READ	READ	READ	READF	READ	READF	READ*	READF
WRITE	WRITE	WRITE	WRITE	WRITE	WRITE	PUNCH*	PUNCH
REWIND	-	-	REWIND	REWIND	REWIND	-	-
REWINDI	-	-	RWI	RWI	RWI	-	-
READB	-	-	READB	-	READB	-	READB
READ(Note1)	-	-	MOVEF	MOVEF	MOVEF	-	-
READ(Note2)	-	-	MOVEB	MOVEB	MOVEB	-	-
SEARCH	SEARCH	WSEARCH	SEARCHF	SEARCH	SEARCHF	-	-
READB(Note3)	-	-	SEARCHB	-	SEARCHB	-	-
SEARCH	LOCATE	-	LOCATEF	-	LOCATEF	-	-
READB(Note2)	-	-	LOCATEB	-	LOCATEB	-	-
WRITE	-	-	PWRITE	-	-	PUNCHS*	-
READ(Note3)	-	-	-	BACKFILE	-	-	-
IGNORED(Note4)	-	-	-	LOBIN	-	RMODE*M	PIN
IGNORED(Note4)	-	-	-	HIBIN	-	PMODE*M	-
IGNORED(Note4)	-	-	-	LOBCD	-	-	-
IGNORED(Note4)	-	-	-	HIBCD	-	-	-
WRITEOF	-	-	-	ENDFILE	ENDFILE	-	-
ERASE	-	-	-	-	WRITED	-	-
SEARCHP	-	SEARCH	-	-	-	-	-
Ignored	-	POSITION	-	-	-	-	-
BLOCKR	BREAD	-	-	-	-	STACK1,1	-
BLOCKS	BSEARCH	-	-	-	-	-	-
BLOCKS	BLOCATE	-	-	-	-	-	-
Ignored	-	-	-	-	-	READNT	-
Ignored	-	-	-	-	-	TRANS	-
MREAD	-	-	-	-	-	MREAD*	-

Note 1 Repetitive Read N blocks  
 Note 2 Repetitive Read back N blocks  
 Note 3 Repetitive Read back to E.O.F.  
 Note 4 Effected via ASG control card

\* Will be submitted to cooperative system MEANS card had PI, PO or BO given for logical unit.

TABLE 9-4 is used to translate REX function code to simulated OMEGA routine and schedule (Table 3).

TABLE 9-4 in REX Mnemonics

	FC 2 <sup>29</sup> -2 <sup>24</sup> FH-880	FC 2 <sup>29</sup> -2 <sup>24</sup> FASTRAND	FC 2 <sup>29</sup> -2 <sup>24</sup> UNISERVO IIA,IIIA	FC 2 <sup>29</sup> -2 <sup>21</sup> UNISERVO IIIC	FC 2 <sup>29</sup> -2 <sup>24</sup> CARD
00	WRITE	WRITE	REWIND	LOBIN	STACK1
01	READ	READ	REWINDI	HIBIN	STACK2
02	BREAD	SEARCH	SEARCHB	LOBCD	READNT
03	LOCATE	WSEARCH	SEARCHF	HIBCD	RMODE*FD
04	BLOCATE	POSITION	PWRITE	SEARCH	RMODE*CBIN
05	SEARCH	-	WRITEO	WRITE	RMODE*RBIN
06	BSEARCH	-	WRITE	READ	READ
07	-	-	ENDFILE	ENDFILE	TRANS
10	-	-	READF	REWIND	MREAD
11	-	-	-	REWINDI	PUNCH
12	-	-	READB	BACKFILE	PUNCHS
13	-	-	MOVEF	MOVEB	PMODE*FD
14	-	-	MOVEB	MOVEF	PMODE*CBIN
15	-	-	-	-	PMODE*RBIN

00	03	03	04	01	01	
01	02	02	05	01	01	
02	11	17	20	01	21	
03	10	24	10	01	01	
04	12	01	03	10	01	
05	10	00	05	03	01	
06	12	00	03	02	02	
07	00	00	07	07	22	
10	00	00	02	04	23	
11	00	00	00	05	03	
12	00	00	13	16	03	
13	00	00	14	15	01	
14	00	00	15	14	01	
15	00	00	00	00	01	
	2 <sup>29</sup>	2 <sup>24</sup>	2 <sup>19</sup>	2 <sup>14</sup>	2 <sup>9</sup>	2 <sup>4</sup>

TABLE 9-4 Actual-Each position contains an increment to the equivalent OMEGA routine described by an entry in Table 9-5.

SCHEDULE HANDLER	SEARCH I.D.			OMEGA FUNCTION CODE	ADDRESS OF EXECUTION ROUTINE	OMEGA FUNCTION
	BCW	29	28			
00	0	0	0	0 0 0 0	IOERR	Error
01	0	0	0	0 0 0 0	I01	NO-OP
02	1	0	0	0 0 0 1	I02	READ\$
03	1	0	0	0 0 0 2	I02	WRITE\$
04	0	0	0	0 0 2 1	I03	REWIND\$
05	0	0	0	0 0 2 2	I03	REWINDI\$
06	0	0	0	0 0 2 3	I03	ERASE\$
07	0	0	0	0 0 2 0	I03	WRITEOF\$
10	1	1	0	0 1 0 6	I04	SEARCH\$
11	1	0	0	0 0 0 3	I02	BLOCKR\$
12	1	1	0	0 0 0 7	I02	BLOCKS\$
13	1	0	0	0 0 1 1	I02	READB\$
14	*1	0	0	0 0 0 1	I05	READ\$
15	*1	0	0	0 0 1 1	I05	READB\$
16	0	0	0	0 0 1 1	I06	READB\$
17	1	1	1	—	I11	SEARCHT\$, SEARCHP\$
20	1	1	0	0 0 1 1	I07	READB\$
21	0	0	0	0 0 0 0	I08	Special READNT
22	1	0	0	0 0 0 1	I09	Special TRANS
23	1	0	0	0 0 0 1	I10	Special MREAD
24	1	1	1	0 1 0 6	I12	SEARCH\$

\* # of blocks to move

TABLE 9-5 Schedule List

TABLE 9-6

△ from Table 2	Label of Simulator Routine	OMEGA function used to simulate	Description of Simulated Function
00	ERR1	0—0	Error MSG for invalid function REX function code
01	I01	0—0	No-operation - No simulation required.
02	I02	READ\$	Normal READ operation
03	I02	WRITE\$	Normal WRITE operation
04	I03	REWIND\$	Normal Rewind operation
05	I03	REWINDI\$	Normal Rewind with interlock operation
06	I03	ERASE\$	Normal overwrite operation
07	I03	WRITEOF\$	Normal write EOF mark
10	I04	SEARCH\$	Normal Search using allocated buffer and BCW
11	I02	BLOCKR\$	Normal FH-880 Block Read operation with logical address
12	I02	BLOCKS\$	Normal FH-880 Block locate operation with logical address
13	I02	READB\$	Normal tape Read back
14	I05	READ\$	Simulate MOVE forward N blocks
15	I05	READB\$	Simulate MOVE back N blocks
16	I06	READB\$	Simulate BACKFILE operation
17	I11	SEARCHP\$, SEARCHP\$	Simulate short and long first word search
20	I07	READB\$	Simulate SEARCHB operation
21	I08	—	Simulate READNT operation
22	I09	(READ\$)	Simulate TRANS operation
23	I10	READ\$	Simulate MREAD operation
24	I12	SEARCH\$	Simulate SEARCH

OMEGA STATUS P-TYPE	Successful Completion	Inapprop. Function	Incorrect Parameter	Unrecoverable Error	End-of- File	End-of- Tape	Unsuccess. Search	Illegal Char.	No Assign	Inter- lock
DRUM	Successful	Incorrect Parameter	Incorrect Parameter	Read Error	Illegal Address		EOB W/O Find		Abort	Interlock
FASTRAND	Successful	Incorrect Parameter	Incorrect Parameter	Subsystem Error	Incorr. Para- meter		Unsuccess. Search		Abort	Interlock
UNISERVO IIA	Successful	Incorrect Parameter	Incorrect Parameter	R/W Error	End-of- File	End-of- Tape			Abort	Interlock
UNISERVO IIIC IVC/VIC/VIIIC	Successful	Incorrect Parameter	Incorrect Parameter	Subsystem Error	End-of- File	End-of- Tape			Abort	Interlock
UNISERVO IIIA	Successful	Incorrect Parameter	Incorrect Parameter	Subsystem Error	End-of- File	End-of- Tape			Abort	Interlock
CARD	Successful	Inapprop. Function	Incorrect Parameter	Illegal Function				Illegal Char.	Abort	Interlock
PRINTER	Successful	Illegal Function	Illegal Parameter	Illegal Error					Abort	Interlock
PAPER TAPE	Successful	Incorrect Parameter	Incorrect Parameter	Subsystem Error					Abort	Interlock

TABLE 9-7

REX STATUS CODES



TABLE 9 - 7 ACTUAL

OMEGA STATUS P-TYPE	1	2	3	4	5	6
1	16	16	12	11	00	03
2	10	10	07	10	00	02
3	12	12	10	04	05	00
4	10	10	07	04	05	00
5	10	10	07	04	05	00
6	12	13	11	00	00	00
7	12	13	11	00	00	00
10	10	10	07	00	00	00
11	10	10	07	00	00	00
12	10	10	07	00	00	00
	2 <sup>29</sup>	2 <sup>24</sup>	2 <sup>19</sup>	2 <sup>14</sup>	2 <sup>9</sup>	2 <sup>4</sup>

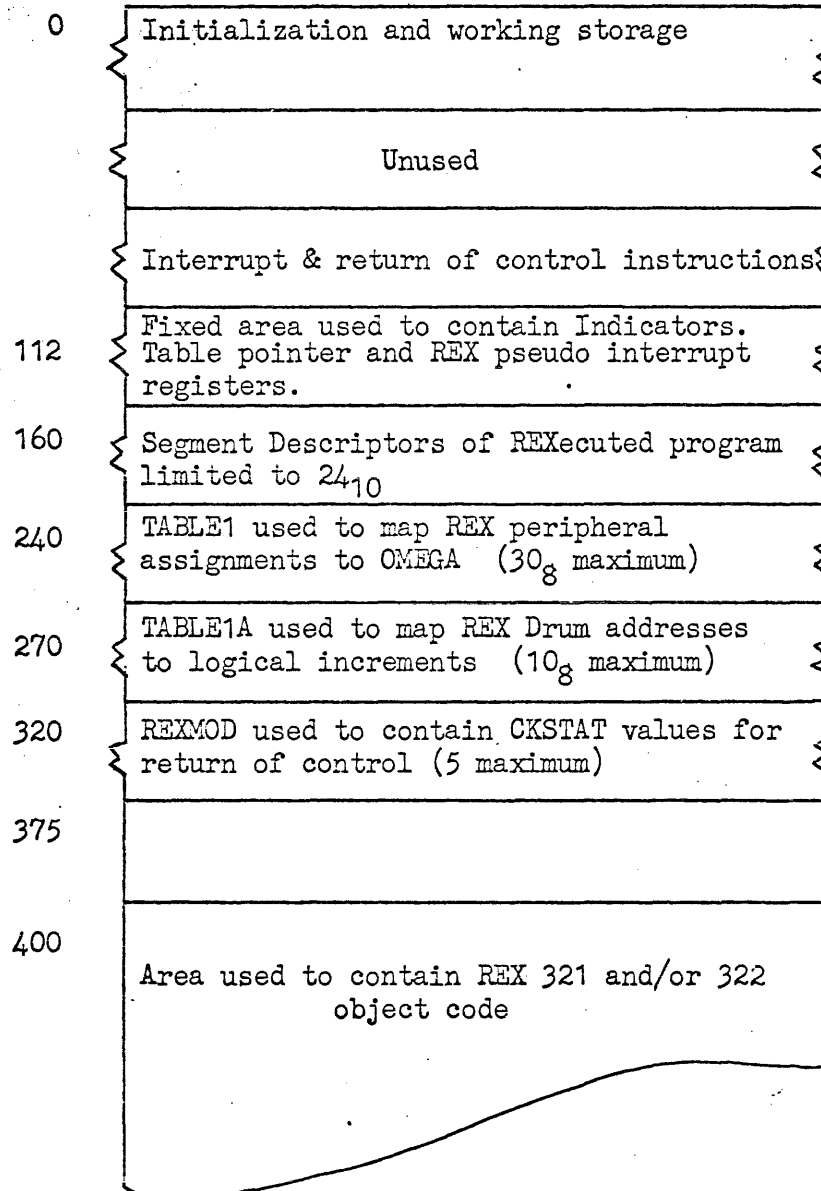
NOTE: OMEGA status codes  
 $\emptyset$ , 7, 10 and 11 are  
 converted directly.

A REX code of 00  
 from the Table will  
 cause the program to  
 be aborted.

### 9.6.8 Execution Area

The Execution area is allocated with each program to be placed under control of the REXecutor. This area is used to contain tables, interrupt instructions, and working storage required to control the REXecuted program. The following map shows general layout of execution area. In general all addresses contained are relative to word 0 of execution area.

Word



Words 0 -

Initialization and Working Storage - This area is initially used to activate the REXecutor and contains instructions to perform the following:

- .. Register queue process Activity 1 used to process instructions of the REXecuted program.
- . Register queue process Activity 2 used to reformat and execute Input/Output calls and REX Loader functions.
- . Establish contingency points for Fault, memory guard violation and privileged instructions.

Once REXecuted program is activated area will be used for working storage to interpretively execute Input/Output calls.

Words 111

Contains instructions required to accept calls to REX pseudo interrupt registers (SILRJP\*U/L (137-157) and calls REXecuter for processing. In general upon REX call P value is saved and a call to Phase 1 or Phase 2 of REXecutor is made via EXRN for processing.

These instructions may be extended to process additional calls peculiar to installation; such as calls to User File Control packages, etc.

Fixed portions of Execution area used to control and return program control during execution of REXecuted program. In general, this area may not be modified with change to REXecutor.

Format:

112	BEGINNING LOGICAL ADDRESS OF SEGMENT ERRATA			
113	ENDING LOGICAL ADDRESS OF SEGMENT ERRATA			
114	E.I.R. ADDRESS	PROGRAM BASE		
115	ZG	SEGMENT LENGTH	} USED FOR SEGMENT LOAD	
116		SEGMENT BASE		
117	LOGICAL Δ OF SEGMENT			
120	EXRN	PHASE 1 FC 07	→ FAULT	
121	EXRN	PHASE 1 FC 10	→ PLR VIOLATION	
122	EXRN	PHASE 1 FC 11	→ PRIVILEGED INSTRU.	
123	# OF DESCRIPTORS	Δ SEG. DESCRIPTORS	} POINTERS TO TABLES	
124	# OF ENTRIES	Δ TABLE 1		
125	# OF ENTRIES	Δ TABLE 1A		
126	# OF REXMOD	Δ REXMOD		
127	EXRN*B3	∅		
130	B7 ON TYPEC CALL			
131	OPTIONS FROM REX & PROG CARD			
132	MREAD COUNT	TRANS COUNT		
133	UNUSED	TAKEOVER INDICATOR		
134	ABORT INDICATOR	DONE INDICATOR		
135	DATE			
136	ERROR	ERROR	} PSEUDO REX REGISTERS	
137	ERROR	ERROR		
140	STANDARD I/O	CKSTAT		
141	TAKEOVER	TYPEC		
142	CONSOLE & EXIT	ERROR		
143	ERROR	ERROR		
144	ERROR	REX LOADER		
145	ERROR	ERROR		
146	ERROR	ADDRESS OF DATE		
147	TIME OF DAY			
150	ON DISTRIBUTED VERSION THEY ALL POINT TO ERROR PROCESSING			} PSEUDO INTERRUPT REGISTERS RESERVED BY SOME USERS
157				

Description of entries in fixed area.

Word	Description
112-113	Beginning and ending random storage logical increments containing segment errata. Each errata Enter on random storage is composed of two words as formed and stored by during initialization of Phase 1.

SEG #	ADDRESS OF WORD
ERRATA WORD	

<u>Word</u>	<u>Description</u>
114	Upper contains address of 321 or 322 program containing P option when loaded for REXecution (main program). Lower contains Base Address for REX program area.
115-117	Contains a random access packet used to load program segments of a 322 program.
120-122	Contains calls to Phase 2 upon occurrence of contingency violations. Fault, Memory Guard and privileged instructions.
123-126	Contains increments and number of entries in each of the tables or lists. These can be varied to reflect installation requirements.
127	A general EXRN used to call for OMEGA function from Execution area, ABORT, ERROR, RETURN, etc., which cannot be performed within the secondary Exec.
130	Used to hold B7 when a TYPEC REX service request is made.
131	Contains options from "REX" and PS control cards. Options A through Z are reflected by the corresponding bit 0 through 26 respectively being set.
132	Used to contain counts for card reader functions during execution of REX card service requests.
133	Lower contains indicator that a REX TAKEOVER was given by REXecuted program.
134	Upper contains indicator to REXecutor program is being abnormally terminated. Lower contains indicator that program is being terminated due to STOPRUN or TERMRUN.
135	Contains today's date by REX conventions.
136-146	Normally REX service registers error positions may be used to call abnormal user functions.
147	Contains Time of Day REXecuted program was activated.
150-157	Reserved for user service calls peculiar to installation.

Words 160-237

Contains segment descriptors of a 322 program loaded for REXecution and are ordered by segment number. Each descriptor is composed of two words as follows:

Word			
0	<table border="1"><tr><td>I/ 0</td><td>LENGTH OF SEGMENT</td></tr></table>	I/ 0	LENGTH OF SEGMENT
I/ 0	LENGTH OF SEGMENT		
1	LOGICAL INCREMENT OF SEGMENT		

Word 0 - contains length of segment in words determined at load time.  $2^{29}$ th set to (1) implies segment errata is present on mass storage.

Word 1 - contains logical increment of segment on random access storage relative to file code ZG.

Each segment descriptor entered in list is in ascending order by segment number. Address of called segment is determined by

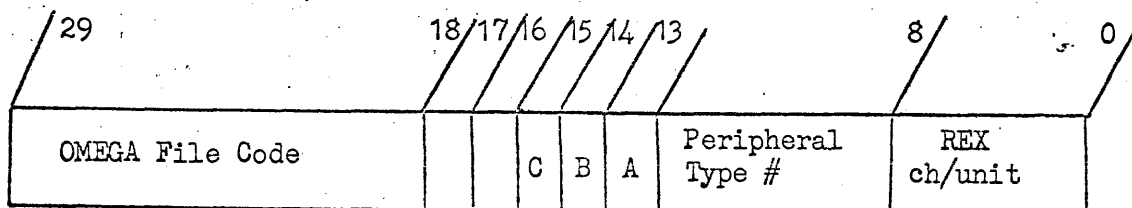
$$L(\text{word } 123) + 2(\text{segment } \#)$$

Once obtained REXecutor will load Load segment and errata.

Words 240-267 Table 1 (MEANS Table)

**Purpose:** Used to contain user supplied parameters to effect the interpretation of a REX I/O packet at execution time. Table 1 is formed from MEANS statements during initialization of Phase 1 of REXecutor and is contained in the execution area of the program.

**Format:** Table 1 is of variable length dependent upon the number of MEANS statement. Each entry within Table 1 is of the following format:



Where -ch/unit number supplied by User via MEANS control card. Unit and/or channel value specified on MEANS card must match that contained in I/O packet at execution time.

Peripheral type number for which the original REX I/O packet was coded for. This number is implied by the mnemonic used on MEANS card.

- |           |            |                       |
|-----------|------------|-----------------------|
| 1 = FH880 | 6 = CRIN   | 13 = Primary Input    |
| 2 = FAST  | 7 = CROUT  | 14 = Primary Output   |
| 3 = UN2A  | 10 = PRINT | 15 = Secondary Output |
| 4 = UN3C  | 11 = PTIN  | 16-37 = Unassigned    |
| 5 = UN3A  | 12 = PTOUT |                       |

A set to (1) implies channel has units; search must be made on both channel and unit. The only peripheral types recognized requiring unit search are the tapes peripheral type 3, 4 and 5.

B set to (1) indicates unit is mass storage device requiring use of Table 1A to map drum address to logical increment and determine file code.

C set to (1) indicates channel/unit are to be used to satisfy an internal facility request, REX packet. Once used to satisfy internal request (C) will be cleared and entry will function in the normal manner for subsequent I/O packets.

File Code - OMEGA file code to which reformatted I/O request is to be submitted.

Words 270-317

TABLE 1A (Drum Area Descriptors)

**Purpose:** Used to contain User supplied parameters to effect the transition of Drum address presented via REX calls to logical increment used by OMEGA Table 1A is formed from MEANS statements during initialization, Phase 1 of REXecutor, and is contained in execution area of REXecuted program.

**Format:** Table 1A is of variable length dependent upon number of MEANS statements describing mass storage were submitted. Each entry in the table is composed of three words in the following format:

Word

0	OMEGA FILE CODE	PERIPHERAL TYPE
1	BEGINNING DRUM ADDRESS	
2	ENDING DRUM ADDRESS	

**Where-** OMEGA File Code is where equivalent mass storage file is assigned to task.

Peripheral Type contains the peripheral type number of mass storage for which the REXecuted program was coded. 2 implies FASTRAND in which after a find is made conversion is as follows:

$$33_{10}(\text{Drum Address} - \text{Word 1}) = \text{Logical Increment}$$

1 implies FH-880 in which the conversion is as follows:

$$(\text{Drum Address} - \text{Word 1}) = \text{Logical Increment}$$

Beginning Drum Address is the base address (word or sector) which the REXecuted program assumes as the base of its mass storage file.

Ending Drum Address is the number of words or sectors contained in the original file plus Beginning Drum Address.



Words 320-374 REXMODS

A REXMOD are pseudo REX storage modules used to contain program registers and status for return of control following the completion of REX service request. Each REXMOD is of the following format:

Word

Ø	STATUS	B1
1	P	B2
2	RETURN POINT	B3
3	NORMAL RETURN	B4
4	ERROR RETURN	B5
5	P-TYPE	B6
6		B7
7		A
10		Q

Status - is an indicator reflecting current status of request.

- 0 - REXMOD is currently unused
- 1 - Request has been queued to Activity 2, CKSTAT has not been given.
- 2 - Request is complete, or CKSTAT with E.A.S. given.
- 3 - Request complete and CKSTAT with E.A.S. given eligible for return of control upon execution of TAKEOVER.
- 5 - Request queued to Activity 2 or CKSTAT and wait given.
- 6 - Request complete and CKSTAT and wait given. Eligible for return of control.
- 77777 - REXMOD has been selected for return of program control and has been queued to Activity 2.

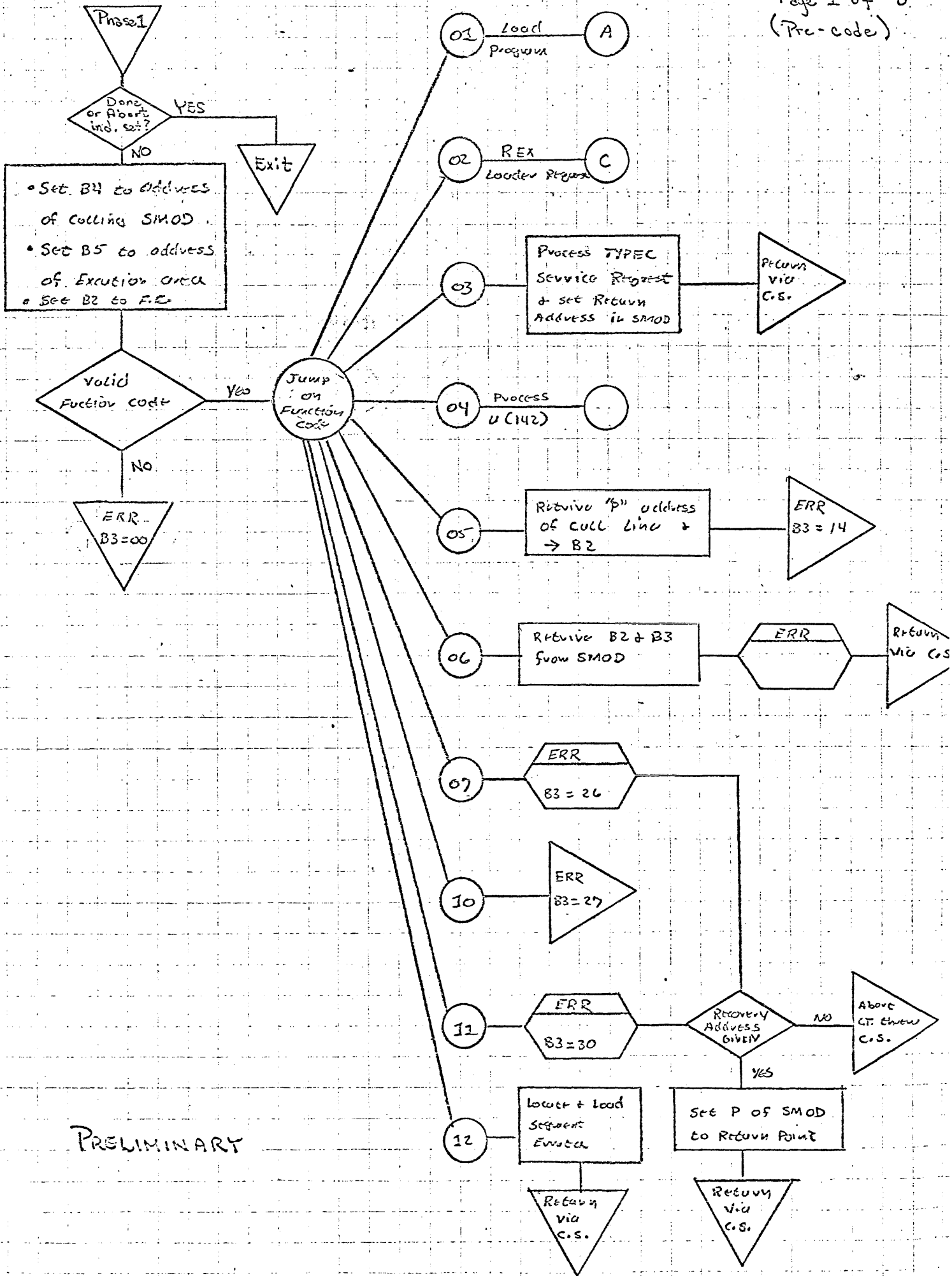
P contains packet or request address relative to RIR of activity.

Return Point - address relative to RIR to which program control is to be returned.

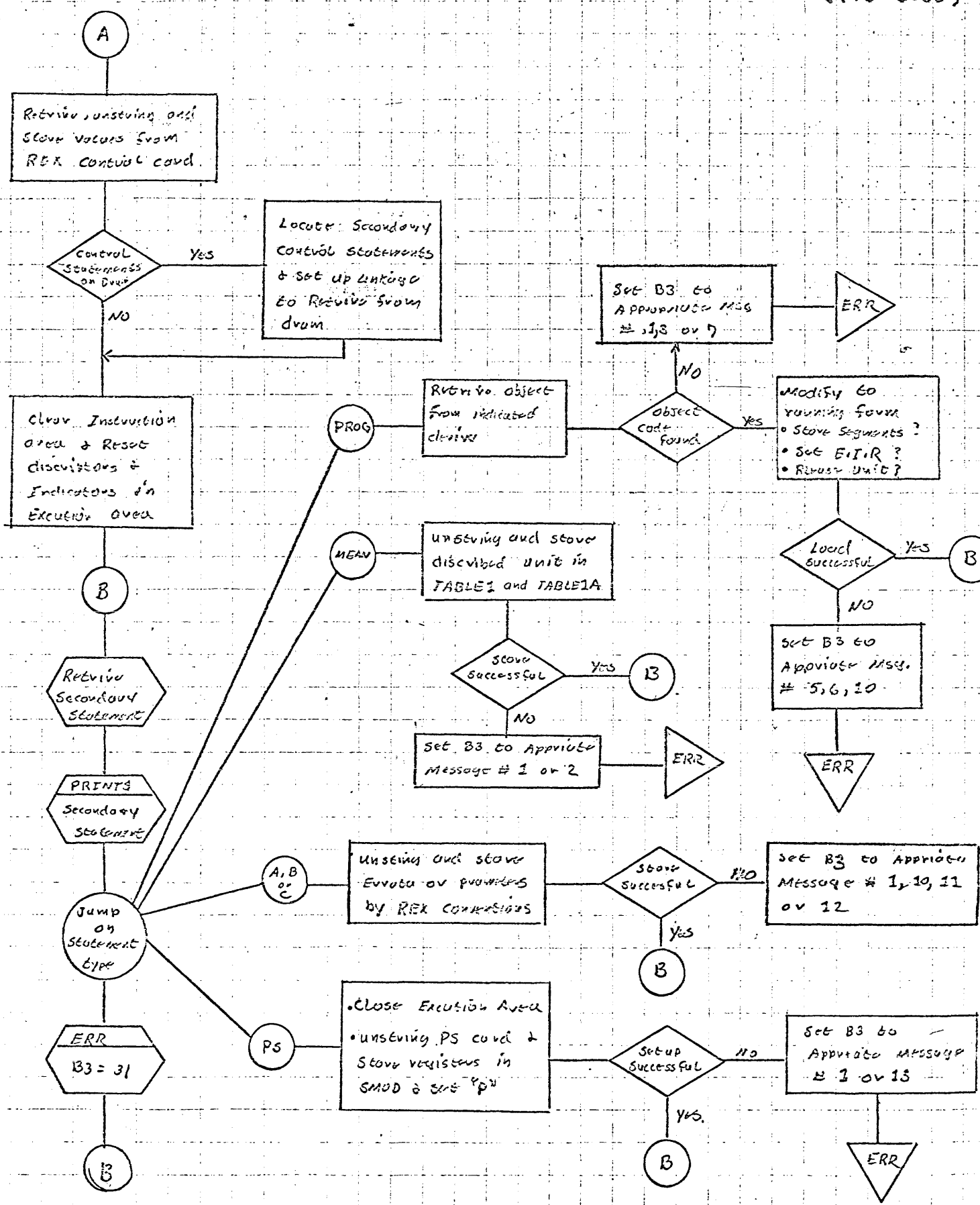
Normal Return - address relative to RIR, to which program control is eligible if request completed without error.

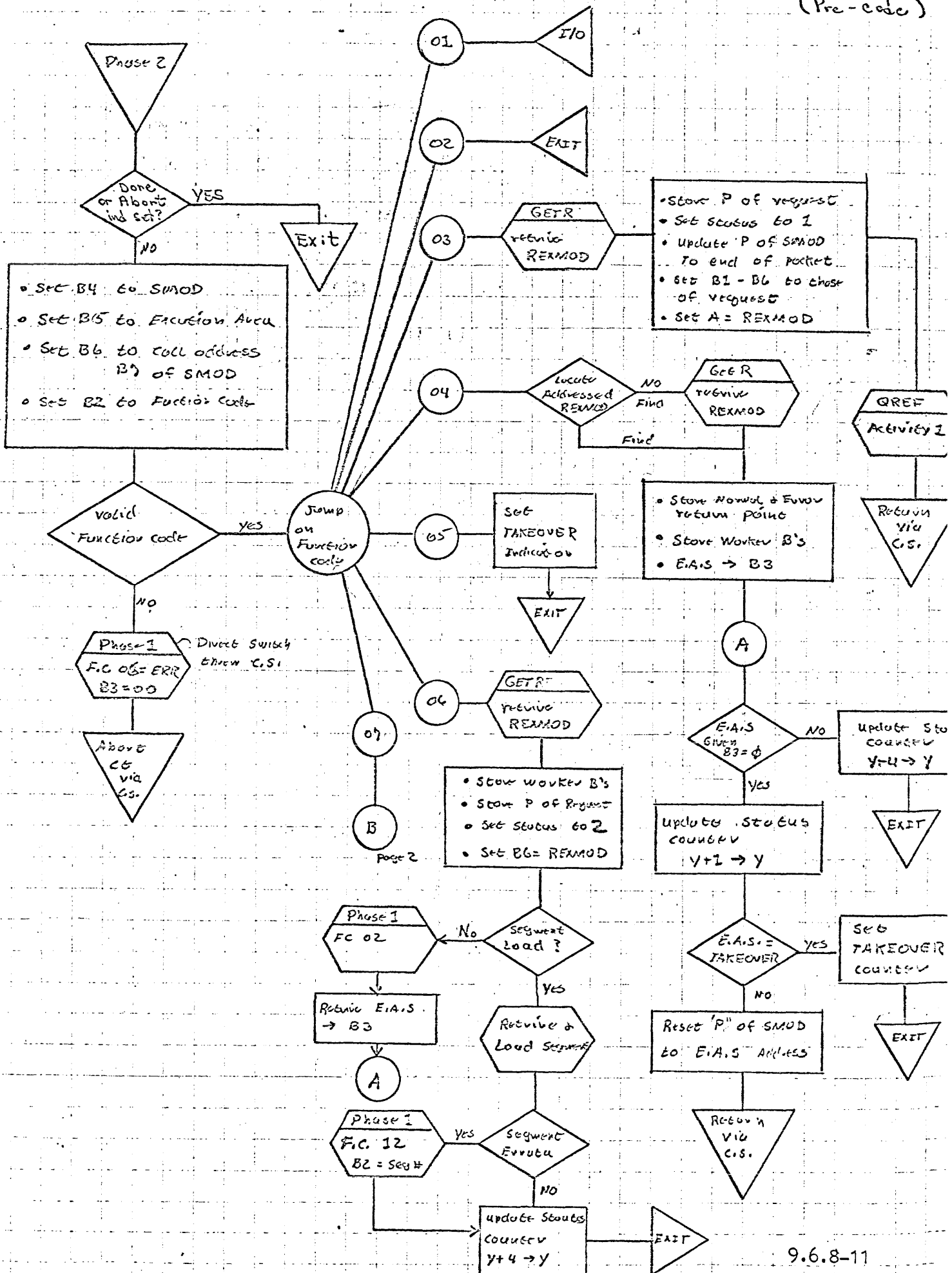
Error Return - address relative to RIR, to which program control is eligible if request completed in error. 1 indicates STOPRUN which will cause termination of program.

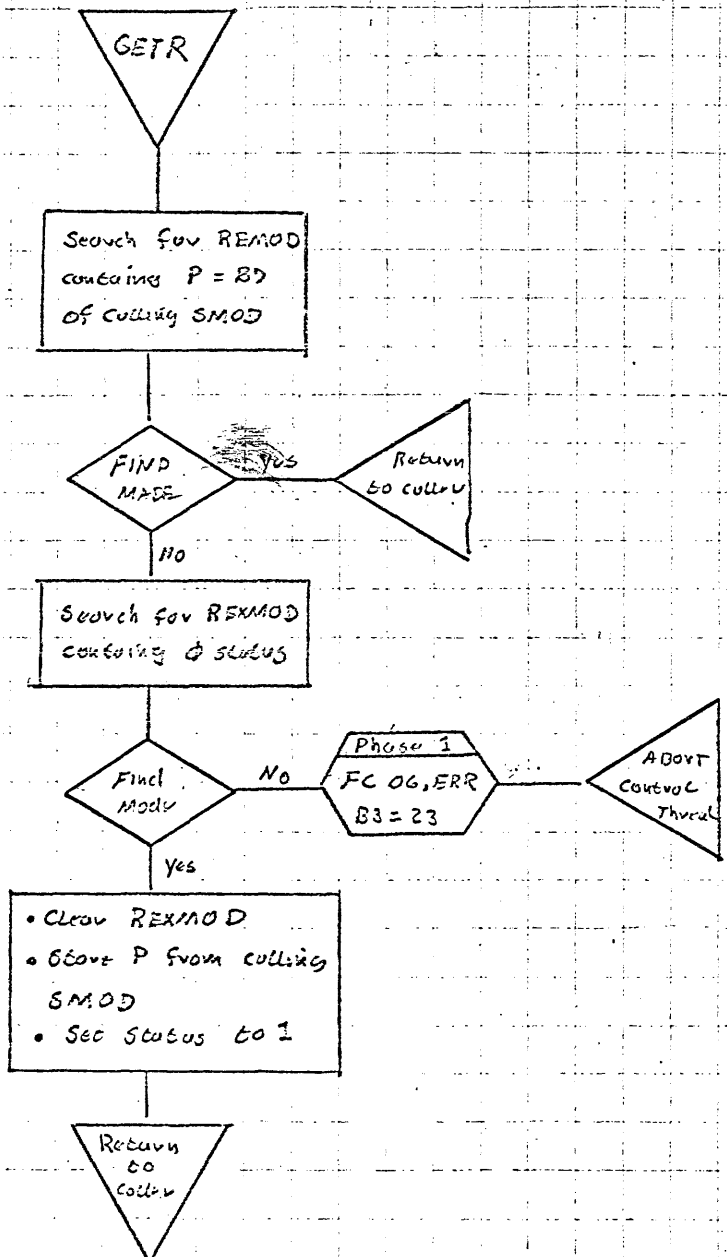
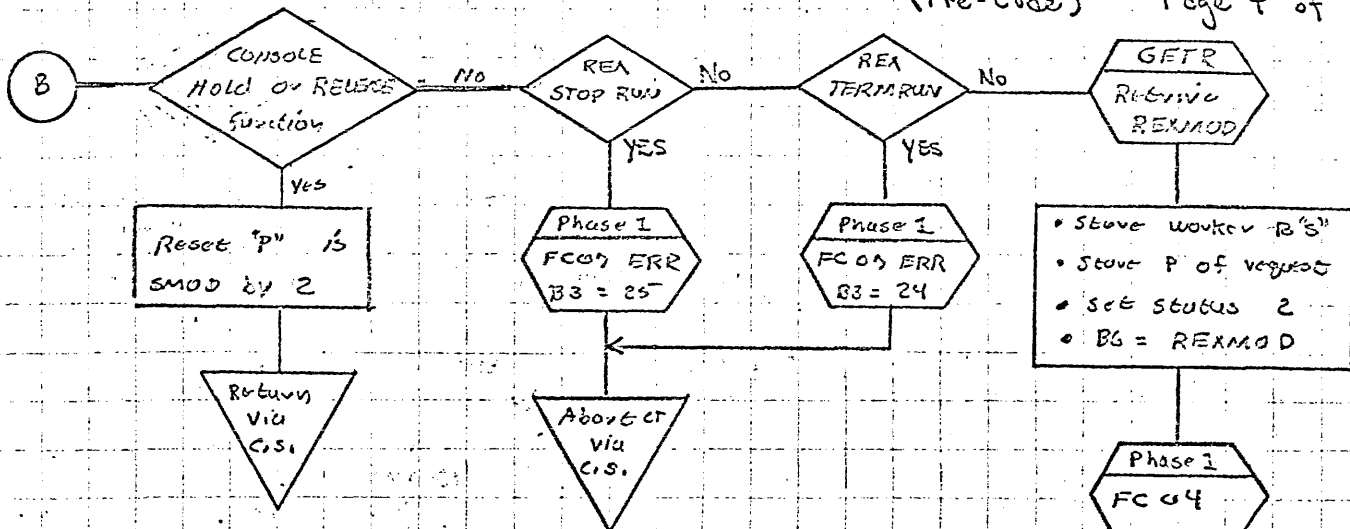
B1-B7, A & Q are register settings used to return control to requestor.



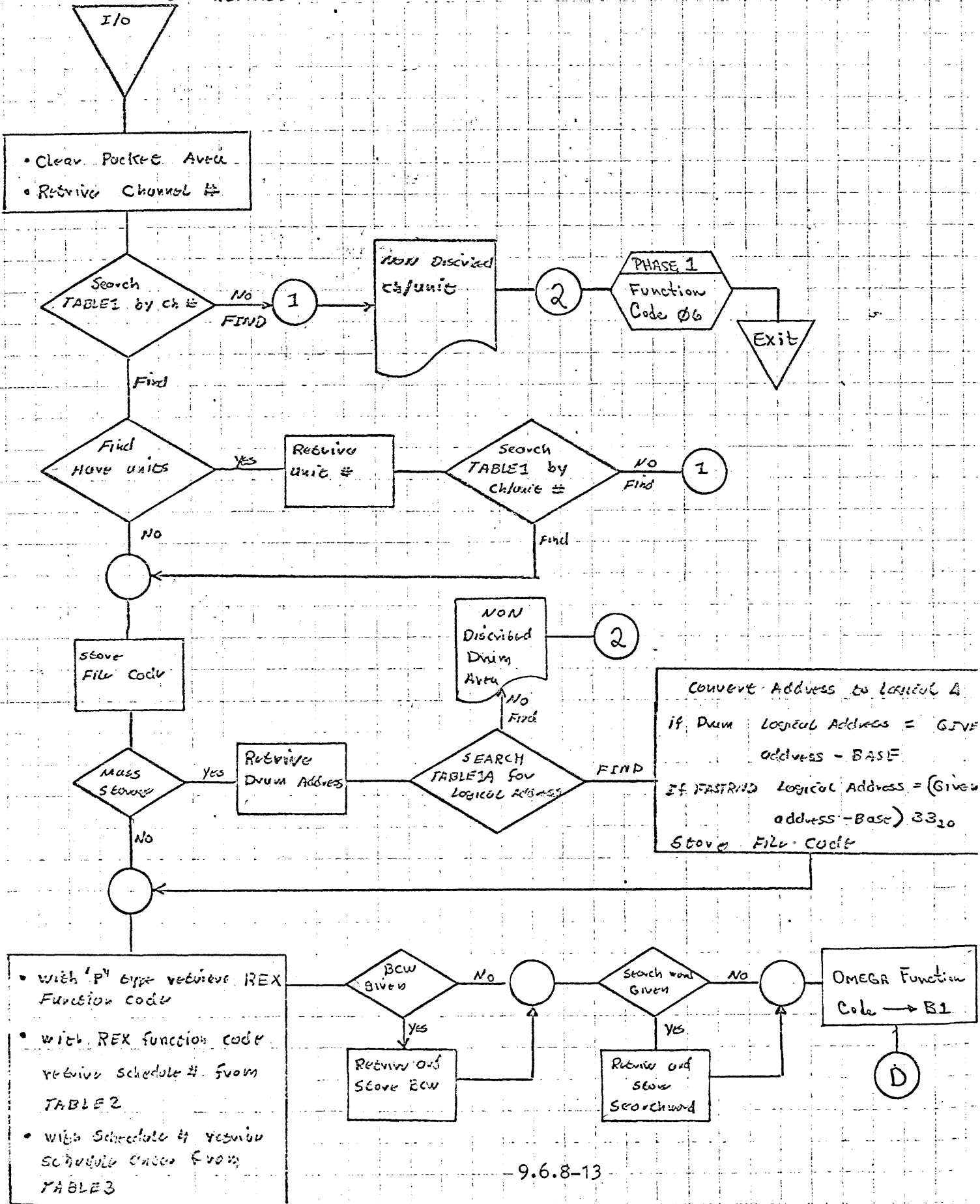
PRELIMINARY

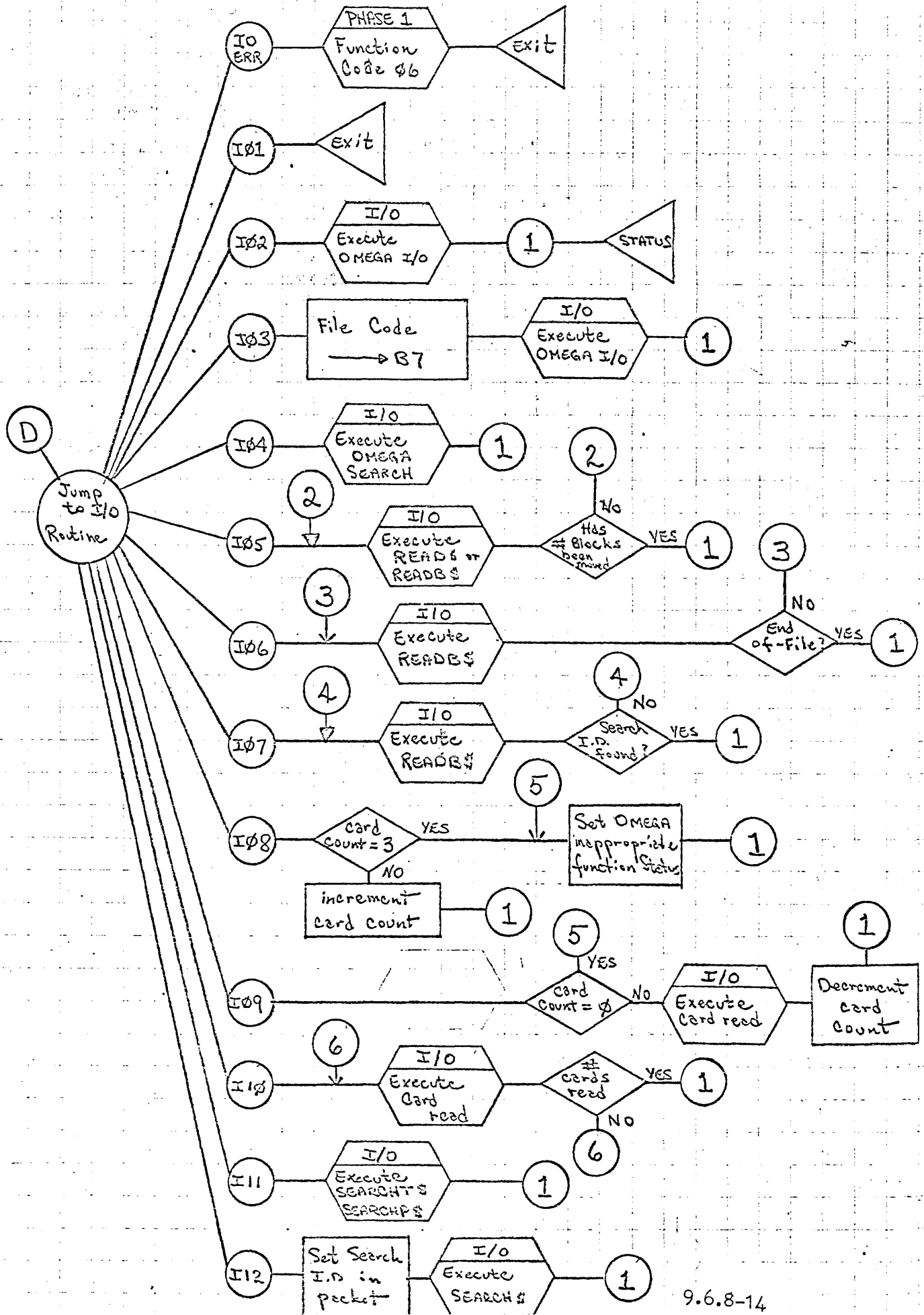


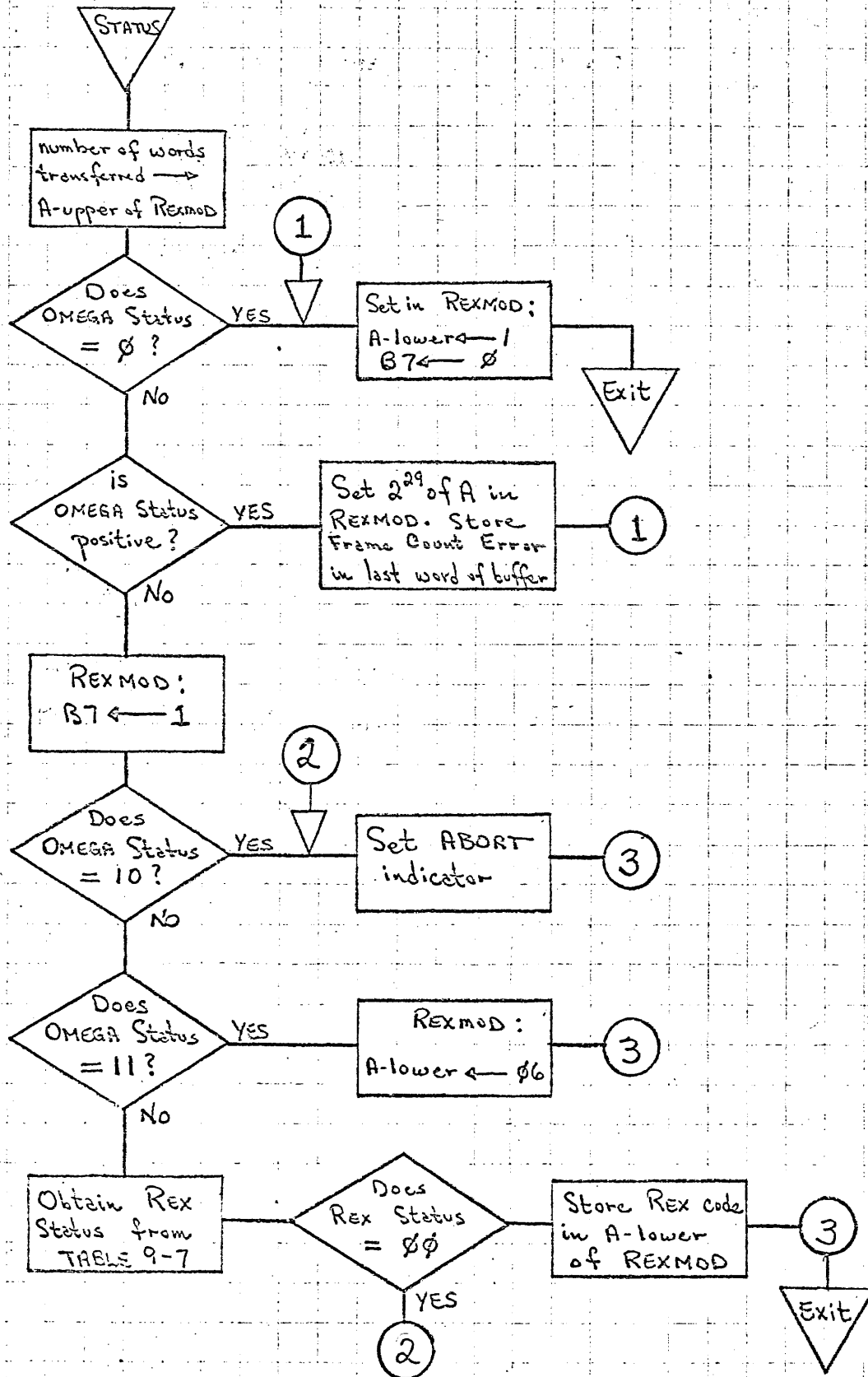




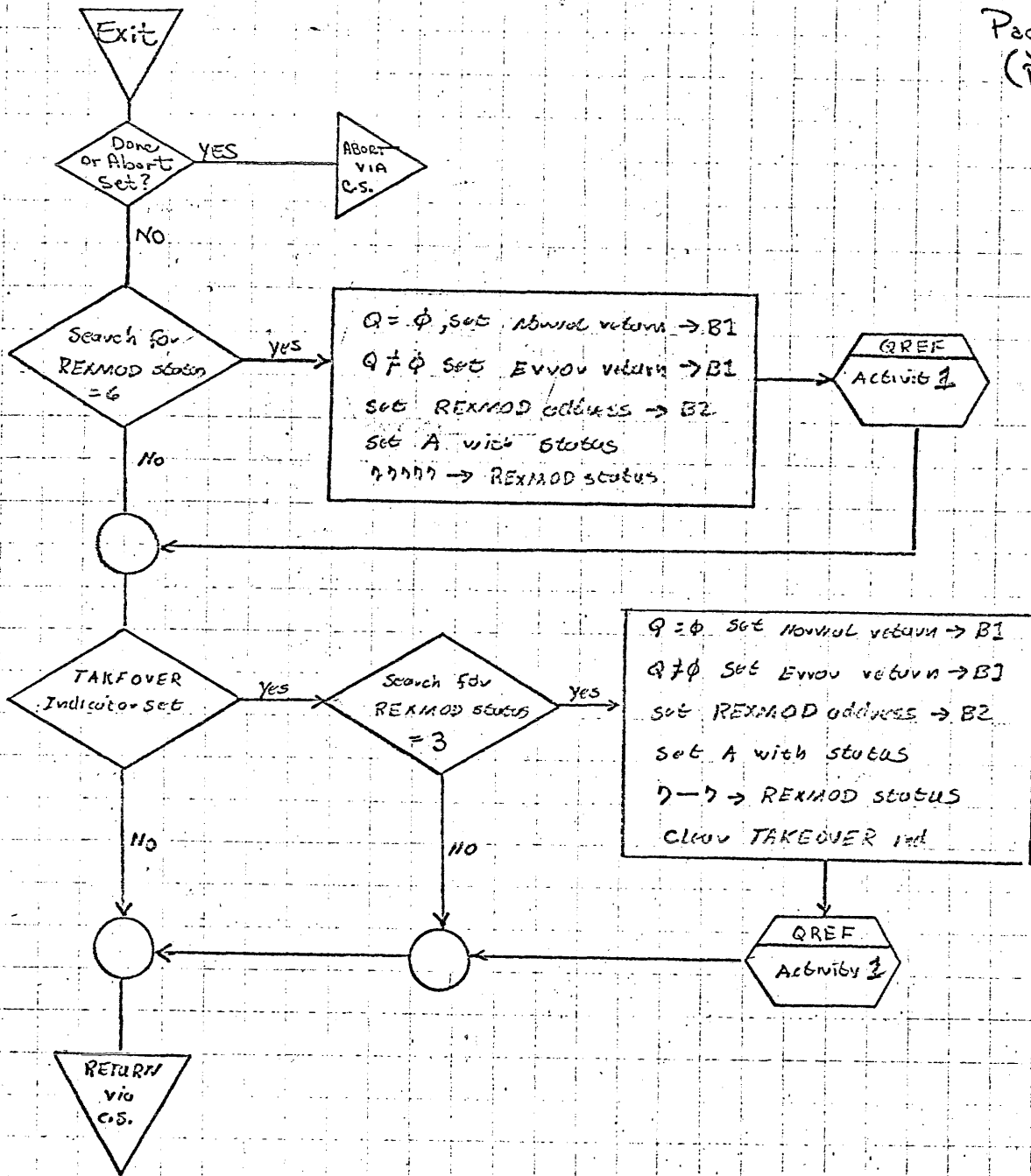
Entered with  
B4 = SMOD  
B5 = execution mode  
B6 = REXMOD











D. List of MACROS simulating 494 instructions on the 490.

These are the 77 XX instructions on the 494 and are illegal instructions on the 490. All macros are constructed so that coding will be identical to that on the 494, e.g., a conditional jump instruction preceding a macro will jump around the macro if the skip condition is satisfied without any special coding requirements for the programmer.

77 21	DPENT	Double precision enter. (Y)→A, (Y+1) → Q.
77 23	DPCME	Compare Y and Y+1 to A and Q respectively. Skip next instruction if equality found.
77 24	DPCP	Double precision complement. (A)' → A (Q)' → Q
77 25	DPSTR	Double precision store. (A)→Y, (Q)→Y+1
77 4x	EBJP·Bx	The P value is entered in Bx and jump performed to Y.
77 52	TSET	Tests bit 14 of Y; if not set it is set and bits 13-00 are cleared. If set jump to TSETENTRY subroutine. Packet is saved within the macro.
77 54	EXRN	The macro sets up a return jump to TEXECENTRY location is simulated OMEGA. The lower half of the EXEC return instruction must be provided as a parameter, e.g. for 77540 00002 the 00002 must be supplied as the parameter in the coding.

The following instructions can only be used by the executive so a test is performed on L(GARDMODIND) to determine if in executive mode (if=0, it is in exec mode). If it is in exec mode the instruction is legal and can be executed; if not, the macro contains a return jump to error location. All instructions above 77 60 are exec only instructions.

77 61	EIFR	Test new IFR to see if it indicates exec or worker B-registers are to be used. Test old IFR to see which set is active. If wrong set is active store that set and enter the other set. Then IFR is entered with Y value and RIR is entered with (Y+1).
77 62	EPLR	Enter Program Lock register with value in Y.
77 65	SIFR	Store value in Internal Function Register in address specified by Y.
77 66	ERIR	Enter Relative Index Register with value in Y.
77 71	EWB	Enter worker B registers; (Y)→B1, (Y+1)→B2, (Y+2)→B3, etc.
77 72	STRC	Store Channel number. Test where channel number is and store it in Y address. The channel number could be in CSR or IASREG.
77 73	ECSR	Store lower five bits of Y address in lower of Channel Select Register.
77 75	SWB	Store worker B registers (B1)→Y, (B2)→Y+1, etc.

All assemblies for Omega on 490 simulation should include these macros in the card deck to facilitate familiarity with the 494 coding repertoire.