GY27-7255-0
File No. S370-36

Systems

OS/VS2
HASP II Version 4
Logic

Program Number 370H—TX—001

VS2 SVS Release 1.7

IBM

First Edition (March, 1973)

This edition, as amended by technical newsletters SN25-0122 and SN27-1555, applies to HASP II Version 4.1 in support of OS/VS2, SVS Release 1.7 and any subsequent versions of HASP and releases of SVS unless otherwise indicated in new editions or technical newsletters.

Information in this publication is subject to significant change. Before using this publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office serving your locality.

Forms for reader's comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150.

PREFACE

Persons interested in determining sources of errors within or making changes to the internal logic of HASP II Version 4 should read this publication. Readers must be familiar with programming techniques and the operating principles of OS/VS2 Release 1.

This manual describes the purpose and function of HASP and its relationship to OS/VS2 Release 1. It does not replace the program listings; it supplements them and makes the information in them more accessible.

This publication contains seven sections:

Section 1 Introduction - describes the general characteristics and functions of HASP II Version 4.

Section 2 Method of Operation - contains HIPO (Hierarchy plus Input-Process-Output) diagrams that describe the operation of HASP II Version 4. The diagrams are high level and are designed to guide the reader to a particular area of the program listing.

Section 3 Program Organization - describes the HASP general program organization and each of the HASP processors.

Section 4 Directory - provides cross-reference lists.

Section 5 Data Areas - contains descriptions of the interrelationship and content of HASP data areas and control blocks.

Section 6 Diagnostic Aids - contains information necessary for interpreting the program listing and diagnosing program failures.

Section 7 Appendix A HASP Programmer Macros - describes HASP macro instructions and their use.

Glossary - defines HASP terms.

Related OS/VS publications are listed in the IBM System/360 and System/370 Bibliography GA22-6822.

Table of Contents

SECTION 1

HASP

INTRODUCTION

HASP INTRODUCTION

HASP II Version 4 is not System Control Programming (SCP). It is optionally available to replace OS/VS2 Release 1 readers and writers. It provides the remote job entry (RJE) support for OS/VS2. Installation remains the responsibility of the user. Programming Service Classification is A.

The HASP System is an extension of OS/VS2 and provides support in the areas of job management, data management, task management, and remote job entry. HASP operates as a systems task and is formally interfaced to OS/VS2. When HASP is used, it supplants the normal OS/VS2 functions of reader, printer, and punch input/output services; SYSIN/SYSOUT SPOOLing; and job scheduling.

Features that may add to system performance are a high performing SPOOL Management routine and the HASP MULTI-LEAVING Line Manager. MULTI-LEAVING is employed with all CPU work stations and will tend to maximize line effectiveness and provide concurrent operation of all supported work station devices.

HASP operation is in a V = V mode. The minimal storage that must be fixed is 12K bytes. The requirement for fixed storage will be approximately 25% or less of the total storage generated for a HASP System.

The job input and output services provided for local peripheral devices, along with a subset of the HASP operator commands capability, are optionally extended to remote work stations, including both CPU and non-CPU terminals. Work station programs for 2922, BSC System/360 Model 20 and higher, BSC 1130, and System/3 are generated as extensions to the central HASP System and operate in the work station on a "stand-alone" basis. The HASP RJE implementation for BSC CPU work stations is based on the HASP MULTI-LEAVING philosophy which provides the capability for concurrent operation for all supported terminal job input, output, and console devices.

HASP is a specialized program that operates in the same CPU with OS/VS2 to perform the peripheral functions associated with batch job processing.

HASP is loaded as a systems task. Control of all online unit record devices is assumed; the designated intermediate storage direct-access devices(s) are initialized; and job processing begins.

HASP has three major processing stages which relate to its three major external functions. These are:

1. INPUT STAGE - This stage reads jobs simultaneously from an essentially unlimited number of various types of online card readers, Internal Reader interfaces, and remote terminals into the system. These jobs are then entered into a priority queue by job class to await processing by the next stage.

2. EXECUTION STAGE - This stage removes jobs, based on priority and class, from the queue established by the input stage and passes

those jobs to OS/VS2 for processing. Input cards are supplied as required to the executing program, and print and punch records are received and written onto HASP intermediate storage. This stage can simultaneously control an essentially unlimited number of jobs being processed by OS/VS2. At the completion of a job, it is placed in a queue to await processing by the next stage.

3. OUTPUT STAGE - One purpose of this stage is to transcribe the printed output generated by jobs in the previous stage to printers. An essentially unlimited number of various types of printers and remote terminals can be operated simultaneously.

   The output stage also transcribes the punch output generated by jobs in the execution phase to punches. An essentially unlimited number of various types of punches and remote terminals can be operated simultaneously.

All of these processes are controlled by reenterable code, so no additional code is required to support multiple, simultaneous functions. Since all of the above functions can occur simultaneously and asynchronously, a continuous flow of jobs may pass through the system.

The following paragraphs describe some of the more significant algorithms employed by HASP to improve function and performance.

SPECIALIZED DIRECT-ACCESS STORAGE ALLOCATION

HASP, through the use of an allocation bit map in main storage, dynamically allocates intermediate storage space for jobs on a record basis, within definable track groups. The use of this technique offers the following advantages:

1. Disk-arm motion and interference is minimized by dynamically allocating space, based on the position of the access mechanism.

2. Disk-area fragmentation is automatically eliminated by allocation of the smallest possible increment of space.

3. The data for a single data set can be spread across multiple direct-access volumes. In addition to further optimizing arm motion, this capability allows for the simultaneous use of multiple channels to increase the data rate for a given job.

4. Since space is allocated only when required, there will be no unused space as a result of over-estimated output requirements.

5. The release of previously-used space is accomplished by a simple algorithm, which requires no I/O operations.

UNIT RECORD DEVICE COMMAND CHAINING

While operating any reader, printer or punch, rather than handling each record separately, HASP constructs a chained sequence of channel command words to pass to the channel. Thus, instead of the overhead of the EXCP and the ensuing interrupts for each record transmitted, only one EXCP and associated interrupt is required for a series of records. For example, when reading a job into the system, HASP might chain 40 commands together to instruct a card reader. This would cause the next 40 cards to be read into storage without requiring the execution of any CPU instructions.

TRANSPARENT BLOCKING

All input, print, and punch for every job is automatically blocked by HASP to improve performance. Since all deblocking is also done by HASP, any program, even if designed to operate with unblocked records, can benefit from the blocking. Also, because all blocking and deblocking is done by HASP, problem programs require buffers only the size of a single card or line. This can reduce a program's partition or region requirement by several thousand bytes over normal full-track blocking.

DYNAMIC BUFFER POOL

HASP maintains a dynamic area of storage, which is allocated as required. This technique allows not only multiple data sets of a job but multiple jobs to share this area, thereby ensuring optimum use of storage.

FEATURES

Standard Features

The standard features of HASP are as follows:

1. Job input service provides for low overhead reading of job streams and storing of data on SPOOL volumes for later high-speed retrieval for up to 99 concurrently-active local card readers in any combination of devices as follows (one required): 2540 reader, 2501 reader, and 3505 reader (80-column punched cards only).

2. Execution services provides for selection of jobs and execution monitoring for up to 63 concurrently-executing jobs with services as follows: selection of jobs based on job class and initiator priority class (list of up to 64 classes for each initiator); automatic delaying of jobs with duplicate OS job names; automatic deblocking and blocking of user SYSIN/SYSOUT data using the HASP dynamically shared buffer pool count of lines, cards, and execution duration with optional operator notification and/or job cancellation; and interface for SMF counting of SYSIN/SYSOUT data.

3. Execution services requires an OS Reader/Interpreter to be active at all times.

4. Multiple SPOOL volume support provides for balanced utilization of up to 36 volumes for any combination of devices as follows (one required): 2305, 2314, 3330 and 3350.

5. Warm start capability provides for checkpointing critical HASP information sufficient for: optionally restarting jobs which were executing, restarting print at the last checkpoint, and restarting punch at the beginning of data set.

6. Job output print service provides for low overhead printing of job stream system message and user data print output for up to 99 concurrently-active local printers in any combination of devices as follows (one required): 1403 Printer, 3211 Printer, and 3800 Printing Subsystem.

7. Special forms feature provides for the routing of print (on a job or data set basis) and punch data (on a data set basis) to special forms output queues for output as directed by the operator.

8. Console support provides for direct entry for HASP commands and HASP abbreviated reply to WTOR through OS/VS2 operator consoles.

9. HASP minimal System Message Block (SMB) output writer provides for retrieval of SMB from the SYS1.SYSJOBQE data set.

10. HASP will interface directly with the OS/VS2 SMF Writer.

Optional Features

In addition to the standard features, the following optional features are available:

1. Internal Reader feature provides the ability for any nonswappable task within the system to submit jobs to HASP for batch execution as though entered from a HASP card reader.

2. Job output punching services provide for low overhead punching of job stream user punch output for up to 99 concurrently-active local punches in any combination of devices as follows: 2520 punch, 2540 punch, and 3525 punch.

3. Execution Batching feature provides the facility for passing jobs directly to a processing program such as a "one-step monitor," reducing the overhead of OS scheduling and allocation of facilities for short running jobs requiring limited system facilities.

4. Priority Aging feature provides for automatically increasing the HASP scheduling priority of jobs that have been in the system for extended periods of time.

5. Remote Job Entry feature provides for high-speed communications with BSC batch work stations which may be used for job stream input and output as well as operator control of the devices and jobs associated with the remote (see HASP Remote Job Entry for features).

HASP RJE Features

Those features common to all HASP RJE configurations are as follows:

1. HASP RJE supports up to 99 remote work stations communicating over leased (point-to-point) or dial lines.

2. HASP RJE provides for concurrent operations over up to 99 lines assigned to unique communication lines adapter addresses of the following types: SDA Type II on a 2701 for BSC, Synchronous Base on a 2703 for BSC, and 3705 providing 270X emulation.

3. Output routing control provides for print and punch output to be directed to the devices attached to the remote, to the central system, or to other remotes as designed by HASP generation parameters, by control card submitted with the job, or by operator command.

4. Remote Operator Control feature provides a subset of the HASP operator commands for display of information and control of jobs and devices associated with the remote.

5. Operator Message Output feature provides for immediate transmission of messages and responses to remote operators with online MULTI-LEAVING work stations with consoles and optional saving of messages for all other remotes until the remote is online and has its primary printer available.

6. Work station programs, when required, are supplied as extensions of HASP and are contained on the HASP distribution tape in source form.

7. Terminal support on the central system provides for communication with: 2770 (BSC), 2780 (BSC), 3780 (BSC), System/360 Models 20, 22, 25, 30, 40, 50, 65, 75, 85 and 195 (MULTI-LEAVING); 1130 (MULTI-LEAVING); System/3 Model 10 (MULTI-LEAVING); and 2922 (MULTI-LEAVING).

8. The sign-on feature provides for remote identification and line security through line passwords.

9. Remote characteristic support utilizes the unique features on each remote as follows: full text transparency (required for object decks), text compression, print line width, buffer size, and blocking capabilities. Note that multipoint or multidrop line features are prohibited.

10. Remote job priority adjustment provides for favoring or limiting the HASP scheduling priority of jobs submitted from each remote work station.

11. Line Restart feature provides for warm starting of print output after remote work station or line failures.

12. Line error recovery provides for continuous retry until successful transmission.

HASP MULTI-LEAVING RJE Feature

MULTI-LEAVING is a term which describes a computer-to-computer communication technique developed for use by the HASP System. In a gross sense, MULTI-LEAVING can be defined as the fully synchronized, pseudo-simultaneous, bi-directional transmission of a variable number of data streams between two or more computers utilizing binary synchronous communications facilities. Those features common to all HASP RJE configurations are provided to MULTI-LEAVING configurations with additional features as follows:

1. Concurrent device operation feature provides for all supported devices to operate concurrently in accordance with the device characteristics, line speed, and characteristics of the data streams.

2. Dual reader/punch device support provides for use as both reader and punch under automatic or operator control.

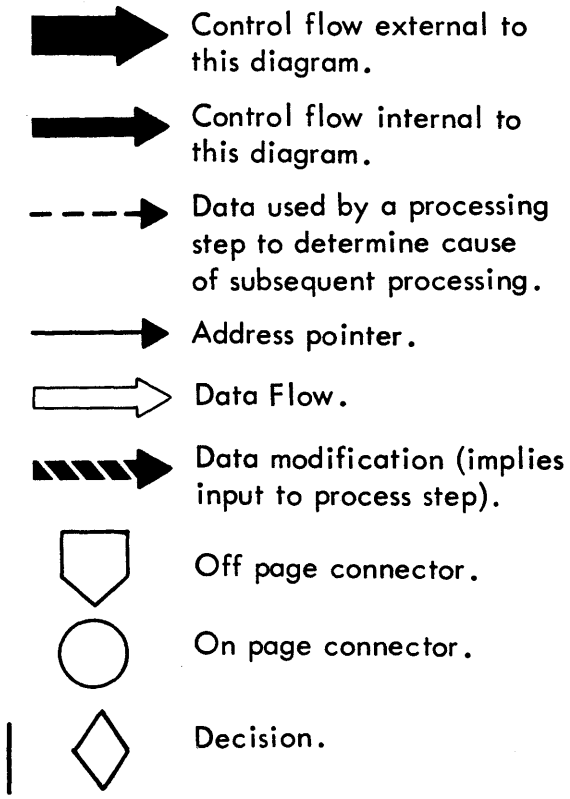
3. Unit record error recovery provides a minimum of operator intervention and continued operations using unaffected devices on operator console configurations.

SECTION 2

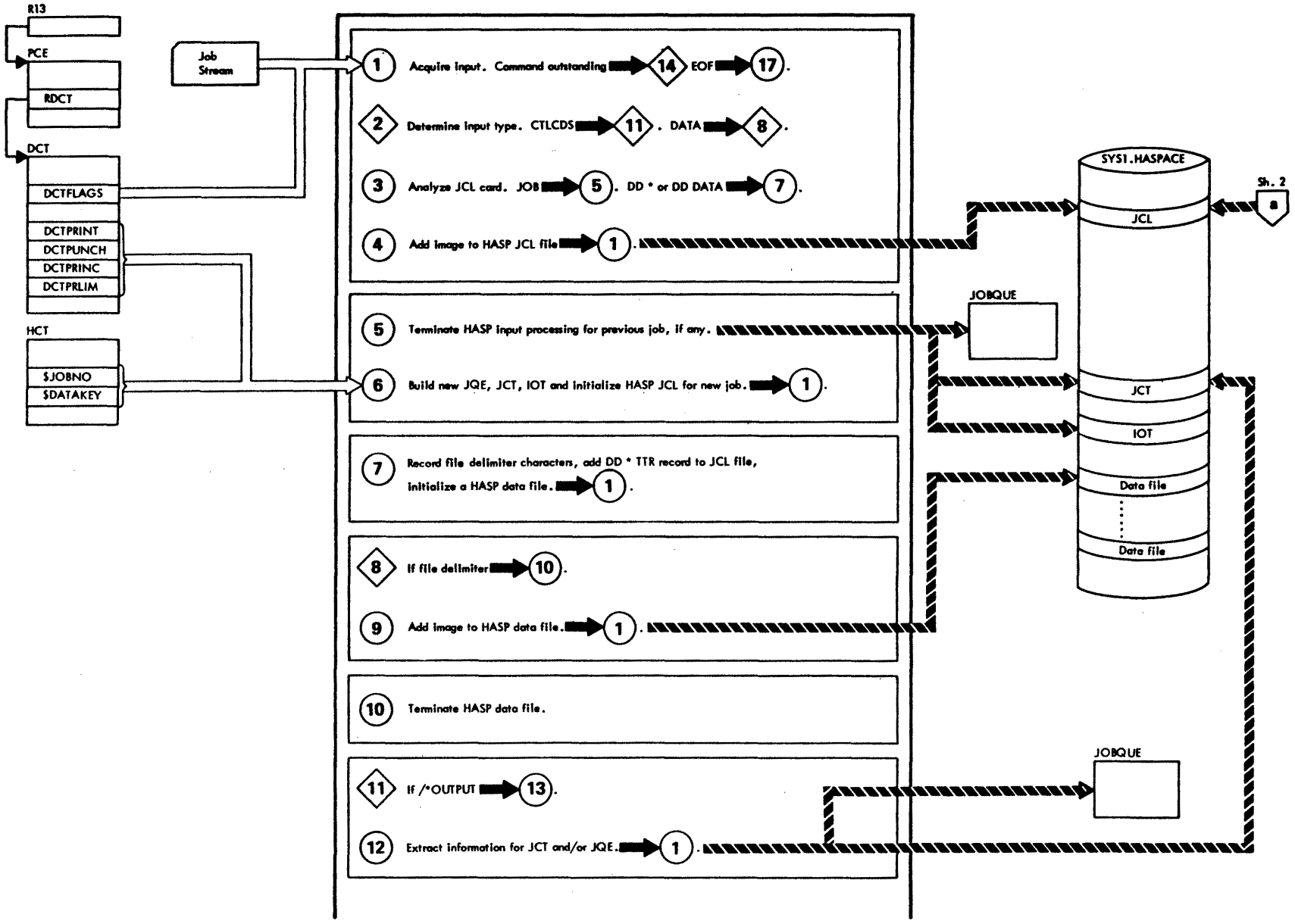
HASP

Method of Operation

LEGEND

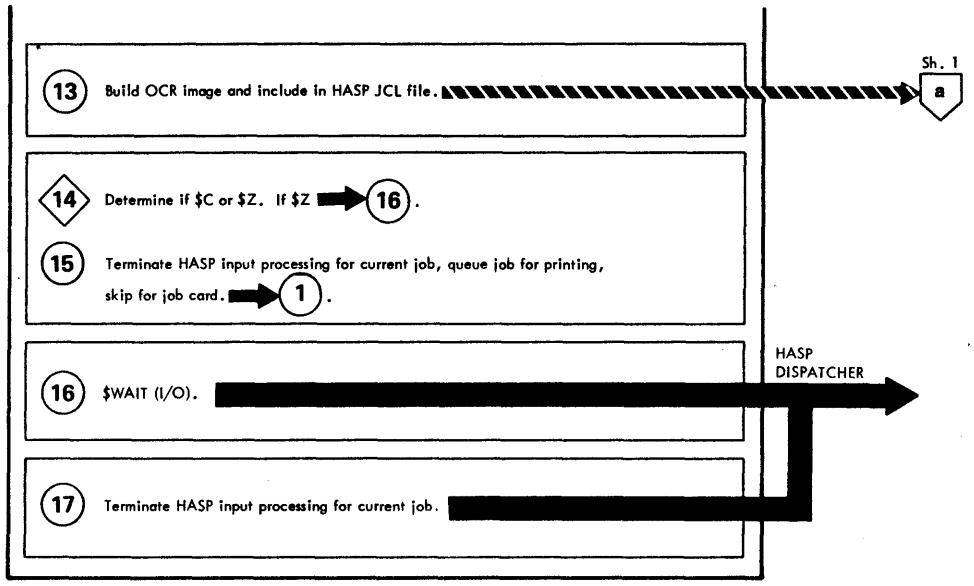


HASP Input Processor (1 of 2)



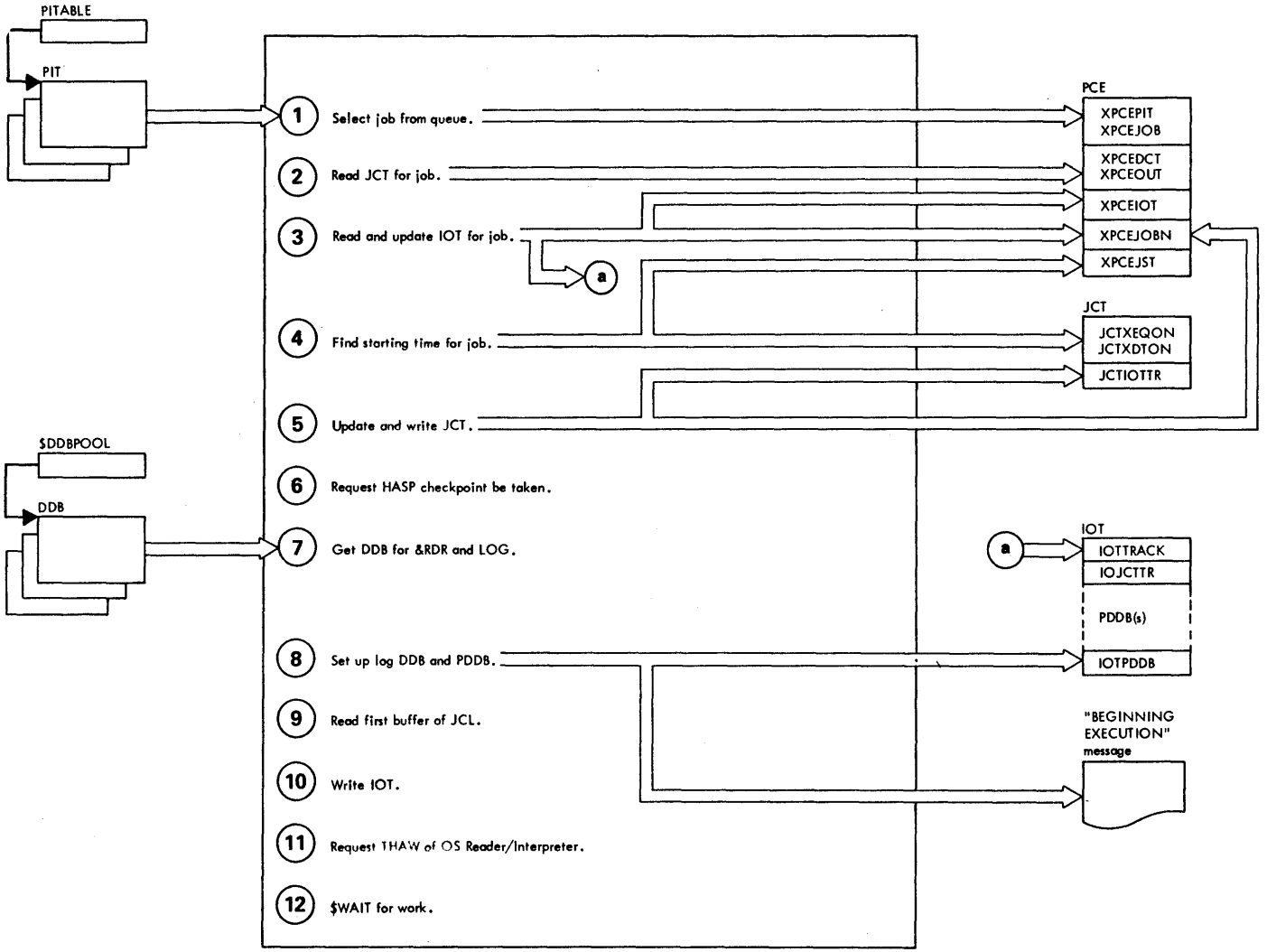
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPRDR	HASPRDR	1 Input is anticipated from a local card reader, the Internal Reader, or a remote work station.
HASPRDR	RCCSERCH	2 The table of control words at RCCTAB is searched for the various HASP control cards.
HASPRDR	RJCLCARD	3
HASPRDR	HASPRJCS	5 The Job Queue Management Service routines of $QLOC and $QADD are used to update the HASP Job Queue.
HASPRDR	RGENNEDD	7 A record is written into the JCL file to record the track address of where the data will be written.
HASPRDR	RFLTEST	9 If flush switch is not on put data record in HASP data file.
HASPRDR	HASPRCC2	12

HASP Input Processor (2 of 2)



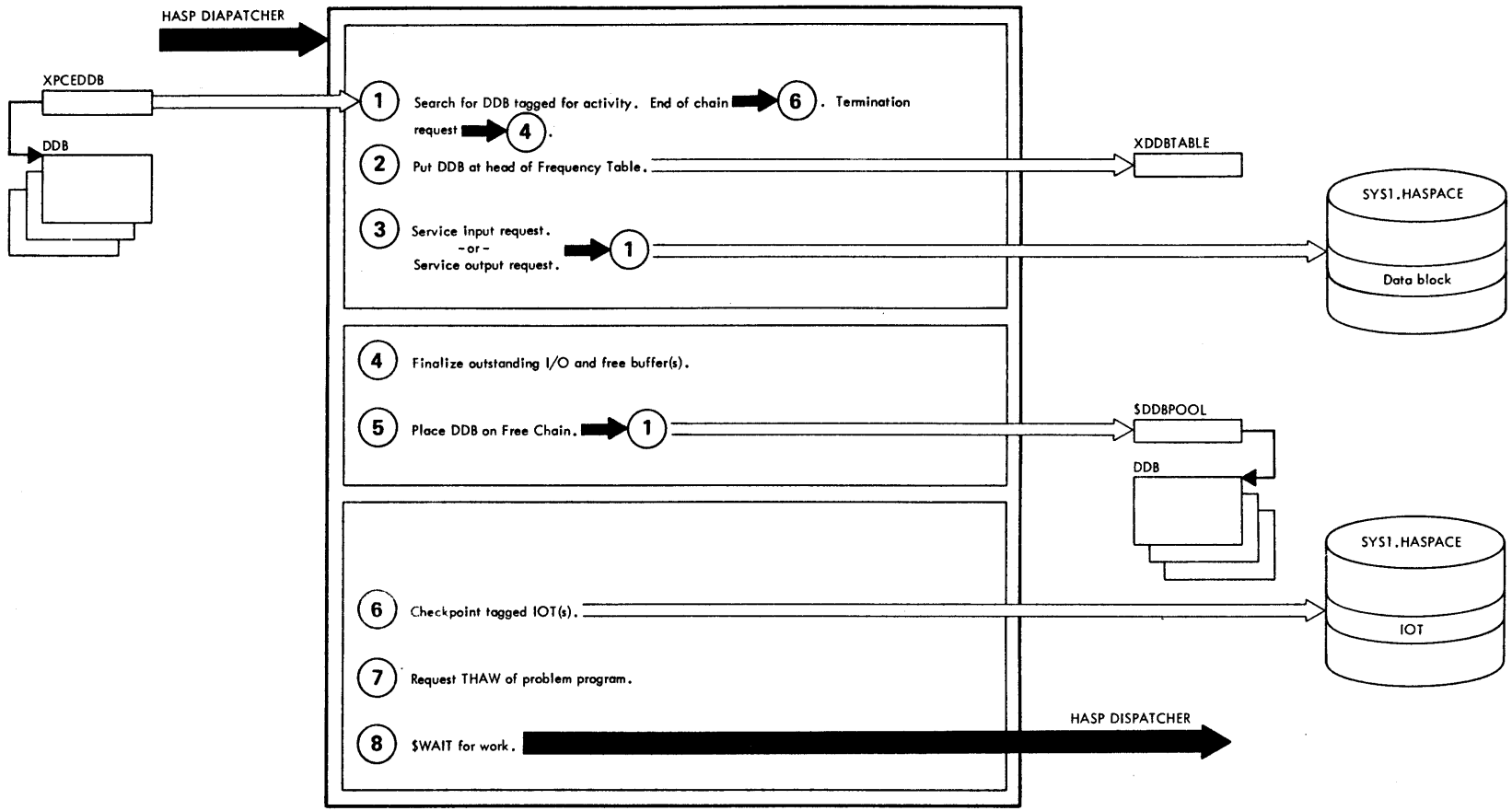
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPRDR	ROTPCARD	13 The Output Control Record is identified as a code X'43' record in the HASP data file.
HASPRDR	RDCKCOM	14 The DCTFLAGS bits are tested for $C or $Z.
HASPRDR	RWAIT	16
HASPRDR		17 The end of the HASP data set is marked with a record containing the code X'04'.

HASP XEQ Job Initiation

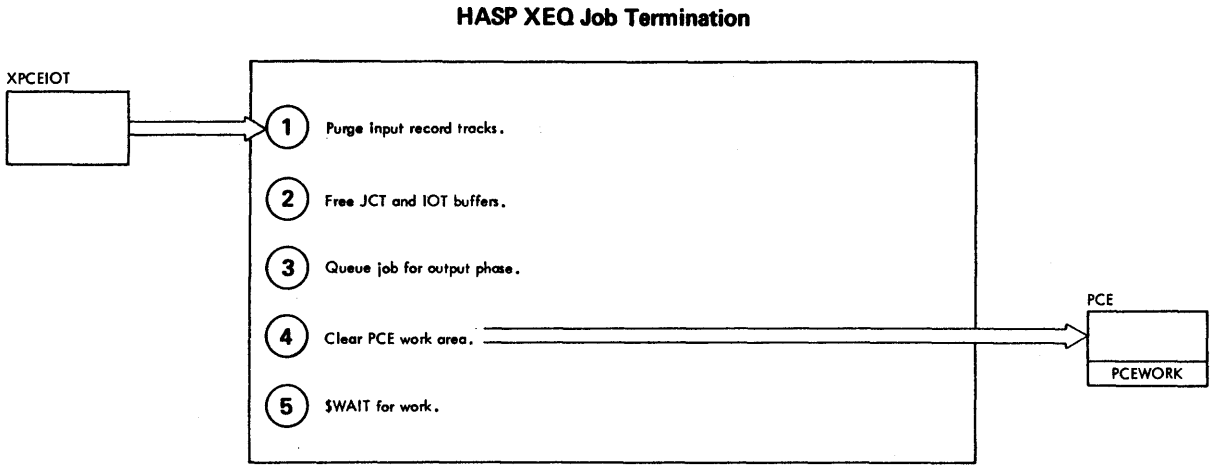


ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XJOBINIT	1 Each bit is analyzed to determine if a job exists on the HASP Job Queue which satisfies its selection criteria.
HASPXEQ	XJCTWOK	6 The checkpoint reflects the change in status for the job selected for execution.
HASPXEQ	XNOBATCH	11 The OS Reader/Interpreter is activated to read the HASP JCL data set related to the job selected for execution.

HASP XEQ I/O Service

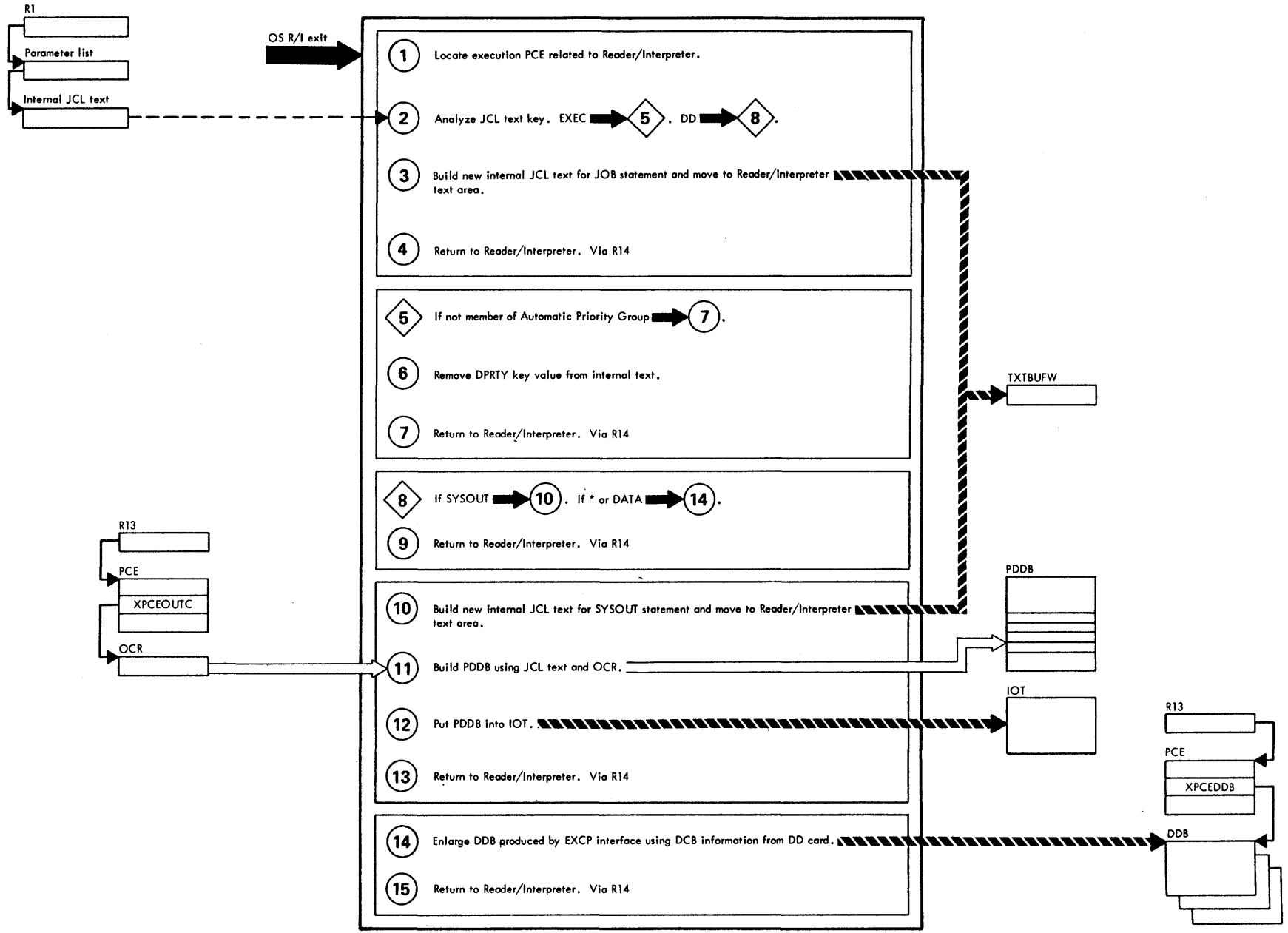


ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XTIMWAT	1 The DDBSTAT2 field determines if action is required by this DDB.
HASPXEQ	XDDBCONT	2 When buffer roll is required, the buffer related to the DDB at the end or bottom of the frequency table is the first candidate for rolling since this buffer has been dormant for the longest time.
HASPXEQ		3 I/O service involves the reading or writing of a HASP data block from SYS1.HASPACE.
HASPXEQ	XTERMIN8	4
HASPXEQ	XTERMFD	5 After the DDB is placed on the free chain, a $POST of DDB is used notifing all interested PCEs.
HASPXEQ	XIOTSCAN	6 If any IOT has changed as a result of the DDB request it is checkpointed.
HASPXEQ	XWAITCN1	7 The task which generated the DDB request is scheduled for reactivation by the Thaw Processor.


HASP XEQ Job Termination
XPCEIOT
1
Purge input record tracks.
2
Free JCT and IOT buffers.
3
Queue job for output phase.
4
Clear PCE work area.
5
$WAIT for work.
PCE
PCEWORK

ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	HASPXEOJ	1 If the job is to be restarted its output tracks are purged and it is requeued for input.
HASPXEQ	XCOMPRIO	3 The job is queued for output with a priority based upon the number of generated output lines as conditioned by HASPGEN parameters &XLIN and &XPRI.
HASPXEQ	XQEND	5 All PCEs are $POSTed with job and the Checkpoint Processor is $POSTed with work so that a checkpoint will reflect the new status of the job which just completed.

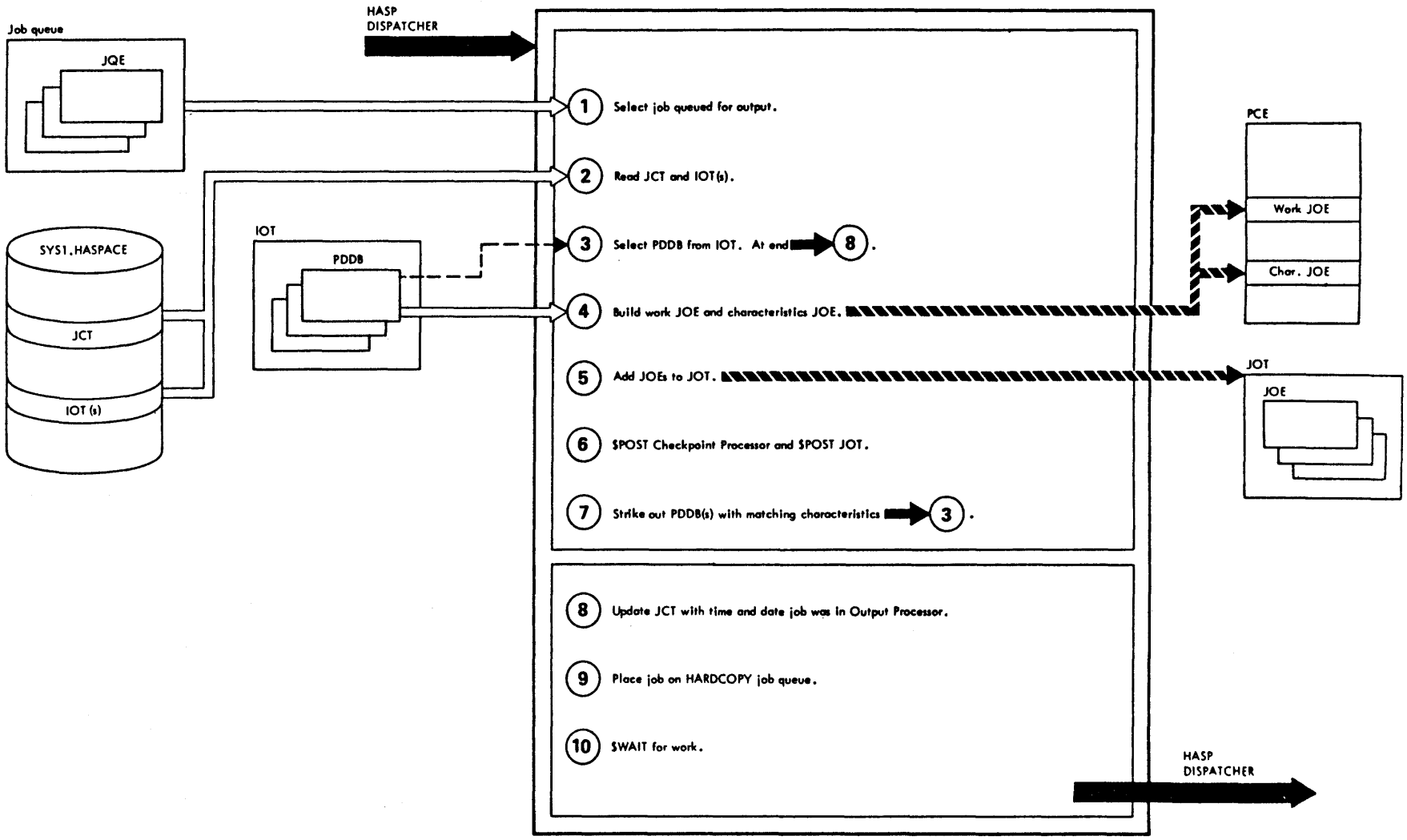
HASP Reader/Interpreter Appendage



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XJCLSCAN	1 Retrieve PCE address from location $RDRPCE.
HASPXEQ	XJCLJBPR	3 Class, PRTY, TYPRUN, and MSGCLASS are screened according to HASP disciplines.
HASPXEQ	XJCLXQ	5
HASPXEQ	XJCLDDPR	8
HASPXEQ	XJCLNOTB	11 The PDDB is made to reflect the UCS, FCB, forms number, maximum record count, destination, 3211 index, number of copies, 3800 Burster-Trimmer-Stacker threading, 3800 forms overlay name and count, 3800 copy modification module name, 3800 character arrangement tables, and 3800 copy groups.
HASPXEQ	XJCLIOT2	12 If an IOT must be written to DASD it is done under the control of the HASP task and the execution PCE.
HASPXEQ	XJCLDDDA	14 The DDB was acquired in the EXCP interface routine when a significant card image was encountered which was planted there during initial reading of the job stream.

Page of GY27-7255-0
Revised September 15, 1976
By TNL SN27-1555

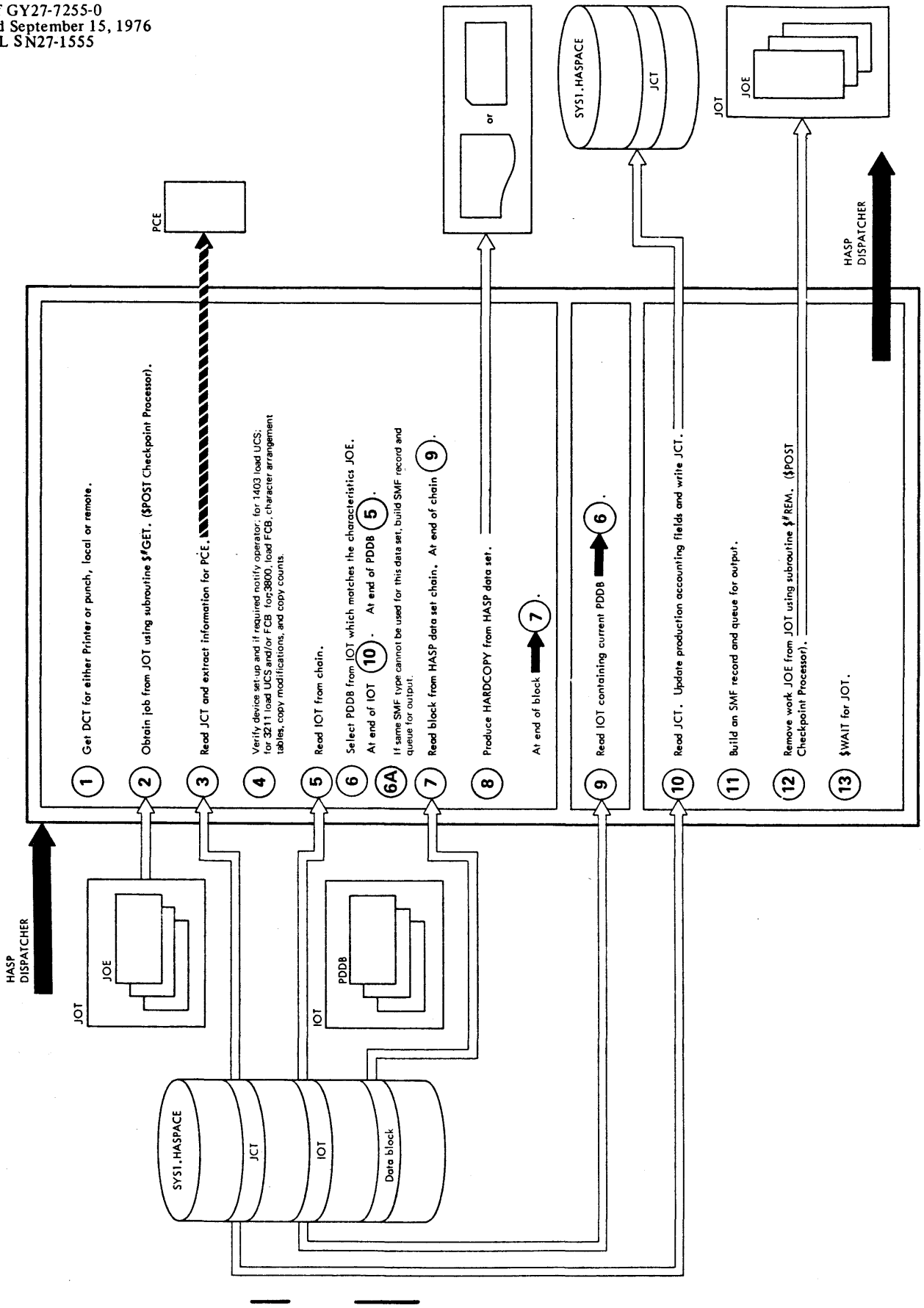
HASP Output Processor



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPPRPU	OPIDLE	1 If the $QGET routine cannot obtain a job queued for output the processor $WAITs for job.
HASPPRPU	OPJOB	2 The DASD address of the first IOT is obtained from the JCT and all IOT(s) are read into core.
HASPPRPU	PDBSCAN	3 The first PDDB whose null bit is not on is selected.
HASPPRPU	PDBJOE	4 A Work-JOE and a Characteristics-JOE are built in the PCE using the information in the selected PDDB.
HASPPRPU	OPJCOPY	5 If insufficient room exists in the JOT to add the new JOEs the processor $WAITS for JOT. The number of job copies requested controls the number of work JOEs placed in the JOT.
HASPPRPU		6 The $POST of the checkpoint processor and the $POST of JOT both occur within the $#ADD subroutine.
HASPPRPU	DDBNEXT	7 The null bit is set on in all remaining PDDBs which possess matching characteristics as to SYSOUT class, Forms ID, FCB ID, UCS ID, security level, route specification, CPU ID, special SYSOUT writer ID, FLASH ID, and BURST specification.
HASPPRPU	OPJCTGET	8 The JCT is read, the signon and signoff time and date fields are updated, and the JCT is written back.
HASPPRPU		9 If the job is marked for purging the job is placed on the purge queue. The hardcopy queue is designed to hold jobs while their SYSOUT records are being converted to hardcopy.

Page of GY27-7255-0
Revised September 15, 1976
By TNL SN27-1555

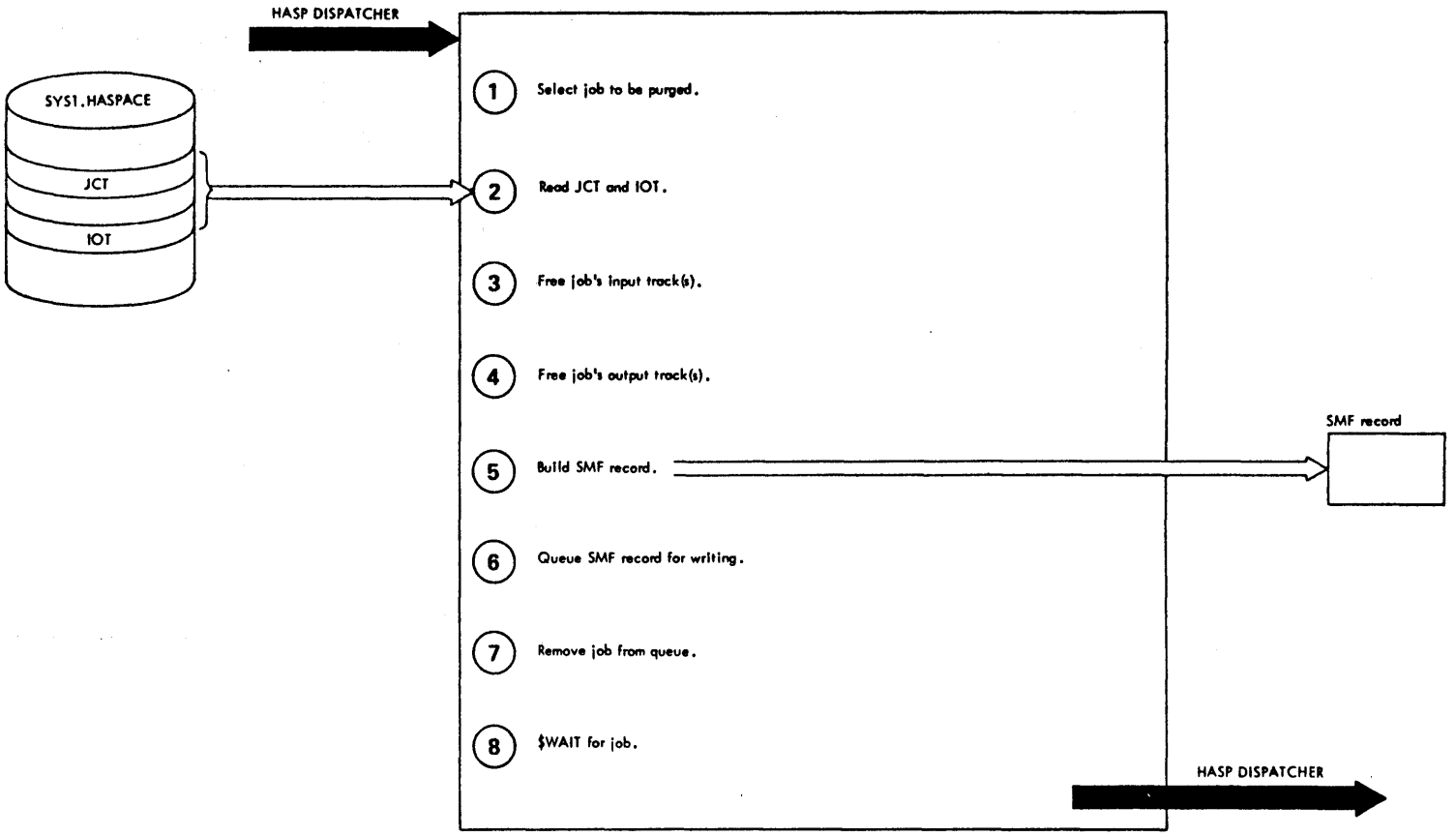
HASP Print/Punch Processor



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPPRPU	HASPPPI1	1 If a requested DCT is not available the Print/ Punch Processor $WAITs for unit.
HASPPRPU	PNORMWTO	The $POST of the Checkpoint Processor is performed within the $#GET subroutine. If no work is available the checkpoint Processor $WAITs for JOT.
HASPPRPU	HASPPPI2	3
HASPPRPU	PRPUDSV DSV3800	4 Ii the device requires set-up the processor informs the operator and $WAITs for I/O which will be $POSTed by the Command Processor when the operator responds.
HASPPRPU	PPIOTRD	5
HASPPRPU	PPPDB	6 Each PDDB represents a chain of HASP data records which contain related SYSOUT data. The field named PDBMTTR conveys the starting MTTR for each set.
HASPPRPU	PDDBFCHK	6A A type-6 SMF record is built reflecting the amount of work done so far on the related job.
HASPPRPU	PNXTBLK	7 Blocks are read from the chain until the MTTR for the next block is zero.
HASPPRPU	PNXTCCW	8 A multi-command channel program is built in the PCE and executed to produce hardcopy.
HASPPRPU	PPPIOTRD	9 The IOT containing the PDDB just processed is read to restart the PDDB search/match sequence.
HASPPRPU	PPDONE	10 The JCT is protected from simultaneous updating by a bit called QUEJCTSW in the Job Queue Element.
HASPPRPU	PJCTWOK	11 A Type-6 SMF record is built representing the amount of work performed in behalf of the related job.
HASPPRPU	PJOEREM	12 The job will be placed on the purge queue if its number of work JOEs on the JOT euqals zero.
HASPPRPU	PNOJOB	13 If another Work-JOE cannot be obtained from the JOT the processor $WAITs for JOT.

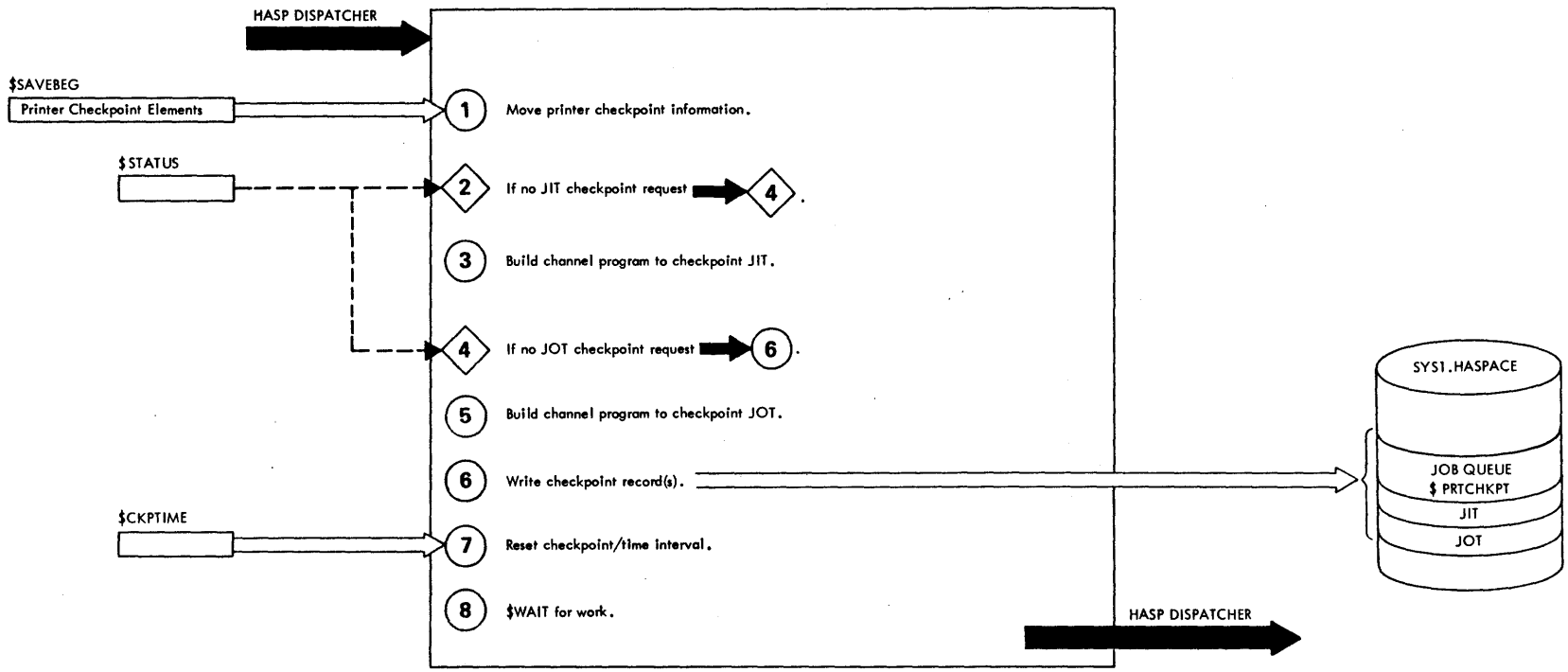
Page of GY27-7255-0
Revised September 15, 1976
By TNL SN27-1555

HASP Purge Processor



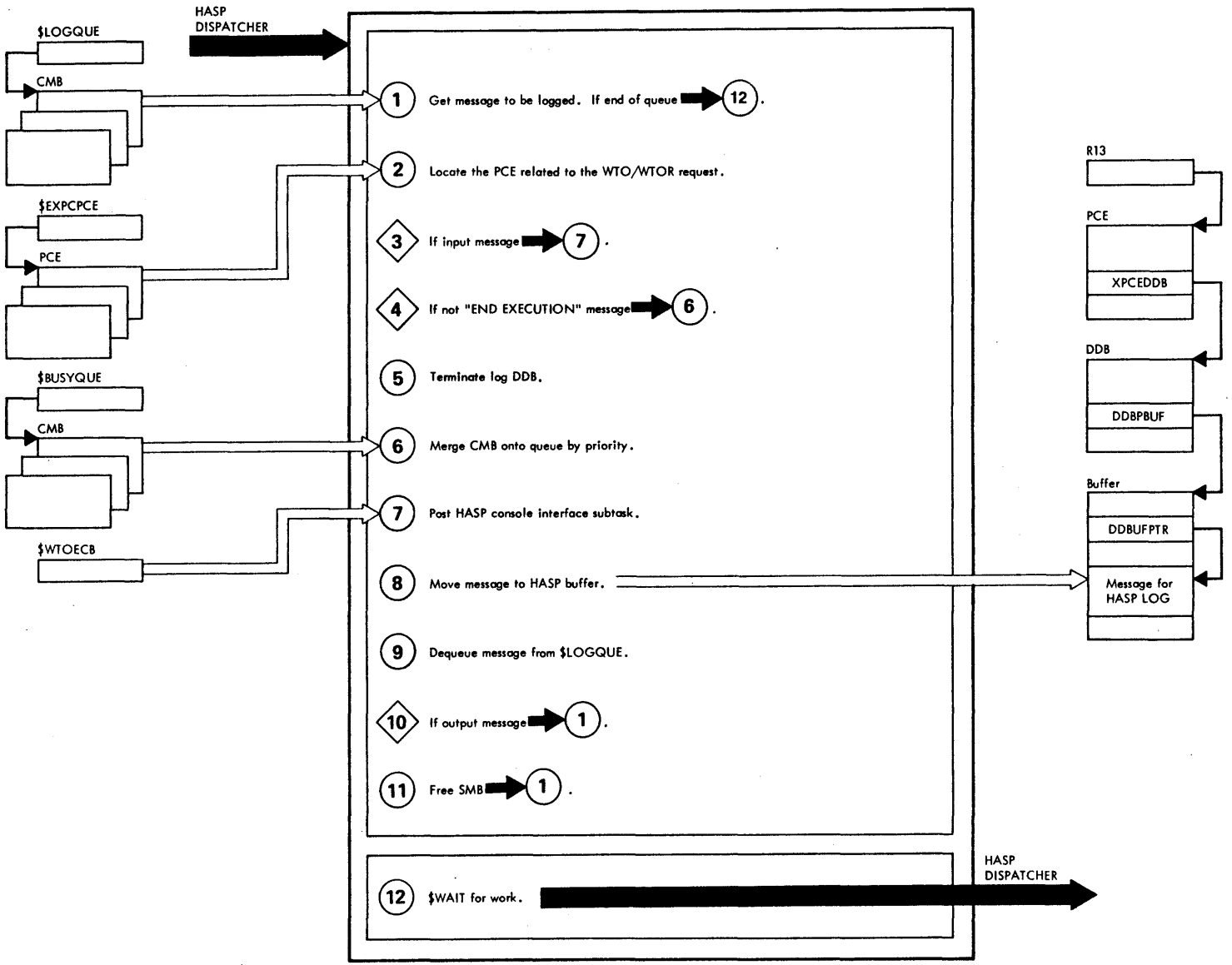
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPMISC	HASPVPUR	1 HASP JQEs are searched using $QGET for a job queued for purging.
HASPMISC	VNOERROR	3 The job's input track are purged using $PURGE and the information in the JCTCYSAV field.
HASPMISC	VPRGCYL	4 The job's output tracks are purged using $PURGE and the information in the IOTCYMAP field.
HASPMISC	VSMFPRG	5 If EXT=YES was specified as a SMF parameter to OS, two SMF buffers (type 26 record and JMR) are built and queued up to be written by the SMF subtask. If EXT=NO was specified, only the type 26 record is built and queued.
HASPMISC	VREMJOB	7 The job is removed from the HASP job queue using $QREM and an operator message indicating the job is purged is created.
HASPMISC	VNOJOB	8 The processor $WAITs for a JOB and returns to the HASP dispatcher.

HASP Checkpoint Processor



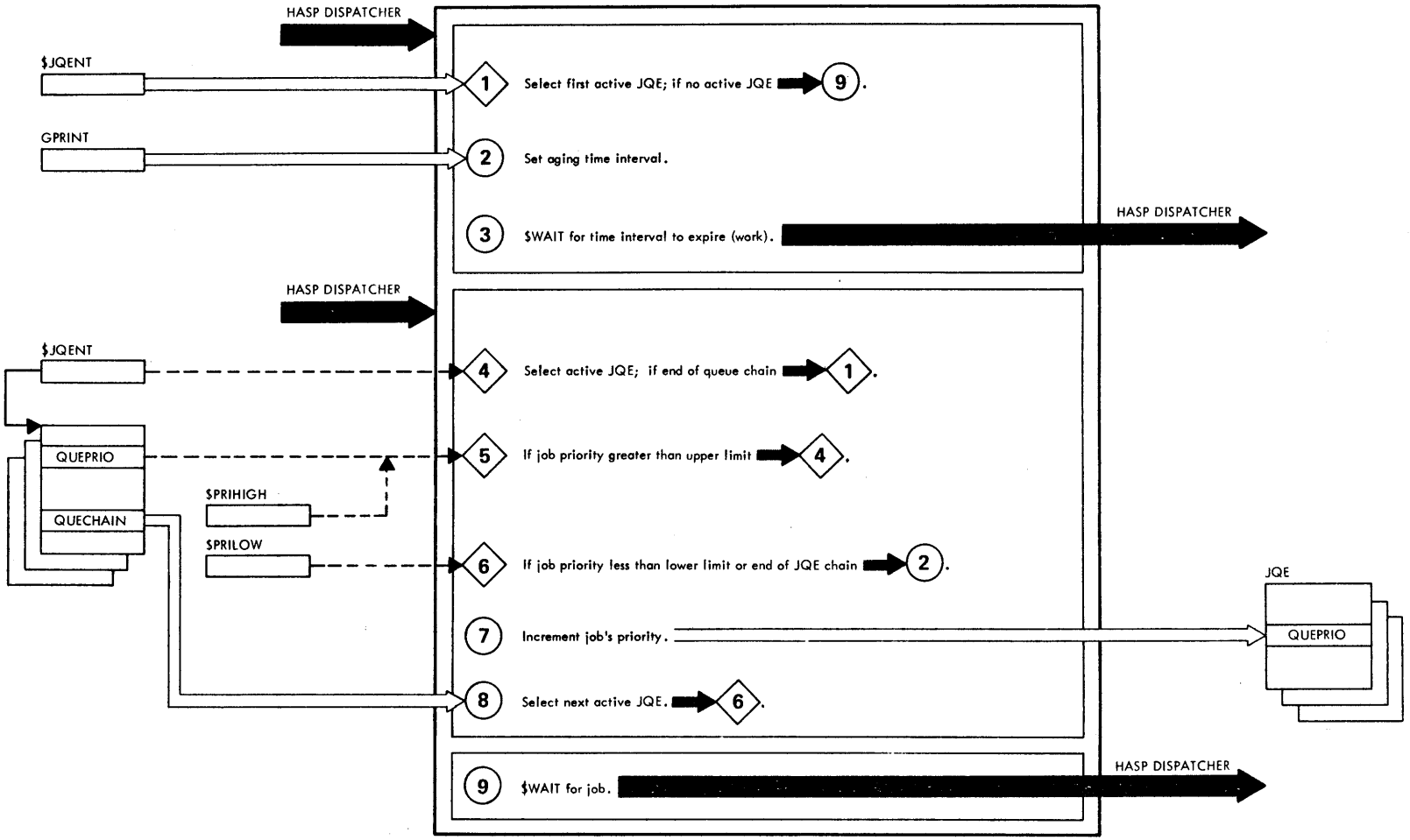
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPMISC	KCHECK	1 The Print/Punch Processor checkpoint element pool is moved to a area contiguous with the HASP Job Queue. This precludes having to WAIT for the checkpoint records to be written before modifing an element.
HASPMISC	KEXCP	6 Up to 3 physical records may be written. Record 1 consists of the entire HASP Job Queue followed by all printer checkpoint records. Record 2 consists of the HASP Job Information Table. Record 3 consists of the HASP Job Output Table. The write is $WAITed upon before processors $WAITing for a checkpoint to be taken are $POSTed.
HASPMISC	KIOWAIT	7 The remaining checkpoint time interval is canceled using $TTIMER and the new interval is set using $STIMER.
HASPMISC	KWAIT	8 The Checkpoint Processor $WAIT for work until it is $POSTed either because the checkpoint interval timer has expired or another processor has recognized a significant event and requests a checkpoint.

HASP LOG Processor



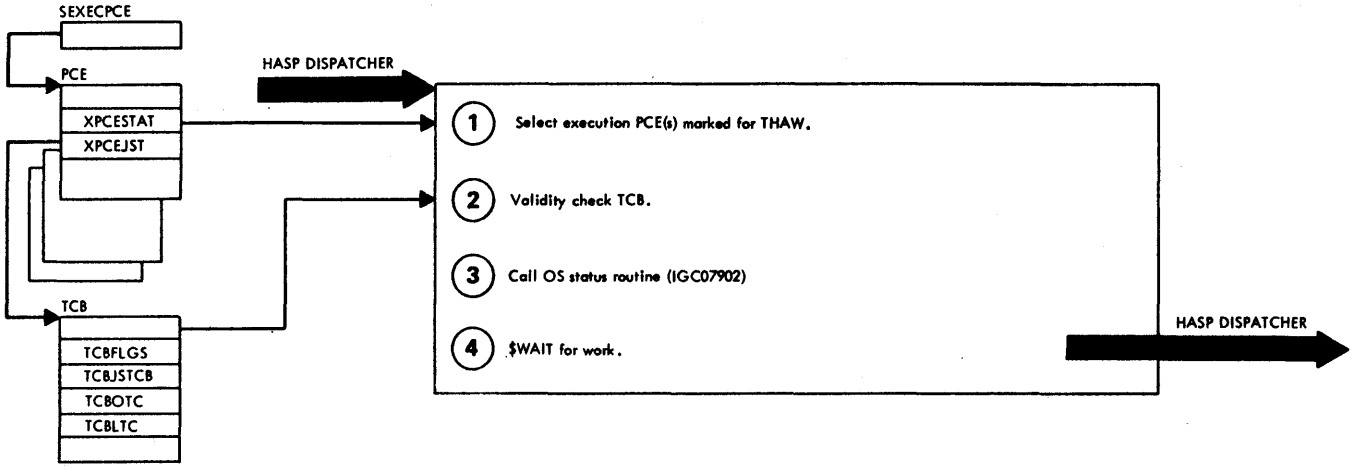
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	HASPLOG	1 The processor had been $WAITing for work.
HASPXEQ	LTESTJNO	2 The PCE relationship is established by job number in the message as compared with the PCE JCT.
HASPXEQ	LHASPMSG	5 The log DDB is marked for action and all PCEs are $POSTed for work.
HASPCON	$WQUEBUF	6 This code is activated by a BAL in HASPXEQ.
HASPCON	WENDQUE	7 An OS POST macro is used to post $WTOECB.
HASPXEQ	LOGOUT	8 The message is written into the HASP log data set.
HASPXEQ	LPUSHUP	9
HASPXEQ	LOGREADX	11 The CMB is freed by using the HASP service routine $FREEMSG.

HASP Priority Aging Processor



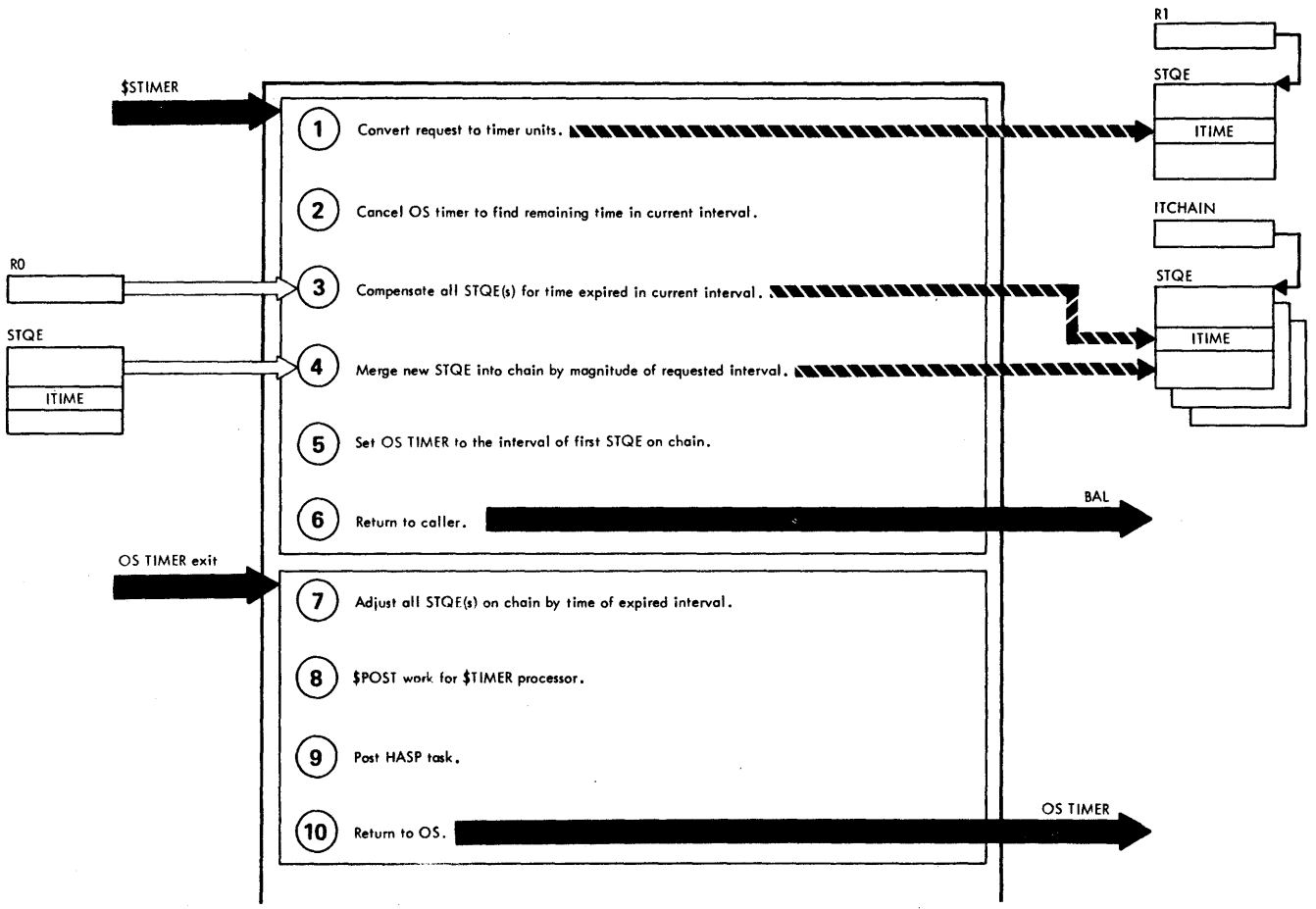
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPMISC	GPRSTART	1 The entire priority aging processor is an overlay segment.
HASPMISC	GPRTIME	2 The aging interval is determined by the HASP generation variable &PRIRATE.
HASPMISC	GPRLOOP1	5 The highest priority to be considered for aging is determined by the HASP generation variable &PRIHIGH.
HASPMISC	GPRLOOP2	6 The lowest priority to be considered for aging is determined by the HASP generation variable &PRILOW.

HASP Execution Thaw Processor



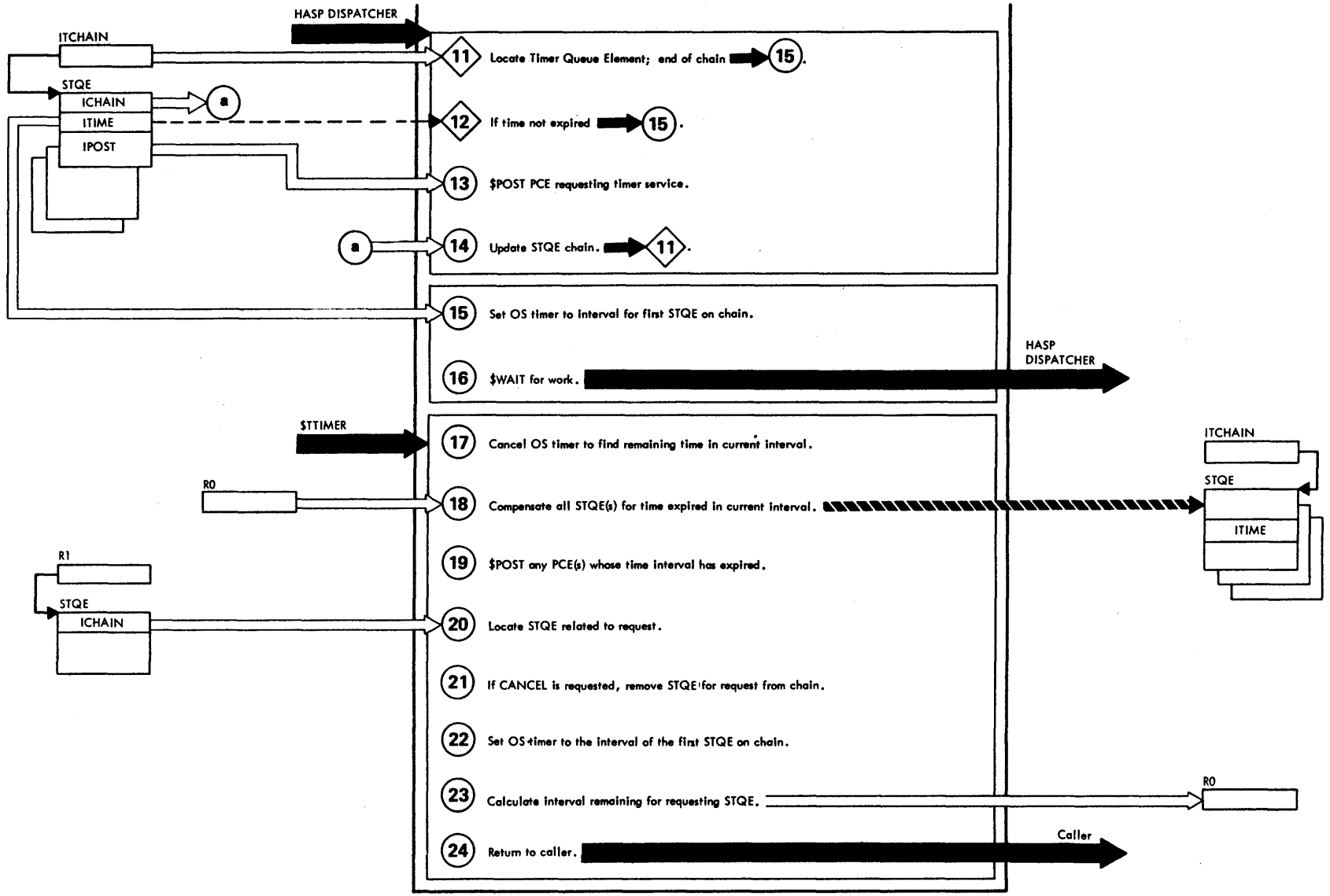
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	HASPXTHW	1 A special thaw cell is inspected for the presence of a TCB address before the execution PCEs are checked.
HASPXEQ	XWARM	2 Checks are made against the argument TCB address to prevent HASP from abending.
HASPXEQ		3 The argument list passed to the status routine causes the subject TCB to be made dispatchable.
HASPXEQ	XTH3	4 After all candidates for thawing have been considered the processor $WAITS for work.

HASP Timer Services (1 of 2)



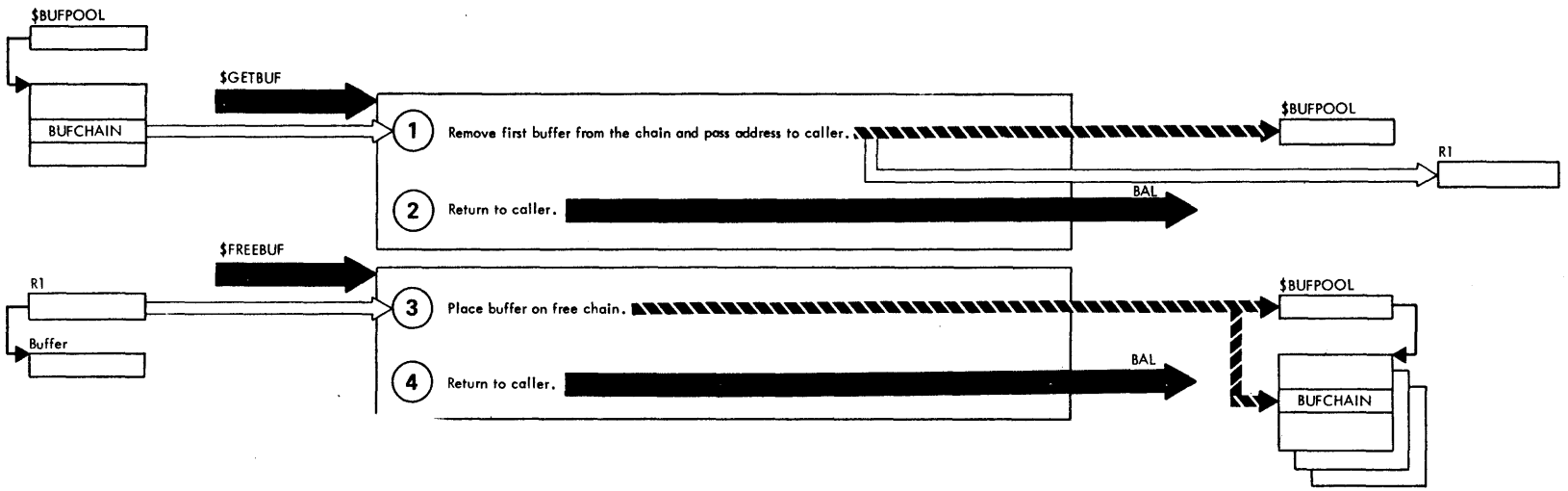
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$STIMER	1 The parameter value passed is converted to S/370 timer units (see $STIMER macro description).
HASPNUC		2 The OS macro TTIMER cancel is used.
HASPNUC	IPOSTIT	3 The PCE for any STQE whose time has expired is $POSTed for work and the STQE chain is updated.
HASPNUC	INEXT	4
HASPNUC	ISETINT	5 The new interval is set using the OS macro STIMER.
HASPNUC	IRETURN	6 The return is to the register link.
HASPNUC	ITIMEUP	7 This is the asynchronous exit for the STIMER macro.

HASP TIMER Services (2 of 2)



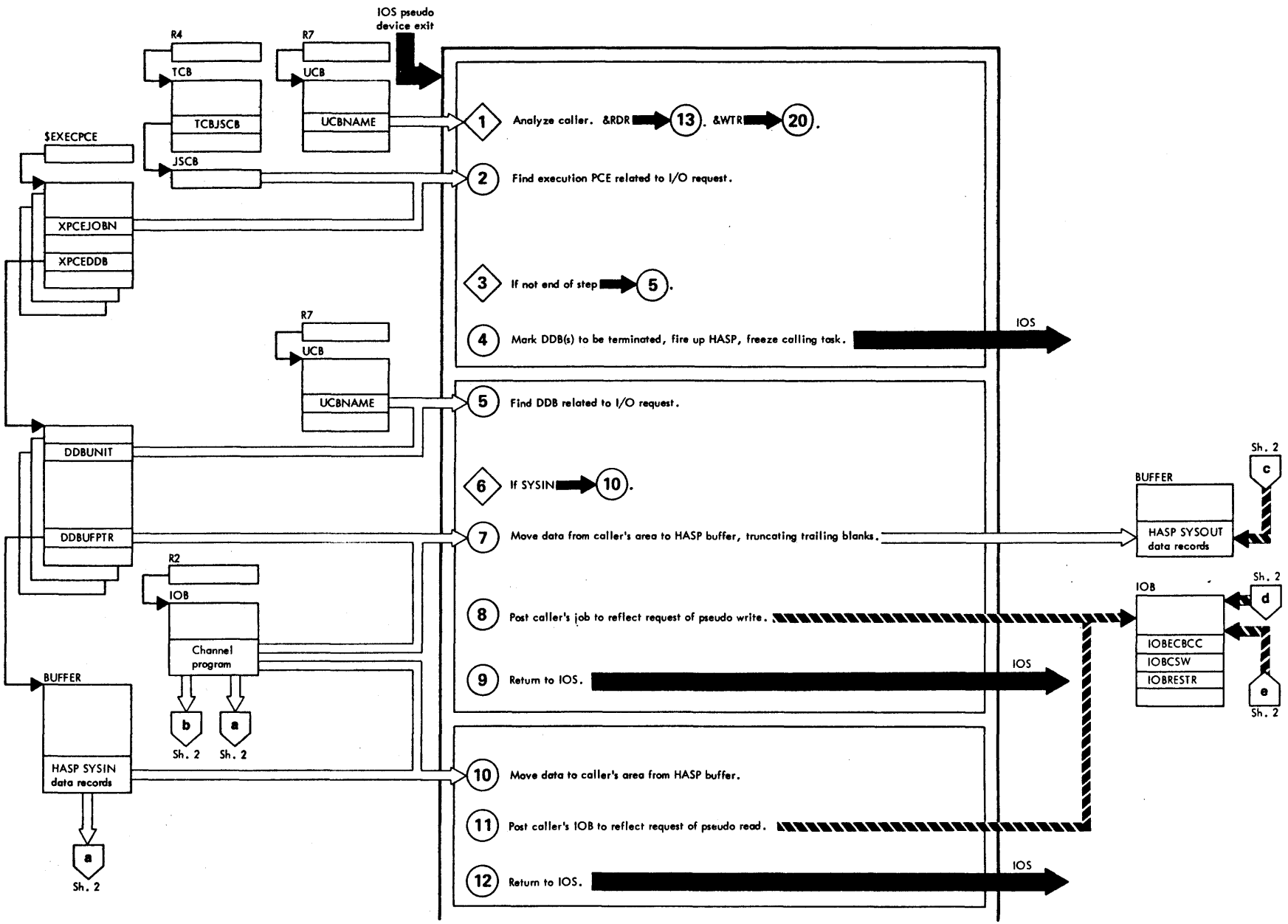
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$TIMER	11 The subroutine IPOSTIT is used to perform steps 11 through 13.
HASPNUC		15 The subroutine ISETINT is used to set the OS time interval for the first STQE by executing the macro STIMER.
HASPNUC	$TTIMER	17 Entry here is from the HASP macro $TTIMER.
HASPNUC	IRET1	22 Same as 15 above.

HASP Buffer Pool Manager



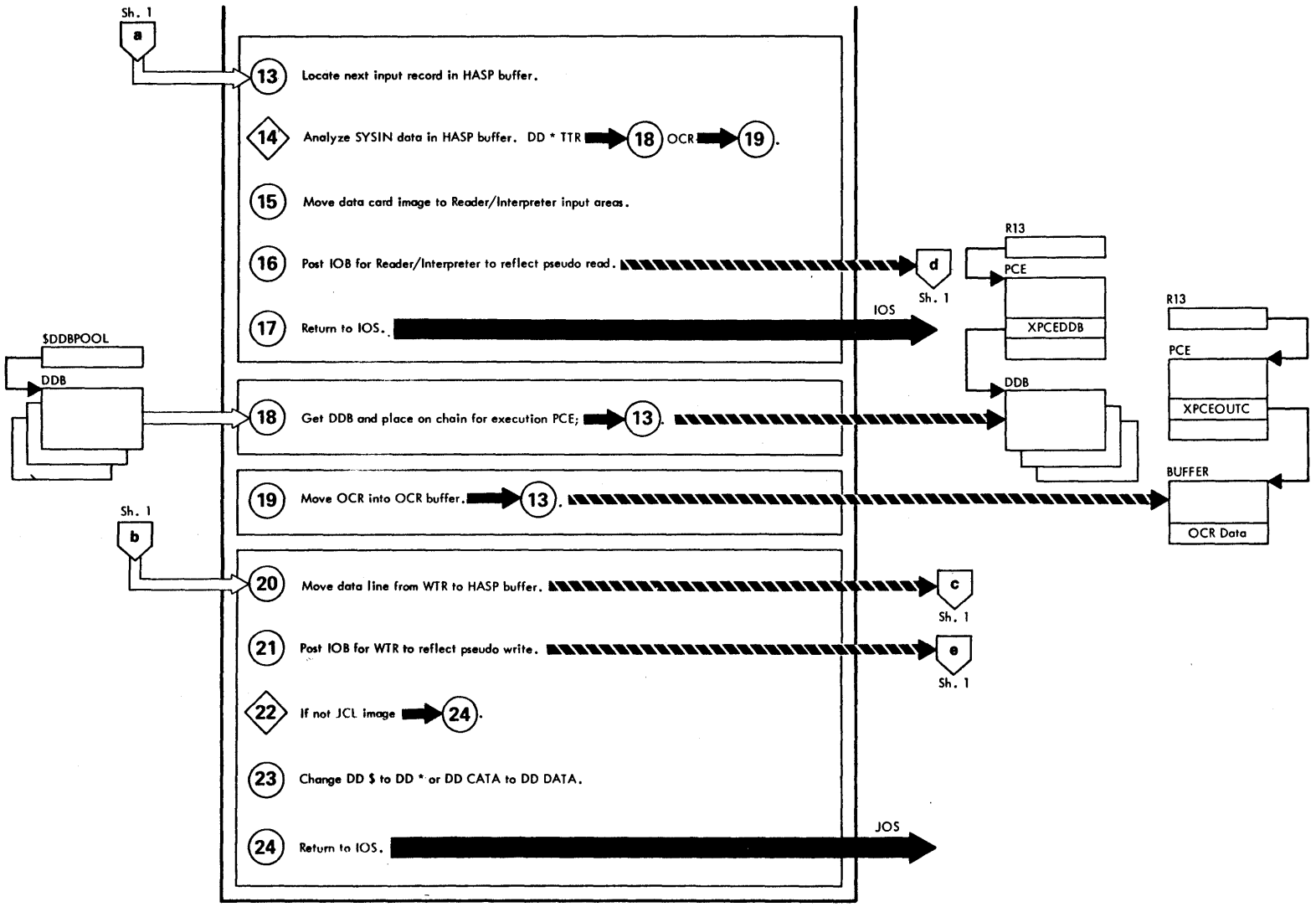
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$GETBUF	1 The subroutine BBUFCHEK validity checks the address for the new buffer. A HASP error B01 will occur if the address fails the tests.
HASPNUC	$FREEBUF	3 The same validity check subroutine is performed as in 1 above.
HASPNUC	BUFPOST	All PCEs are $POSTed to notify them of the availability of a free buffer in the event they may be $WAITing for one.

HASP EXCP Interface (1 of 2)



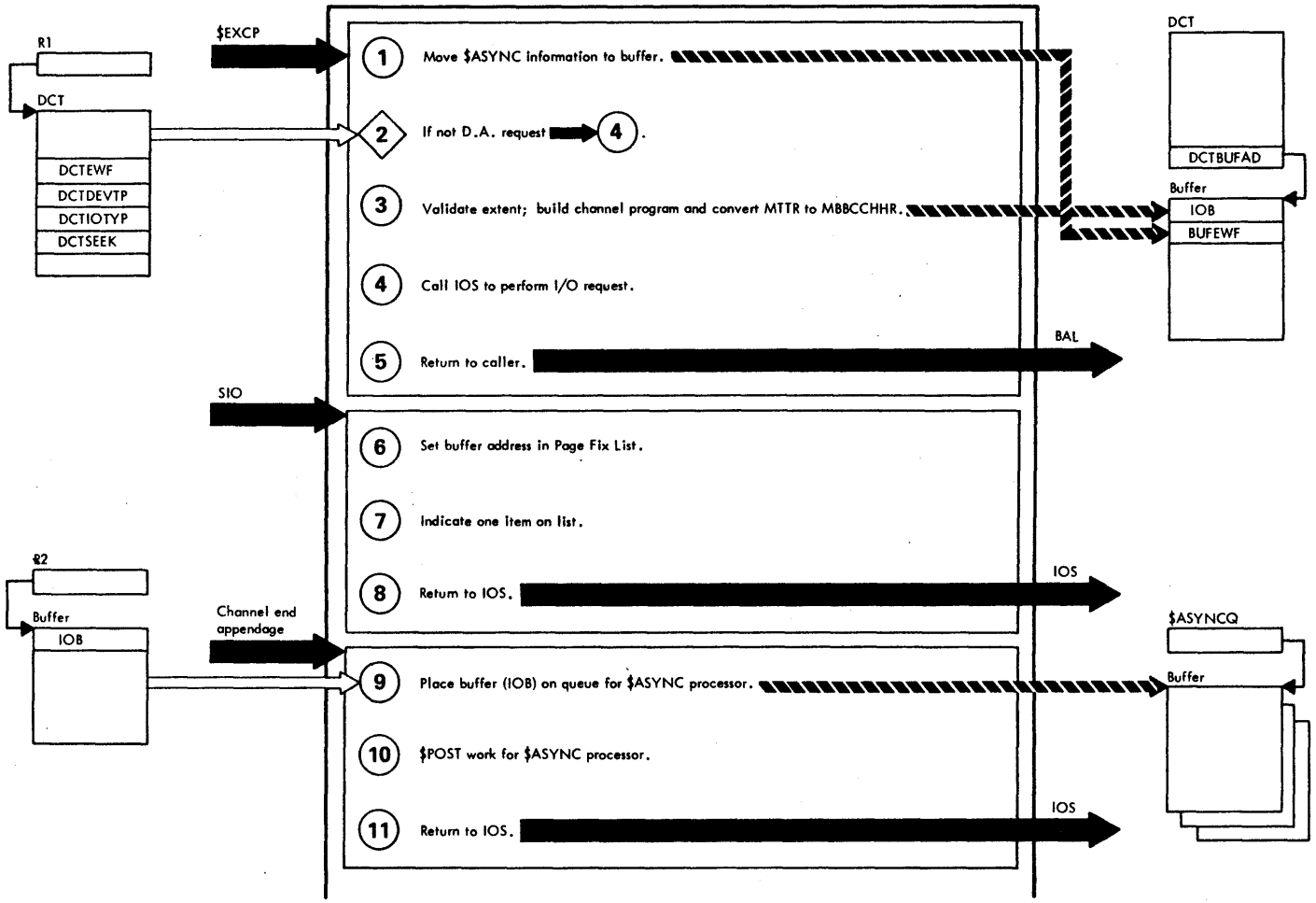
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$EXCP	1 The synchronous activity to be performed at I/O completion is established.
HASPNUC	ESENDIT	4 The channel program is executed using a standard OS EXCP macro.
HASPNUC	ESIOPGEX	6 The start I/O appendage is used to page fix the HASP buffer which contains the IOB.
HASPNUC	ECHANEND	9 The ASYNC processor operates under control of the HASP task and performs functions related to the completed I/O.

HASP EXCP Interface (2 of 2)



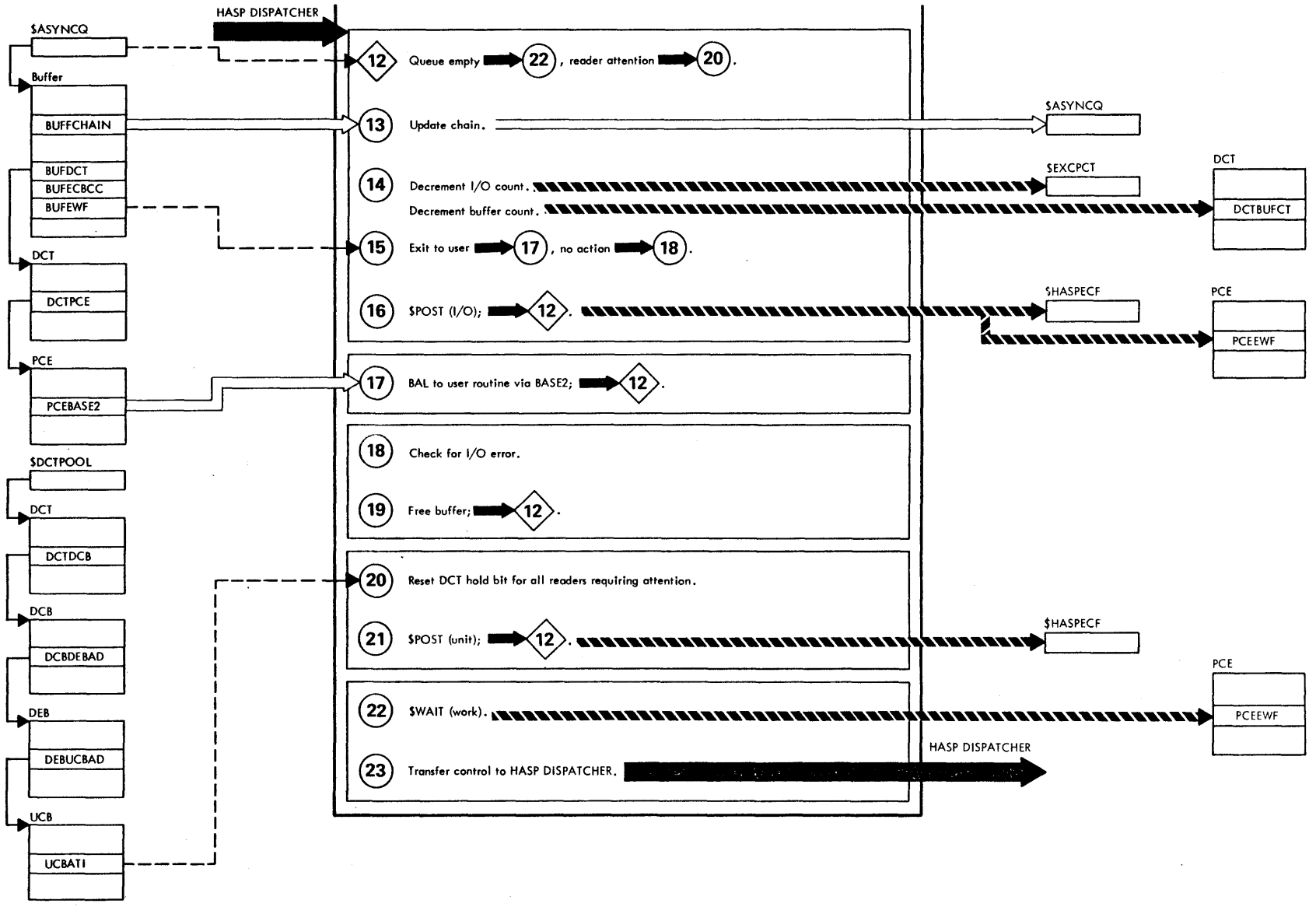
ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPNUC	$ASYNC	12 The HASP hot reader is controlled from this entry.
HASPNUC	AOK	13 The system is disabled while the chain is updated to prevent interference from the channel end appendage.
HASPNUC		15 If BUFEWF is negative, a user exit is indicated. If BUFEWF is zero, no action is indicated.
HASPNUC	AENTER	17
HASPNUC	AFREE	18 If an I/O error is detected the HASP macro $IOERROR is issued to note same.
HASPNUC	ATT1	20 All readers whose UCB attention bit is on are $POSTed with unit which causes them to begin reading cards.

HASP I/O Services (1 of 2)



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XJOBSRCH	2 Step change is determined by the manner in which the related PCE is found.
HASPXEQ	XTERMODB	4 The DDB(s) are only marked. Actual termination occurs under the control of the HASP task.
HASPXEQ	XFINDDDB	5 If no DDB exists one is obtained and place on chain of PDDBs for this PCE.
HASPXEQ	XSYSOUT	7 If the CCW is not a write no data storage space is used in the buffer.
HASPXEQ	XRET	8 The IOB is updated to reflect the effect of the pseudo I/O and the IOB ECB completion code is stored in the ECB.
HASPXEQ	XSYSIN	10

HASP I/O Services (2 of 2)



ASSEMBLY LISTING NAME	ASSEMBLY LISTING LABEL	NARRATIVE
HASPXEQ	XRDRCALL	13
HASPXEQ	XDDSTAR	18
HASPXEQ	XOCRCARO	19 If no OCR buffer is available the reader task is frozen, HASP is fired up, a buffer is gotten, and the reader is thawed.

SECTION 3

HASP

PROGRAM ORGANIZATION

HASP PROGRAM ORGANIZATION OVERVIEW

This section presents a description of the constituent elements of HASP in terms of their functions and organization. As an aid to understanding, the HASP functional elements are arranged into six groups that are similar in terms of their functions, their basic method of operation, and their relationships to OS and other elements of HASP. These groups are:

1. HASP Management Programs
2. Basic Functional Processors
3. Control Processors
4. Control Service Programs
5. Miscellaneous Programs
6. HASP Remote Work Station Programs.

HASP MANAGEMENT PROGRAMS

These are HASP programs functioning as tasks directly under control of the Operating System. For example, the HASP Task Dispatcher is entered as an OS task and subsequently passes control to the various component processors of HASP as their functions are required.

BASIC FUNCTIONAL PROCESSORS

These are the HASP programs that operate under control of the HASP Dispatcher to carry out the main processing flow of HASP; that is, input, execution, output, and purge. These programs are in turn supported by the control processors and control service programs.

CONTROL PROCESSORS

Operating under the control of the HASP Dispatcher, these programs control and provide the various HASP and OS facilities used in support of the basic functional processors.

CONTROL SERVICE PROGRAMS

These are a series of subroutines operating under control of the basic functional and control processors and providing various system resources in support of HASP.

MISCELLANEOUS PROGRAMS

These are HASP programs providing initialization and overlay building facilities.

HASP REMOTE WORK STATION PROGRAMS

These are HASP programs, operating in remote CPUs, furnishing remote terminal support in extension of the basic functional processors, control processors, and control service programs.

HASP DISPATCHER

The HASP Dispatcher allocates HASP CPU time to the HASP processors. It receives control each time the HASP task goes from the WAIT state to the ACTIVE state. The HASP Dispatcher utilizes an ordered queue of Processor Control Elements (PCEs) to distribute the CPU time among the HASP processors. The Event Wait Field (EWF) in each PCE is a 2-byte field which is used to control the dispatchability of the processors. Any processor or control service routine may issue a $WAIT macro instruction at any time, which turns on a particular bit in the EWF corresponding to the event waited for and returns control to the HASP Dispatcher to allow other processors to be dispatched. The processor will not be given control again until some other system function issues a $POST to the appropriate EWF.

The events reflected by the EWF fall into two categories, the first of which is the synchronization of the use of common system resources such as buffers, direct-access space, etc. With the exception of the general $POST bit $EWFPOST, the bits in the first byte of the EWF field are used to indicate the particular resource being waited for and correspond exactly to the Event Completion Field (ECF) in the Dispatcher. The ECF is posted whenever a resource becomes available and is propagated through all processor EWFs by the Dispatcher. Thus, if a track becomes available on a direct-access device, every processor (PCE) which has issued a $WAIT for a track will be put in contention for CPU time to try to obtain the track or tracks that have been released.

The second byte of the EWF is used to synchronize a processor with respect to a specific event applicable only to that processor, such as a particular I/O completion. This section of the EWF must be posted directly by the system routine performing the required function.

When scanning the PCE chain, the HASP Dispatcher, upon discovering a zero EWF (no events being waited for), will enter the code controlled by the PCE immediately below the $WAIT which had returned control to the Dispatcher. All registers of a processor which issues a $WAIT are preserved in the PCE and are reloaded prior to entering the processor (register R15 is destroyed by the $WAIT macro to provide the address of the $WAIT, i.e., the resume point). A processor may return control to the Dispatcher only by means of the $WAIT macro. In the event any $POST macro was executed by the processor dispatched or by any of the HASP asynchronous service routines, the Dispatcher's ECF field will be altered to reflect the $POST. The general $EWFPOST bit represents a $POST of a specific processor (second byte of the EWF). If the ECF field indicates no $POST has occurred, the HASP Dispatcher continues to scan down the PCE chain starting with the next PCE. However, if the ECF field indicates $POSTs have occurred, the bit for the general $POST is removed and scanning is resumed at the beginning of the PCE chain, after promulgating any remaining ECF $POST indicators.

Upon reaching the end of the PCE chain, the Dispatcher examines several counts to determine if any jobs, RJE lines, or messages are being processed. If there is uncompleted system activity, an OS WAIT state is entered to wait for some external event (I/O interrupt, etc.) to activate HASP again. This WAIT state allows use of the CPU by other tasks in the system. If there is no activity, the message:

ALL AVAILABLE FUNCTIONS COMPLETE

is sent to all local operator consoles, and HASP is placed in the WAIT state.

When scanning the PCE chain, the Dispatcher may detect the special case of a PCE which is not dispatchable (PCEEWF is not zero) but is waiting only for the OROL bit. This situation is created when, while the PCE is waiting for other event(s), the overlay area being used by the PCE is preempted by the Overlay Roll Processor for other use. Subsequently, the other event(s) being waited for are posted, allowing the Dispatcher to detect the "OROL only". The processor in such a condition is dispatched into Overlay Services. The actions performed are identical to those described for the $LINK service under Overlay Service, beginning with the fourth paragraph describing search of overlay areas. The processor will be dispatched by Overlay Service (if the requested routine is in storage) or will be set to $WAIT for OLAY, allowing the Dispatcher to continue its PCE scan.

When reaching the end of the PCE chain, the Dispatcher also performs special actions for the Overlay Roll Processor. If the queue of work for this processor (PCEs waiting for assignment to an overlay area) is not empty the processor is dispatched. Because this processor has a required position at the end of HASP's CPU priority chain and because it does not return to the dispatcher by $WAIT, there is no PCE for it and the Dispatcher enters it by direct branch.

OVERLAY ROLL PROCESSOR

This processor operates in conjunction with the Overlay Service routines. The description of these routines should be read also to provide proper background to understanding of Overlay Roll. The objective of this processor is to prevent system lockout due to $WAITs in overlay routine coding.

This processor does not have a PCE but effectively operates as the lowest processor on the HASP Dispatcher chain. This means that all processors with their requested overlay routine in an overlay area will have at least one chance to execute code or otherwise use their overlay routine before the overlay area containing that routine is taken for other use. Overlay Roll does not receive control unless all other HASP processors are in a $WAIT state, i.e., HASP is ready to relinquish control to OS by WAIT.

Since there is no PCE for Overlay Roll, the Dispatcher enters it directly, with the content of $WAITACE in register WD, when that content is nonzero. The Dispatcher provides local addressability in BASE2, and Overlay Roll establishes the base address for Overlay Service in register WC so its subroutines and tables may be used. On each entry, the queue beginning with $WAITACE of PCEs waiting for an overlay area is processed. The following attempts are made to secure one or more overlay areas and begin reading a requested routine into them.

For each group of one or more waiting PCEs requesting the same overlay routine, all overlay areas are searched to find a suitable one. If a read operation to load another overlay routine is in process, the area is never taken. Users of that other routine are allowed at least one chance to execute after read completion is processed by Overlay $ASYNC Exit.

For each overlay area which does not have a read in process, the OACEPRIO field is examined and the chain of all current users (beginning at OACEPCE) is searched to determine if any user is waiting for I/O. This I/O would be other than an overlay read operation, would be expected to complete soon, and would therefore make it less desirable to preempt that area. The lowest priority area with no user waiting for I/O is chosen, if any; otherwise, the lowest priority area is chosen.

Since an overlay routine is "refreshable" at $WAIT time, it is not necessary to literally "roll" (i.e., write to direct access) a preempted overlay area. Each PCE on the chain of current users (OACEPCE) is processed to prevent further use of the preempted area by it. The reentry address (PCER15) is sized to determine if it points into the overlay area and, if so, is made relative by subtracting the overlay area address. The PCE is forced into a $WAIT OROL state, in addition to the other $WAIT conditions present. When other $WAIT conditions have been POSTed, the Dispatcher detects the PCE waiting for OROL only and sets it to call on Overlay Service. The OLOD subroutine is eventually called to refresh the routine directly or, if the PCE gets into the $WAITACE Queue, by OEXIT subroutine or by Overlay Roll.

The area preempted is used to read in a new overlay routine, to be used by the highest priority PCE group on $WAITACE. The OLOD subroutine is called to begin the read operation.

If there are more PCE groups on the $WAITACE Queue, the above actions are repeated. When Overlay Roll finally exits directly to the Dispatcher, the $WAITACE queue is either empty or all overlay areas have an overlay read operation started, to be posted by Overlay $ASYNC Exit.

HASP COMMUNICATIONS SUBTASK

All HASP $WTO messages directed to Operating System consoles (including logical consoles) are queued to the $BUSYQUE queue and are serviced by the HASP Communications subtask (HASPWTO). The task is used, in lieu of the HASP task issuing WTO SVCs, to prevent HASP job processing from stopping while the Operating System resources are not available. When a Console Message Buffer (CMB) is placed in the $BUSYQUE queue the HASP task POSTs the subtask. The subtask performs the following functions:

1. $BUSYQUE queue is examined for CMBs queued for the Operating System consoles messages. If the subtask is in the process of continuing a multiline WTO (MLWTO), all logical console requests are ignored until the MLWTO terminates. If no eligible CMBs are found on the $BUSYQUE queue, the subtask waits for new CMBs (POSTed by the HASP task) and repeats the $BUSYQUE queue scan.

2. The list form of an Operating System WTO is formatted for normal HASP WTO messages. If the CMB indicates that the message is for a specific console (UCMID), the WTO is altered to indicate such; out-of-line areas are detected and result in MLWTO formatting of the WTO.

3. Logical console routings are converted to Operating System equivalents and the message text is copied to the WTO from the CMB. Control fields are shifted to complete the WTO parameter list for normal logical console and UCMID WTOs.

4. If the logical console message contains a $DOMACT flag, the WTO parameter list is altered to indicate immediate action required. The WTO SVC is issued and the DOM ID is saved in the CMB. If the $DOMACT flag is still on, the CMB is queued to the $DOMQUE queue and control is returned to step 1 above. If $DOMACT flag has been reset, the DOM SVC is issued directly.

5. The CMB is freed via the $FREEMSG routine, and if the condition code is zero on return, the HASP task ECB is POSTed.

6. If the WTO contains immediate action flag, control is given to step 1 above; otherwise, the WTO SVC is executed. If the request is for a MLWTO, the connect ID is saved and a MLWTO in process flag is set or reset, depending on whether or not the end line indicator appears in the WTO after formatting. Control is passed to step 1 above.

HASP SMB WRITER

The primary function of the HASP SMB Writer (HASPWTR) is to read System Message Blocks (SMBs) from the data set SYS1.SYSJOBQE and "print" them to HASP. The process signals the end of the OS execution phase of a job's processing and makes the messages (JCL, JCL diagnosis, allocation/disposition, SMF, etc.) available to HASP, to be later printed with print data sets of the job previously SPOOLed by HASP. This program is used as an attached subtask in the HASP region.

The program HASPWTR depends on OS Queue Management structures (QCR, LTH, SMB, no-work ECB) as documented in the OS/VS2 Job Management PLM. Functions (such as enqueue, dequeue, or delete a job; ENQ/DEQ to control access to Queue Management resources; conversion of record addresses between NN, TTR0, and MBBCCHHR forms; and computation of sector numbers when SYS1.SYSJOBQE is on an RPS direct-access device) are all performed in a manner consistent with that described for the standard OS Job Management modules.

Microfiche listings for Queue Management functions were consulted as examples during the development of HASPWTR. However, no actual Job Management modules are executed by HASPWTR.

On initial entry after being attached, the program saves three addresses passed to it by HASP initialization: storage address of the pseudo 1403 UCB designated by the HASPGEN parameter &WTR, address of a HASP subroutine to be called to signal end-of-job, and address of an ECB which will be posted by HASP if the operator enters the command $PHASP. After signaling HASP (via a POST) that ATTACH was successful and setting up addressability to the OS Queue Manager resident DCB and DEB for SYS1.SYSJOBQE, the program enters its major processing loop.

The major processing loop is driven by inspection of a list of ECBs. One is the $PHASP ECB which, if posted, causes the program to terminate as described later. All other ECBs are each part of an 8-byte no-work element. One such element is present for each SYSOUT (MSGCLASS) to be processed, as indicated by the list of classes assigned to the HASPGEN parameter &WTRCLAS. If an ECB is posted, the Queue Control Record (QCR) for that class is read and a job is dequeued, if present. The dequeued job's last logical track header (LTH) must be read to perform the dequeue. The updated QCR is rewritten. If there were no jobs to dequeue, or if the one dequeued was the only one, the class ECB is cleared and the no-work element is chained from the QCR before rewriting.

If a job was dequeued, its SMBs are read, messages are formatted into print lines, and the lines are "printed" to HASP using the pseudo 1403 UCB. If non-SMB blocks such as Data Set Blocks (DSBs) are encountered, they are simply skipped. The data sets they represent are not printed or scratched. When the end of the job is reached, a small subroutine in HASP is called to signal end-of-job to HASP.

The HASPGEN parameter &WCLSREQ controls the disposition of the job after processing. If the position in the list &WCLSREQ, corresponding to the position of the job's message class in the list &WRTCLAS, is a valid SYSOUT class, then the job is requeued in the QCR for that new class.

Any tasks (e.g., other system writers such as TSO) whose no-work elements are chained from that QCR are posted. The requeue action always places the job in the new QCR chains at highest priority.

If &WCLSREQ does not indicate requeueing ("*" in a list position instead of a class), the job's tracks in SYS1.SYSJOBQE are released by chaining them to the chain of free space beginning in the Master QCR, posting any tasks waiting for job queue space, and rewriting the Master QCR.

The major processing loop is repeated until no ECBs are found posted. An OS multiple WAIT is executed and, when any ECB is posted by another task (usually an OS Job Terminator), the major processing loop is resumed.

If the operator enters $PHASP, HASP will post an ECB to signal termination actions to this program. All QCRs for processed classes (&WTRCLAS) are read, the no-work chain of each is zeroed, and the QCR is rewritten. HASPWTR exits with a zero completion code.

If permanent I/O errors occur during any I/O on the SYS1.SYSJOBQE data set, an error message is always written to the operator. For write operations, no further special action is taken and processing continues. For read operations, an attempt is made to minimize system damage. No input from an incorrect read is ever used for processing. If the error occurs in reading a QCR or LTH while attempting to dequeue a job, the ECB is set so that no further processing of that class will be attempted. If there is an SMB read error, end-of-job is signaled to HASP and no further blocks on that job's chain are read. If a QCR read error occurs during a requeue attempt, the job is deleted (tracks are released).

HASP SMF SUBTASK

The function of the HASP SMF Writer (HASPACCT) is to look at the $SMFBUSY queue, take buffers off the queue, call IEFUJP exit if necessary, interface with the OS SMF writer, and place freed buffers on the $SMFFREE queue.

The HASPACCT routine runs as a subtask of HASP, even though it is a part of the HASP load module. When the program is entered, one main loop is begun. The $SMFBUSY cell in the HCT is interrogated. If zero, the $STATUS cell in the HCT is checked to see if a $PHASP command was issued. If so, the HASPACCT subtask exits with a zero completion code. Otherwise, the program WAITs to be POSTed by HASP again. If $SMFBUSY was nonzero, the first SMF buffer is removed from the busy chain. When the buffer contains an SMF record, the SMFWTM macro is issued immediately to write the record. If the buffer contains a copy of the common exit parameter area (JMR section of a JCT), a parameter list is prepared for the IEFUJP user exit. The parameter list pointed to by general register 1 when IEFUJP is called consists of two full words. The first word is the address of the common exit parameter area, and the second word is the address of the SMF RDW of the type-26 purge record. The IEFJUP routine is then entered. If, upon return from the IEFUJP routine, register 15 contains a four, the type-26 record is not written. Otherwise, the SMFWTM macro is issued to write the SMF record.

Next, the SMF buffer is placed on the $SMFFREE queue. If there was a JMR buffer in addition to the SMF record buffer, it is also freed. If $SMFFREE was zero, HASP is $POSTed for SMF. Then the routine loops back to check $SMFBUSY for other SMF records in the same manner as described above.

The HASPACCT CSECT is null if the HASPGEN parameter &NUMSMFB is less than two. In this case, no SMF recording code is generated within HASP. The HASPACCT subtask is IDENTIFYed and ATTACHed by HASPINIT and DETACHed by HASPCOMM if HASP is stopped. The program is activated by an OS POST when any HASP routine is ready to write a HASP SMF record.

HASP SETPRT SUBTASK

The purpose of the HASP SETPRT subtask is to issue the SETPRT SVC to set up 3800 printers. There is only one SETPRT subtask which services all 3800 printers. The SETPRT subtask is invoked by the $XFER Marco. When $XFER is issued with the TASK=STPT option, then the next sequential instruction following the macro expansion will be executing under the SETPRT task. $XFER TASK=HASP is used to return to the HASP task.

The SETPRT task has a dispatcher similar to the HASP dispatcher. In fact, the elements being dispatched are PCEs. When a program $XFERs from HASP, the $EWFXFER bit is set in the PCEWAIT field. The PCE is no longer dispatchable in the HASP task. The SETPRT task is posted if it is necessary, and the only PCEs eligible to run are those with $EWFXFER set.

SETPRT is issued in a separate task, because there are implied waits within the SETPRT SVC which would seriously affect performance.

HASP INPUT SERVICE PROCESSOR

The functions of the Input Service Processor are as follows:

1. To read card images from an input device
2. To detect and scan JOB cards, extracting parameters for job accounting, job control, and print and punch identification
3. To detect and process other control cards, such as the COMMAND, PRIORITY, ROUTE, SETUP, MESSAGE, JOBPARM, OUTPUT, DD *, and DD DATA cards
4. To assign a unique HASP job number to each job
5. To log jobs into the HASP System
6. To assign job priority based upon PRIORITY card, JOB card, or JOBPARM card parameters
7. To generate, from cards read, a JCL file and input data files, and to record these files on direct-access storage device(s) for later use by the Execution Control Processor
8. To generate HASP Job Control Tables, Input/Output Tables, Job Queue Entries, and other HASP control blocks required for later job processing
9. To queue jobs for processing by the Execution Control Processor.

The Input Service Processor is coded reenterably in such a way that it can accept jobs from a number of different input devices (with different hardware characteristics) simultaneously. The reenterability is attained by retaining all storage unique to a job in the Processor Control Element which must be unique for each input device.

The Input Service Processor is divided into three phases, 14 subroutines, and two nonprocess exits. This section will give a functional description of each of these phases, subroutines, and exits to aid the System Programmer in gaining a working knowledge of the processor.

PHASES

Phase 1 - Processor Initialization

The Initialization Phase, which is written as an overlay segment, begins by attempting to acquire an input device. If no input device is available, the processor is placed in a HASP $WAIT state until a device is made available, whereupon the procedure is repeated until an input device is available. Upon acquiring an available input device, the processor continues by acquiring a Device Control Table (DCT) for the direct-access device(s) and initializing both DCTs for processing.

If the input device is a card reader, a HASP buffer is obtained for use as an input buffer and a chain of Channel Control Words (CCWs) is constructed in the input buffer which will be used to read 80-byte records from the input device into the rest of the input buffer. These CCWs are constructed in such a way that the input records will be read into adjacent areas in the input buffer with as many cards being read as the buffer will hold. The initialization of the PCE work area is then completed and control is transferred to Phase 2.

If the input device is a remote terminal, transmission is initiated by calling upon the Remote Terminal Access Method to open the Remote Terminal Device Control Table. Control is then passed to Phase 2.

Phase 2 - Main Processor

The Main Processor Phase reads cards from the input device, scans each card to detect HASP control cards, and processes these cards as follows:

1. /*control card--The appropriate control card scan routine (HASPRCC1 or HASPRCC2) is called to process the control card and take any appropriate action.

2. JOB card--The JOB card scan routine (HASPRJCS) is called to terminate the previous job (if any), to scan the JOB card, and to initiate the processing of the following job.

3. DD * or DD DATA--A track address is obtained for the first data block of the input data set. A dummy card is added to the JCL file which contains the track address in columns 1-4. This card is differentiated from other cards by setting the control byte. The DD * or DD DATA statement is then added to the JCL file in normal fashion. Control is subsequently returned to the main processor for processing of the input data.

When a hardware end-of-file is detected on the input device, control is given to Phase 3.

Phase 3 - Processor Termination

Upon receiving control from the Main Processor, the Processor Termination Phase, which is written as an overlay segment, terminates the last job (if any), frees the input buffer if one is present, closes the input DCT if it is for a remote device, releases the input and direct-access DCTs, and returns control to Phase 1.

SUBROUTINES

HASPRCC1 -- Subroutine To Process HASP Control Cards

The HASPRCC1 subroutine, which is written as an overlay segment, is called whenever the Main Processor Phase encounters a HASP control card of any of the types listed below. The control card type is first determined, and then processing continues as follows:

1. COMMAND Card -- The command is listed on the operator's console and then is added to the Command Processor's input command queue.

2. PRIORITY Card -- The previous job (if any) is terminated, the priority specified is converted to binary and saved, and the scan is continued with the next card. If the following card is not a JOB card, the message, "device SKIPPING FOR JOB CARD", is written on the operator's console, the effect of the PRIORITY card is nullified, and the input stream is scanned for another PRIORITY or JOB card.

3. ROUTE Card -- The appropriate routing byte is set to the value associated with the destination indicated. If an invalid field is encountered, an appropriate message is issued, both to the operator and to the programmer, and further job processing is bypassed.

4. SETUP Card -- The volume(s) to be mounted are listed on the operator's console, and the job is placed in "hold" status.

5. MESSAGE Card -- Leading and trailing blanks are removed, and the message is routed to the operator's console.

HASPRCC2 -- Subroutine To Process JOBPARM And OUTPUT Control Cards

The HASPRCC2 subroutine, which is written as an overlay segment, is called whenever the Main Processor Phase encounters a JOBPARM or OUTPUT control card. The control card type is determined and processing continues as follows:

1. JOBPARM card -- The KEYSCAN subroutine is called to scan the parameters coded on the control card.

2. OUTPUT card -- The KEYSCAN subroutine is called to scan the parameters coded on the control card. The destination fields (if any) are then scanned and converted to binary route codes. Finally the output table, which is in fact an Output Control Record (OCR), is added to the JCL file for further processing by the Execution Control Processor.

HASPRJCS--Subroutine To Initialize Job Control Information

The HASPRJCS subroutine, which is written as an overlay segment, is called whenever the Main Processor Phase encounters a JOB card. The previous job (if any) is terminated by calling the RJOBEND subroutine. The master job number is incremented, and its new value is assigned to the current job. A Job Queue Element is then created for this job and added to the HASP Job Queue in active input status. A buffer is obtained and the Job Control Table is initialized. The JOB statement is scanned, the first JCL block is initialized, and if the HASPGEN parameter &RJOBOPT indicates that an accounting field scan is to be performed, control is passed to the HASPRSCN overlay for accounting field interpretation. Otherwise, control is returned to the caller through the overlay return facility.

RJSCAN -- Subroutine To Extract Fields From JOB Statement

This subroutine scans the JOB statement and extracts fields for further processing. The fields may be split between several cards (in accordance with OS JCL standards), and may be enclosed in either parentheses or apostrophes. This routine is used to scan the accounting field and the programmer name from the JOB cards which are processed by the Input Service Processor.

RCONTNUE -- Subroutine To Read And Validate Continuation Cards

This subroutine reads and validates JCL continuation cards by ensuring that columns 1 and 2 are punched with slashes and that column 3 is blank. The start of the continuation card is located and control is returned to the caller. If an invalid continuation card is discovered, control is passed to the illegal JOB card routine for further processing.

HASPRSCN -- Subroutine To Scan Accounting Field Of JOB Card

The HASPRSCN subroutine, which is written as an overlay segment, is called whenever a JOB card is processed to interpret any variables present in the JOB card accounting field and to set the appropriate fields in HASP control blocks representing these variables. The contents of the accounting field are scanned from the JOB card(s) by the HASPRJCS routine and left in the Job Control Table (JCTWORK) as input to this subroutine. Depending on the value of the HASPGEN parameter &RJOBOPT, this routine may or may not enforce certain HASP and/or OS JOB card standards during the scan. Upon completion of the scan, control is returned to the caller through the overlay return facility.

RJOBEND -- Subroutine To Complete Job Input Processing

This subroutine tests whether the Input Processor is currently processing a job, and if it is not, returns control immediately. If a job is being processed, the RJCTTERM subroutine is called to complete processing of the JCT. The execution priority of the job is then

determined, the RJOBTERM subroutine is called to terminate the input processing of the job, and the job is queued for the Execution Control Processor in the logical queue associated with the job class. Control is then returned to the calling routine.

RGET -- Subroutine To Get Next Card

This subroutine returns the address of the next card to be processed by the Input Service Processor in register RPI. If the input device is a card reader and if the input buffer is empty or if all the cards in the input buffer have been processed, an IOS read is staged from the input device and the subroutine places the processor in a HASP $WAIT state until the input buffer has been filled. If the input device is an Internal Reader, the associated caller is posted to indicate completed processing of the last card and the Input Service Processor is $WAITed until the next card is provided. If the input device is a remote terminal, a "call" is made on the Remote Terminal Access Method to obtain the next card. If a permanent error is detected on the input device, no action is taken until after the last card has been processed, and then the job currently being processed is deleted with appropriate comments to the operator. Processing then continues by scanning the input stream for the next JOB card.

This subroutine also processes the operator commands "$Z ($STOP) input device" and "$C ($DELETE) input device" by entering the HASP $WAIT state and calling the subroutine RJOBKILL to delete the job, respectively.

There are two returns from the subroutine. If return is made to the first byte following the Branch and Link (the call) instruction, it indicates that the last card has been processed and that an end-of-file has been sensed on the input device. If return is made to the fourth byte following the Branch and Link, it indicates that register RPI contains the address of the next card.

RPUT -- RPUTOLAY -- Subroutine To Add Card To Output Buffer

This subroutine accepts 80-byte card images and blocks them into standard HASP Data Blocks. If the current output buffer is full, it is truncated and scheduled for output, and a new HASP buffer is acquired and used as the next output buffer. If no output buffer exists upon entry, the processor is skipping for a JOB card, and the subroutine returns without taking any action.

RKEYSCAN -- Subroutine To Scan And Process Keyword Values

This subroutine accepts a pointer to a parameter field and the address of a control card scan table (keyword table) and returns with the field specified by the next parameter updated according to the value(s) specified in the parameter field and the conversion indicated in the table. If an error is encountered, control is returned to the location specified by Reader Link Register 1 (RL1). If no error is encountered,

control is returned four bytes beyond the location specified in RL1. The control card scan table specifies such characteristics as keyword name, keyword abbreviation, whether the value should be converted to binary, whether the value should be left- or right-adjusted in the field, whether the value should be filled with blanks or zeros, and the maximum number of values permitted. This subroutine is used by both the JOBPARM and OUTPUT card processors to perform their respective scans.

RJOBKILL -- Subroutine To Delete Current Job

This subroutine tests whether the input processor is currently processing a job, and if it is not, returns control immediately. If a job is being processed, the operator is notified that the job is being deleted, the RJCTTERM and RJOBTERM subroutines are called to terminate the input processing of the job, and the job is placed in the output queue for subsequent processing. Control is then returned to the calling program.

RJCTTERM -- Subroutine To Terminate JCT

This subroutine performs the final update of the Job Control Table. The time estimate is converted from minutes to seconds, the estimated line count is converted from thousands of lines to actual lines, and the total output estimate is computed and set. Control is then returned to the calling program.

RJOBTERM -- Subroutine To Terminate Job

This subroutine first terminates the last input stream data set if not already terminated. Two JCL null statements are added to the end of the JCL stream to force the reader-interpreter to terminate and queue the job when the stream is later being processed. A message is added to the end of the JCL file which will be printed only in the event that the job is cancelled before it is processed by the Execution Service Processor. The JCL file is then terminated and the last buffer is scheduled for output. The JCT is updated to reflect end-of-job status, the IOT is created, and both blocks are written to disk. Control is then returned to the calling program.

RGETBUF -- RGETBUFO -- Subroutine To Initialize Output Buffers

This subroutine acquires a HASP buffer for an output buffer, initializes this buffer with a chain track address and the job's data set key, and returns with the address of the buffer in register R1.

NONPROCESS EXITS

The following routines are used to put the Input Service Processor in a HASP $WAIT state if a HASP resource is not available. In all cases

Reader Link Register 2 (RL2) must have been set to the restart address before the routine is entered.

1. RNOCMB - A HASP Console Message Buffer was not available.

2. RNOJOB - The HASP Job Queue was full and a new entry could not be added.

When the respective resource is available, the processor is $POSTed and another attempt is made to acquire the resource.

HASP EXECUTION PROCESSOR

The assembly listing of HASPXEQ contains several HASP processors the Execution Processor, the Execution Thaw Processor, and the HASP Job Log Processor. In addition to processors, the listing contains the EXCP exit, the reader-interpreter exit, and the job and step termination exit.

The Execution Processor (XEQ) selects from the HASP execution job queue a job for OS execution, passes it to OS for execution, services the job's SPOOLing requests, and, at termination of the job, enqueues it on the HASP output queue.

The Reader/Interpreter exit analyzes JCL statements (passed by the Reader/Interpreter in internal-text form) and modifies various JCL parameters to process the job's data sets. The HASP Reader/Interpreter exit operates under the OS Reader/Interpreter TCB.

To allow HASP to process requests for pseudo-unit I/O, the OS Input/Output Supervisor gives control for each pseudo-unit EXCP to the HASP EXCP exit, $EXCPSVC.

For a pseudo-I/O request, $EXCPSVC may be used to delay the requestor in order, for example, to read another buffer of SYSIN data. The Execution Thaw Processor, XTHAW, releases the caller to allow him to reissue his EXCP request.

To support the HASP Job Log of WTO and WTOR messages and replies provided on a user's output listing, the Log Processor scans HASP Console Message Buffers and writes as appropriate to the HASP Job Log data set.

The job and step termination exit provides certain information about each HASP job to the OS System Management Facility.

XEQ controls each OS job submitted via HASP throughout its HASP execution phase. (HASP execution starts when a job is selected for the OS/HASP reader-interpreter and ends when the last of the job's System Message Blocks (SMBs) has been read by HASPWTR from the OS job queue.)

Chronologically, a job proceeds through three distinct phases in XEQ. First, a job is chosen from the HASP execution job queue. The choice is based on the priority of the job in the HASP job queue, the job's class (taken from the CLASS= parameter of the JOB card), and Execution Batch Monitor considerations. The Execution Batch Monitor is an installation option.

In the second phase of HASP execution, XEQ provides SPOOL and special services for the job, based mainly on requests from $EXCPSVC. While the job's JCL is being interpreted by OS, XEQ will service the job's JCL data set, allowing $EXCPSVC to supply JCL card images to the reader-interpreter. While the job is in OS execution, XEQ will service the job's SYSIN and SYSOUT data sets. After the job ends OS execution and has been retrieved by HASPWTR from the OS output job queue, XEQ will service the job's SMB data set, collecting SMBs to be printed on the user's output listing.

The last phase of HASP job execution is job termination. Resources used by XEQ to control the job are freed and the job is queued for output or, if the operator requested it, for reexecution.

To these three phases correspond three distinct sections of XEQ: job initiation, DDB services, and job termination.

JOB INITIATION

The object of job initiation is to select a job for execution. The means of selection are the Partition Information Tables (PITs). Control of job selection is provided by an execution Processor Control Element (PCE). In a typical HASP System there are as many PITs as execution PCEs.

A PIT defines a HASP logical partition and looks to the operator nearly like an OS initiator (he can change its job selection classes, start it, and stop it). When a PIT becomes free to be used in job selection, the operator receives the message INIT nn IDLE. When a job has been selected for the PIT, the operator receives the message JOB jjjj -- jobname -- BEGINNING EXEC - INIT nn - CLASS c.

To select a job for execution, XEQ examines each PIT. If the PIT is not busy and not drained (stopped by the operator), XEQ attempts to select a job for it. The PIT contains, among other things, a string of job classes; any job whose class matches one of the classes in a PIT is eligible to execute for that PIT. Starting with the first class in PITCLASS, XEQ issues a $QGET to attempt to select a job. If that selection fails, XEQ repeats the process with subsequent classes until either a blank class (X'40') is found or a job is selected. If a blank class is found, this PIT cannot select a job; XEQ examines the next PIT to see if a job is available for it. If no PIT can select a job, XEQ $WAITs for JOB.

During this time, XEQ is running under a single execution PCE. Any other execution PCE entering the job selection logic will find byte XPITSENQ set to X'FF' and will $WAIT for work, to be posted when the PITs are again available to be searched.

Having selected a job, XEQ sets the PIT's busy flag and increments the active-processor count. In preparation for reading the job's HASP Job Control Table (JCT), XEQ gets a direct-access Device Control Table (a DCT is the argument to $EXCP) and a HASP buffer. XEQ calls $EXCP to read the JCT and $XCTLs from the first overlay of job initiation to the second, where it $WAITs for I/O.

When posted, XEQ verifies that the JCT was read without error and that it is valid; if either is not true, the job is queued for the HASP Purge Processor.

XEQ then checks for another job in HASP execution with the same job name. Since HASP will need to associate an executing job with its execution PCE by job name, it can let only one job with a given name run

at a time. If this job's name matches the name of a job already running, XEQ requeues it for execution but sets a special hold flag in its Job Queue Element (flag QUEHOLD2) and a flag in the execution PCE running the identically named job; XEQ then attempts to select another job.

If this job passes the duplicate-name test, XEQ gets another HASP buffer and reads the job's Input/Output Table (IOT). If the IOT was read without error and is valid, XEQ will get output tracks for the JCT and IOT and write them.

When XEQ selects a job, the JCT and IOT reside on input tracks, that is direct-access addresses allocated by HASPRDR from the job's input track group allocation map. (This map is created and saved by HASPRDR in the JCT). The input track group allocation map (input map) is responsible in addition for allocating space for the JCL file and files for input stream data sets; this space will be freed when the job ends execution. All other SPOOL direct-access space allocated to the job is allocated from the output map, which is in the IOT. This map is first used by XEQ to get output tracks for the JCT and IOT, and continues to be used as the job creates SYSOUT data sets.

XEQ gets an output track for the IOT and puts the track address in both the IOT and the JCT. It gets an output track for the JCT, puts the track address in the IOT (to be used to validate the IOT) and the DCT, and writes the JCT. If the write fails, XEQ gets another track and retries the write. When the write succeeds, XEQ puts the JCT track address in the JQE and causes a HASP checkpoint to be taken.

In the third overlay of job initialization, XEQ sets up a Data Definition Block (DDB) to be used to read the JCL file. A DDB is the pseudo unit interface between $EXCPSVC and the HASP DDB service routines; in this case, the pseudo unit is a HASP-2540R with a unit address of &RDR (default 0FC) The first track address of the JCL file is contained in the only Peripheral Data Definition Block (PDDB) in the IOT. A PDDB is a description of an output data set; if this job had been cancelled before execution, its JCL file would have been printed since the HASP Output processor would interpret the job's IOT's PDDB as defining an output data set.

Once the DDB for the OS Reader/Interpreter has been set up, XEQ moves into the IOT two dummy PDDBs, to be used for the HASP Job Log and system messages data sets. XEQ then stores into the execution PCE the address of the Reader/Interpreter TCB, to be activated to read this job, and sets XPITSENQ to zero to show that the PITs are again available for scanning.

Unless the user (or the installation) has requested no HASP Job Log, XEQ gets a DDB for the HASP Job Log and initializes it. XEQ stores the address of the HASP Job Log PDDB into the HASP Job Log DDB so that, when the first track address is gotten for the log, the PDDB can be located quickly; the track address will then be stored in the PDDB.

XEQ now writes to the operator a message that the job is beginning execution and enters the DDB service routines via a $RETURN.

DDB SERVICE

To accomplish SPOOLing of input and output data sets, a pseudo unit must be correlated with records on direct-access devices; this correlation is accomplished by the DDB. For every SPOOLed data set actively being used by a job there exists a DDB; these DDBs are chained from the job's execution PCE. When $EXCPSVC finds it cannot process a job's pseudo-I/O request, it makes the job's TCB nondispatchable and invokes DDB service. The DDB service section of XEQ also contains special routines needed when a service must be performed synchronously with other HASP PCEs.

DDB service inspects each DDB on the PCE. A DDB may require no service, I/O service, data set termination service, or error message service. I/O service includes buffer service.

When a DDB needs I/O service, it is put at the top of a special table to indicate that it is the most-recently serviced DDB of all DDBs in the system. If it is an input DDB, that is associated with a HASP-2540R, service consists of getting a primary or secondary buffer (if required) and initiating a read to that buffer. If it is a previously-opened output DDB, service consists of truncating an output buffer, getting a track address to which the next output buffer will be written, putting that track address in the current buffer, and writing the current buffer. Another output buffer is gotten and in it is saved the track address to which it will be written.

For an unopened DDB, an initial track address is gotten and put into the PDDBs that describe the output data set. (There will be more than one PDDB only if more than one DEST was specified on the /*OUTPUT control card.) Then DDB service flags the modified IOTs to be checkpointed and gets a buffer as above.

DDB termination service processes both input and output DDBs. For an output DDB, the current buffer is truncated, its track chaining address is made zero, and it is written. The DDB is dechained from the PCE and added to the pool of free DDBs. All execution PCEs are $POSTed by DDB service for DDB.

For an input DDB for which no I/O is active, DDB service frees primary and secondary buffers and frees the DDB as above.

DDB error message service informs the operator that an I/O error has occurred and terminates the job using the OS cancel command.

DDB service also performs special services for the job as a whole. There are several job services, requested by $EXCPSVC and by the HASP exit from the OS reader-interpreter. The services write messages to the operator to inform him of various normal or abnormal conditions and get additional working storage for the job. Invocation of a job service is dependent on flag bits in the PCE.

$EXCPSVC gets a DDB for a SYSIN data set when it recognizes a special control card inserted by HASPRDR in the JCL file; it gets a DDB for a SYSOUT data set when the job, in OS execution, first issues an EXCP to the data set's pseudo unit. If $EXCPSVC fails in its attempt to get a DDB, it cannot continue without help from HASP. It freezes the caller (makes his TCB nondispatchable) and invokes the awaiting-allocation service. That service informs the operator and $WAITs for DDB. The operator can be informed many times in the course of a job that $EXCPSVC is awaiting HASP allocation.

$EXCPSVC similarly recognizes another type of control card in the JCL file as an Output Control Record (OCR), a condensation of a /*OUTPUT control card, which it must save throughout the Reader/Interpreter phase of job execution. It uses a HASP buffer as a work area in which to save these OCRs. If no OCR work area exists (the pointer in the PCE is zero), $EXCPSVC invokes a job service to get such a work area; then it moves the OCR into the work area. But if an OCR work area already exists and is so full of OCRs that another OCR will not fit, $EXCPSVC invokes a job service to write to the operator a message that the job uses too many /*OUTPUT control cards. The message will appear once at most for a job.

For each SYSOUT statement encountered by the HASP Reader/Interpreter exit, one or more PDDBs are created. A PDDB is variable in length; its length depends on the complicity of the DD statement and of its associated OCR, if any. PDDBs are packed one by one into an IOT, so the possibility exists that there will not be room for a next PDDB. If this happens, the HASP Reader/Interpreter exit invokes the job service, which creates a new IOT, and does an OS WAIT. This service gets another HASP buffer, for use as an IOT, and an output track address, to which the IOT will be written, and chains the new IOT to the chain of IOTs with both storage and track addresses.

when the job ends OS execution, the OS initiator queues the job in the OS output queue, where it is picked up by HASPWTR. As HASPWTR processes the job's SMBs, the job remains in the HASP execution phase. When it completes its processing, HASPWTR posts HASP and invokes the job end service. This service issues the message JOB nnnn END EXECUTION and marks all DDBs except the HASP Job Log DDB which will be marked for termination when the HASP Log Processor recognizes the END EXECUTION message.

The DDB service section examines requests in this order: first, requests for special job services, then requests for service from each DDB. Having completed services for all DDBs, it then checks a flag to see if any IOT needs to be checkpointed. An IOT must be checkpointed when one of its PDDBs has newly received an initial track address for an output data set.

Checkpointing an IOT consists of writing it out at its assigned track. If an I/O error occurs, another track is gotten for the IOT, the track address is put into the IOT and, for chaining, the previous IOT (or the JCT) and both control blocks are rewritten.

At its conclusion, DDB service posts the Execution Thaw Processor (whose purpose it is to make again dispatchable any task frozen by $EXCPSVC) and waits for work. But if the end-of-job flag is on in the PCE, and if the PCE has no DDBs on its chain, then DDB service gives control to the third phase of XEQ, job termination.

JOB TERMINATION

Having completed HASP execution, a job must be queued for output. Alternatively, if required by the operator via the $EJ command, it may be requeued for execution. In either case, resources no longer required by the job must be freed for use by other jobs.

The third phase of XEQ, job termination, determines by a flag in the PCE whether the job is to be reexecuted or not. If the job is to be reexecuted, its currently-collected output is to be purged; job termination calls the $PURGE service to do this. Job termination then moves to the JQE the track address of the input JCT; the address had been saved in the JCT by HASPRDR. It frees all buffers in use by the job, and it queues the job again for execution.

But if the job is not to be reexecuted, job termination purges instead its input data (input JCT, input IOT, JCL file, and SYSIN data sets) with a call to $PURGE, recomputes (if necessary) its HASP priority based on number of print lines and punched cards and an installation - defined table, rewrites the JCT and IOTs on their assigned output tracks, and queues the job for HASP output service.

In either case, if the name of the job now ending caused an identically-named job to be held, job termination resets in each active JQE the special hold flag QUEHOLD2, $POSTs for JOB all execution PCEs, and causes a HASP checkpoint to be taken.

Finally, XEQ clears the work area in its execution PCE, reduces the active-PCE counter by one, and $RETURNs to attempt to select another job.

THE EXCP EXIT ($EXCPSVC)

For HASP SPOOLing service to provide SPOOLed input to and receive output to be SPOOLed from a job, it and the job must agree on a common means of requesting service. This common means is the pseudo UCB, created by SYSGEN at the request of the installation. The HASP exit from the OS Reader/Interpreter modifies the internal text of a JCL statement to cause the initiator to allocate one of these pseudo UCBs to each of a job's data sets that is to receive SPOOLing service. I/O activity of the job on such a data set uses a normal OS Sequential Access Method (SAM), whose end-of-block routine issues an EXCP to the UCB, with an appropriate channel program, just as if the pseudo UCB represented a real 1403 Printer, 2540 Reader, 2540 Punch, or (for jobs using the HASP Internal Reader feature) a 2520 Punch. Each pseudo-UCB is flagged as such; upon EXCP, IOS recognizes the flag and routes the request to

$EXCPSVC, the HASP EXCP exit. IOS validates the request before it sends it on, and continues to provide paging services while $EXCPSVC operates. ($EXCPSVC runs totally disabled.) When such a page fault occurs, IOS initiates page-in of the required page and backs up the EXCP caller's SVC old PSW. When the page fault has been resolved, IOS will reenter $EXCPSVC at its start to reinitiate processing for the request.

It is the purpose of $EXCP to process the channel program associated with a user I/O request, simulating the I/O device represented by the pseudo UCB. IOS does not create an RQE for a request for pseudo I/O; therefore $EXCPSVC must keep a record of the progress of the request, specifically, which CCW it is currently processing and whether or not that CCW has been completed. This information, necessary because of the possibility of page faulting, is kept in the IOB's CSW and restart-CCW fields; the validity of these fields is indicated by the IOB's restart flag.

For this reason, on first entry for a request, $EXCPSVC moves the starting CCW's address to IOBRESTR, clears IOBCSW, and turns on the restart flag. At this time the SMF EXCP exit is taken.

A test occurs next for Internal Reader processing. If the UCB represents an Internal Reader, and if the Internal Reader is represented by a DCT (the only type of DCT which does not represent a real device), then the user has provided an 80-column card image which is to be moved into a data area in that Internal Reader DCT for further processing by HASPRDR. $EXCPSVC ensures that a reader PCE is active for this DCT, freezing the caller and invoking HASP if necessary, and that the data area in the DCT is ready to receive input; then it moves the card image to the DCT, $POSTs the reader PCE for I/O, and posts HASP. This ends the request unless the channel program contains more than one write CCW, in which case the caller is frozen (to be activated again by HASPRDR).

If the EXCP request is not for an Internal Reader, it is either for the OS reader-interpreter, for HASPWTR, or for a problem program. If the request is from the OS reader-interpreter, the execution PCE address and DDB address associated with the request are available directly in the normal case. In other cases the reader is to be frozen pending selection of another job by XEQ for Initialization, or the HASP System is stopping and the OS reader-interpreter should be stopped; $EXCPSVC returns unit-execution in IOBCSW to cause the reader to stop.

If the EXCP request is from the writer, either the execution PCE and DDB addresses are available directly, or HASPWTR is just now writing a job's JOB card to its pseudo unit. In the latter case, the job name from the JOB card is compared to the job name in each execution PCE to find the proper PCE, and end-of-step logic in entered.

Regardless of the source of the request for pseudo I/O, the associated execution PCE must be located, and the associated DDB must be found or a DDB must be created. A subroutine named XJOBSRCH is used to find the PCE; it does this by examining first JSCHPCE in the job's JSCB. If this field is zero, the subroutine compares the job's name in the TIOT with

job names in the execution PCEs. It returns a condition code indicating how the PCE was found.

If the PCE was found from JSCHPCE, no job step change has occurred. (The initiator zeros JSCHPCE at the beginning of each job step). But if a step change has occurred, $EXCPSVC flags for termination each DDB that was active on behalf of the previous step; then, it stores into the PCE the current step number indicated by the initiator's LCT.

Having found the PCE, $EXCPSVC must next locate the DDB. It runs the chain of DDBs comparing the unit address in each with the unit address in the pseudo UCB. When the DDB is thus found, $EXCPSVC continues with input or output processing.

If the DDB is not on the PCE chain, the request is for an output pseudo device. The case may be, however, that the DDB is on the chain but does not yet contain a device address (it contains the step number and number of the data set); the request is then for an input pseudo device. In either case, $EXCPSVC must find the step and number of the DD statement associated with the pseudo-I/O request. The step number has been established above; the number is found by counting entries in the TIOT (skipping PGM=*.DD and JOBLIB and its concatenations) until the TIOT offset agrees with DCBTIOT, a field set in the DCB by OPEN. Then, for a pseudo-reader request, the DDBs are again scanned, and the DDB, whose step number and number match the data set's, is chosen.

But for an output data set, the PDDBs are scanned to find the one with matching step and numbers, and a new DDB is obtained for the data set. The PDDB address and the address of the PDDB's IOT are saved in the DDB, so that DDB service can assign an initial track to the data set and save its address in the PDDB.

The DCB is checked for an OPTCD=J option. If OPTCD=J was selected, then that is indicated in the DDB and will be indicated in all PDDBs for this data set.

Now that the PCE and DDB for the data set have been found, the correct I/O routine receives control: XSYSIN if the pseudo-device is a HASP-2540R, or XSYSOUT if it is a HASP-1403 or a HASP-2540P.

XSYSIN processes four types of card images: normal, null, input stream definition, and output control record. The JCL file may contain all four types; all other files processed by XSYSIN contain only normal card images.

Null card images are messages to the user, delivered in the event that his job is cancelled before execution; they are to be passed over.

An input stream definition card is inserted by HASPRDR in the JCL file just prior to a DD * (or DD DATA) card; it contains the initial track address of the input stream data set and causes XSYSIN to get and initialize a DDB for that data set before passing the DD */DATA card to the reader-interpreter.

An output control record card appears at the same place in the JCL file as the /*OUTPUT control card from which it was condensed appeared in the original JCL stream. XSYSIN moves from it to the job's OCR buffer a 68-

byte OCR. Neither the output control record card nor the input stream definition card is passed to the Reader/Interpreter.

A normal card image is passed by XSYSIN to the EXCP issuer according to his channel programs. Using the DDB, XSYSIN points to the proper location in the data set's primary buffer (all input data sets are double-buffered) and inspects the 2-byte logical-record control field. The first byte contains the text length; a length of 255 is used as an end-of-buffer indicator. If the current buffer offset in the DDB points to an end-of-buffer indicator, and if the secondary buffer is ready with data, XSYSIN swaps buffers and proceeds; otherwise it freezes the caller and, if necessary, fires up HASP. But if the primary buffer was not ready, the primary end-of-file switch may be on; if so, the EXCP request is ended with unit-exception in the CSW. If the primary end-of-file switch was off, however, XSYSIN merely freezes the caller and, unless some I/O was active for this DDB, fires up HASP.

If the text length was not 255, a logical record is to be examined. Its type is given by the byte following the text length: X'04' for a null record, X'73' for an input stream definition record, and X'43' for an output control record. Record type X'19' also occurs to describe the last card of a JCL file, which is a null JCL card (to cause the Reader-Interpreter to enqueue the job in the OS job queue before returning for another job and being frozen). To avoid sending end-of-file to the Reader-Interpreter, XSYSIN never will update the DDB's current buffer offset past the null JCL card.

XSYSIN moves a normal card to the address specified by the user's CCW, updates the current buffer offset (except for X'19' cards), sets device-end in the IOB's CSW to tell XCCWSCAN that the CCW has been completely processed, and again invokes XCCVSCAN to get another CCW.

When XSYSIN has processed the last CCW of a channel program, it tests for an end-of-buffer indication before returning to the user. If the EOB indicator is next in the primary buffer, XSYSIN switches buffers and fires up HASP to start I/O on the now-secondary buffer; then it returns to IOS with the IOB's CSW field set correctly.

At end-of-channel program for the Reader/Interpreter, XSYSIN additionally checks for a X'19' card. If this card has just been sent to the reader, the reader is disconnected from the execution PCE it was serving and is frozen.

XSYSOUT processes requests to HASP-1403 and HASP-2540P pseudo devices. Each record presented by the user is truncated of trailing blanks and moved to a HASP buffer. In addition, XSYSOUT edits JCL cards for HASPWTR, changing DD $ to DD * and DD CATA to DD DATA. All SYSOUT data sets are dynamically buffered; that is, whenever a buffer fills with output data it is queued by EXCP for output and a new buffer is obtained. The HASP channel-end appendage and the $ASYNC processor automatically free an output buffer once it has been written.

XSYSIN and XSYSOUT use subroutine XCCWSCAN to obtain the next CCW. XCCWSCAN has been coded with the possibility of page faults in mind. It

maintains current channel program status in the IOB and allows its callers to indicate in the CSW abnormal completion or normal completion of a CCW.

On entry to XCCWSCAN, IOBRESTR points to the current CCW and the CSW shows normal completion, abnormal completion, or no completion. On the basis of this and of the current CCW's command chaining flag, either a next CCW is chosen or the channel program ends. If a next CCW is chosen, XCCWSCAN examines it for validity, processes a TIC (X'8') or a NOP (X'03') and returns to the caller pointers to the CCW and its data area and the contents of the CCW's length field. If no CCW remains or if a CCW chain has been broken, XCCWSCAN returns an end-of-channel-program indication.

READER/INTERPRETER EXIT

XJCLSCAN examines and modifies a JCL statement that is currently being processed by the OS Reader/Interpreter. The JCL statement is passed by the Reader/Interpreter in internal text form. XJCLSCAN's examination and modification has two main purposes: (1) to force the OS initiator to allocate a pseudo device to a data set that requires one and (2) to allow correlation of the data set's characteristics with its pseudo device by $EXCPSVC. To force allocation, XJCLSCAN changes the text of certain DD statements to UNIT=R, UNIT=A, or UNIT=B; to allow correlation to pseudo device, XJCLSCAN preserves the DD statement's step number and DD number in a PDDB it constructs (for SYSOUT data sets) or in the DDB constructed for the data set by $EXCPSVC (for SYSIN data sets).

In addition to processing DD statements, XJCLSCAN makes minor modifications to JOB and EXEC statements.

To the JOB statement, XJCLSCAN adds values for the MSGCLASS=, CLASS=, and PRTY= keywords, eliminating any user-specified values. It also removes a TYPRUN= specification. The job's priority is forced to &PRI(n) and its class to &OSC(n), where n is its HASP logical partition number. If the user-specified message class matches one of the characters in &WTRCLAS, it is retained; otherwise the first character of &WTRCLAS is used as message class. (However, the original user-specified message class will be used as a HASP output class to print the HASP Job Log and system message data sets; if the user specifies no message class, these data sets will be class A data sets.) When processing the JOB statement, the real reader start time and date and the reader device type and class are placed in the Job Management Record (JMR) for the current job.

If the job is to run at the VS2 Automatic Priority Group level, its EXEC statement is modified: the DPRTY= parameter, if specified, is nullified. Also, encountering an EXEC statement causes initialization for the tables used to compute step number and DD number for certain DD statements. A 5-entry table used to record appearances of DDNAME= parameters in the step is zeroed, the current DD number is zeroed, and the current step number is set from the JMR.

Each DD statement is analyzed to see if it defines a SPOOL data set or might refer forward to one. If it defines a SYSOUT data set, one or more PDDBs are constructed for it and its SYSOUT= keyword is changed to UNIT=A or UNIT=B according to installation-defined table XTRTABLE. If it defines a SYSIN data set, the DDB for the data set is found and modified and its positional parameter ($ or CATA) is changed to UNIT=R. (If HASPRDR had allowed * or DATA to be presented through $EXCPSVC to the reader-interpreter, that program would have allocated OS direct-access space for the data set before entering the HASP reader-interpreter exit.) If the DD statement uses DDNAME= to refer to a subsequent DD statement, the subsequent statement might be SYSOUT or SYSIN; information is saved to be used by subroutine XFINDDDN, which supplies step number and DD number to its caller.

To build a PDDB for the SYSOUT statement, XJCLSCAN first extracts from the statement the values for CLASS, FORMS, FCB, UCS, COPIES, FLASH, BURST, CHARS, and MODIFY, placing them in a working PDDB. (But if the character in XTRTABLE corresponding to the class is an asterisk, the DD statement is returned to the Reader/Interpreter unchanged.) The SYSOUT forms field may indicate not forms but OCR ID; if forms are specified, the OCRs are searched to find the matching ID. If an OCR is to expand this SYSOUT definition, the forms-field-present flag in the PDDB is reset and fields present in the OCR are added to the working PDDB. Any values specified in an OCR override corresponding values from a DD statement. If FCB or CHARS are specified, the values are added to the internal text for the DD entry.

With the working PDDB now complete, it is compressed to minimize its size in an IOT and is moved into the IOT that is last on the execution PCE's chain of IOTs; if this PDDB will not fit, HASP is invoked to assign another IOT buffer.

Up to four destinations may be specified on a /*OUTPUT control card. If the OCR indicates that more than one destination exists, the second destination is set into the working PDDB and the PDDB is added to the IOT as above.

Finally, if the SYSOUT statement uses the DCB= keyword, its parameters are examined and, if necessary, modified. The resultant internal text is returned to the reader-interpreter for further processing.

If the DDNAME= keyword appears on a DD statement in the step, XJCLSCAN saves that keyword's argument and the DD number of the statement using DDNAME=. Later, subroutine XFINDDDN will compare current DD name with each saved DDNAME= argument and, upon a match, return the associated DD number instead of the current DD number.

A SYSIN DD statement is modified to contain UNIT=R instead of * or DATA, and step number and DD number are stored in its associated DDB. In addition, it is checked for DCB parameters; if none appear, DCB parameters RECFM=FB, LRECL=80, and BLKSIZE=80 are added.

A general subroutine modifies user DCB parameters as they appear on any SYSIN or SYSOUT statement. If neither LRECL nor BLKSIZE is specified, the subroutine adds no parameters. If LRECL is specified and is greater than a HASP-defined maximum value (80 for SYSIN, 204 for SYSOUT), it is set equal to that maximum. If both LRECL and BLKSIZE are specified, BLKSIZE is set equal to LRECL (or four greater if "V" appears in the RECFM argument). If only BLKSIZE appears, it is retained unchanged. If only LRECL appears, BLKSIZE is added as if both keywords had been coded.

THE EXECUTION THAW PROCESSOR (XTHAW)

During the course of its operation, $EXCPSVC may want to stop its caller temporarily while related DDB services are performed under the HASP TCB. It invokes subroutine XCOOL which in turn invokes the OS status service via branch entry IGC07902 to turn on TCBHNDSP, a nondispatchability bit in the caller's TCB (bit 32.3).

DDT service, having performed all service it can for an execution PCE, will $POST the Execution Thaw processor (XTHAW) for work. It will also flag the execution PCE it is serving so that XTHAW will recognize it. At this time the execution PCE must contain the address of the job's job step task.

Processor XTHAW gains control after all execution PCE's have returned control, from the HASP Dispatcher. It checks a flag (XPOSTBIT) in each execution PCE and, if the flag is on, conveys the job step TCB address to XWARM, the comparison routine of XCOOL. XWARM in turn calls the OS status service via branch entry IGC07902 to reset flag TCBHNDSP in all TCBs of this job step.

In addition, XTHAW thaws HASPWTR if it is frozen and cannot be associated with an execution PCE.

SMF TERMINATION EXIT

XTERMSMF is entered immediately prior to the user SMF exit routine IEFACTRT when HASP is active. The address of this routine exists in the HVT section of the HCT. The routine searches for a match on job name (in the common exit parameter area) against active PCE job names. If no match, the routine returns immediately. If there is a match on job name, the routine saves part of the common exit parameter area in the appropriate JCT for HASP SMF records. If this exit is a job termination exit, the reader stop time and date and the job class from the JOB card are placed into the type-5 SMF record. Then control is returned to the calling routine after register R15 is zeroed to indicate normal return.

HASP OUTPUT PROCESSOR

System output consists of operator console messages, job statistics, messages from the Operating System, and data sets written by the program. Operator console messages are saved in a HASP-defined data set by the WTO/WTOR HASP interface. Job statistics are maintained and updated in the Job Control Table (JCT) by various HASP processors. Messages from the Operating System are written into the system job queue, then copied into a HASP-defined data set at job termination Data sets to be processed as output are placed on the HASP SPOOL volume(s) by the HASP Execution Processor during job execution.

There are 36 classes of system output permitted in the Operating System, and in the HASP Output Processor there is an output queue that corresponds to each class. The programmer defines a data set as being a system output data set by using the SYSOUT keyword in the DD statement; he specifies the class to which it belongs in the parameter associated with the SYSOUT keyword. Using the MSGCLASS keyword in the JOB statement, the HASP Execution Processor assigns a Peripheral Data Definition Block (PDDB) to each SYSOUT data set, as well as one to the console messages and Operating System messages pertaining to his job.

System Message Blocks (SMBs) containing interpreter, allocation, and termination messages are built and placed in entries for the designated queue. If the programmer makes no message class specification, the system uses the class designated as the message class when the system is generated.

Each output queue entry describes the system output, of the corresponding class, for one job. If the entry corresponds to the message class, it contains SMBs and may also contain DSBs if there are system output data sets exempt from HASP processing in that class. If the entry does not correspond to the message class, it contains only DSBs for exempt data sets. At job termination, a system output writer in the HASP subsystem transcribes the SMB data to the HASP SPOOL volume(s) and signals end of job to the appropriate HASP Execution Processor. The job is then transferred from the HASP execution queue to the HASP output queue for analysis by the HASP Output Processor.

The task of the HASP Output Processor is to analyze the PDDBs built for the job by the Execution Processor and to build a set of Job Output Elements (JOEs), which represent unique print/punch requirements. The JOEs are placed in the Job Output Table (JOT), which contains all output requirements currently available to be processed by HASP Print/Punch Processors.

HASP OUTPUT PROCESSOR

At its initial entry point, the HASP Output Processor uses the HASP macro $POST to request a checkpoint of the JOT to the HASP SPOOL volume and to alert all Print/Punch Processors that requests for work may be made. Two "warm start" fields in the JOT are then checked to determine if this is a warm start of a system that was interrupted while adding a job to the JOT. If the fields are nonzero, one will contain the job number of the interrupted job and the other will contain a counter which

denotes the last JOE successfully added. The HASP $QLOC macro is used to locate the job by job number in the HASP job queue. If the job is not found, the HASP $DISTERR macro is used to warn the operator of possible loss of data and the "cold start" path of the Output Processor is taken. Having located the Job Queue Element, (JQE) the Output Processor sets the "entry busy" flag and resumes processing.

If it is determined that no enqueueing of a job has been interrupted, the HASP macro $QGET is used to search the HASP job queue for new work from the output queue. In the event that no job is available, the HASP macro $WAIT is used to release control until the status of some job in the queue is changed.

Having procured a job, the Output Processor uses the HASP macros: $ACTIVE to indicate to the Dispatcher that the system is not dormant, $TIME to get a starting time and date for Output Processor, $GETUNIT to get a control block for accessing data on the SPOOL volume(s), and $GETBUF to get a HASP buffer into which SPOOL data can be read.

The $QGET macro returns the address of a JQE if a job is available for output processing. All references to job data by the Output Processor are made through the JQE, which contains the track address (on the SPOOL volume(s)) of the Job Control Table (JCT). The OPIOCK subroutine is used to read the JCT from the SPOOL volume into the HASP buffer acquired earlier.

A check is made to ensure that the data read from the SPOOL volume is a valid JCT and belongs to the job being processed. In the event of an IO error, the HASP macro $IOERROR is used by the OPIOCK subroutine to augment console data for operator information. A failure during any of the above checks results in the HASP macro $DISTERR being used to alert the operator of possible data loss, and a sequence of macros is executed to release buffer and control block resources held by the Output Processor. $QPUT is used to place the job in the HASP purge queue and $DORMANT is issued to indicate an inactive processor. The Output Processor then returns to the "cold start" entry point and will attempt to find another job.

At normal completion of JCT checks, the job level copy count, message class, and default job forms ID are copied from the JCT to the Processor Control Element (PCE) used as a work area. The JCT contains the track address of the first Input Output Table (IOT) that contains the PDDBs and the track address of the next IOT if additional are required.

The OPIOCK subroutine is used to read all IOTs for the job into HASP buffers before the scan of PDDBs begins. If the validity check for any IOT fails, $DISTERR is issued to alert the operator and processing is continued if at least one IOT was valid. Absence of any valid IOTs results in the same processing path as was taken for an invalid JCT.

Two prototype JOEs are built for the first PDDB that does not have the NULL flag set. The first JOE (work JOE) contains routing information and points to other JOEs of the same SYSOUT class, and the second JOE (characteristics JOE) describes the device setup (FORMS, FCB, UCSB, and, if &NUM3800 is greater than zero, FLASH and BURST)

necessary to process this PDDB. The HASP macro $#ADD is used to add the two prototype JOEs to the JOT in the SYSOUT class queue specified in the PDDB. Each $#ADD done on behalf of the job is done "job copy" times to allow parallel processing by Print/Punch Processors. If the $#ADD macro indicates that the JOT currently has no available space for insertion of JOEs, the HASP macro $WAIT is used to relinquish control until space becomes available.

Having added this processing requirement to the JOT, the NULL flag in the PDDB is set along with the NULL flags in all subsequent PDDBs that represent similar class, route, and setup characteristics. The first PDDB that does not meet the above test is used as a restart point for the next prototype JOE build and $#ADD. When all PDDBs have been set to NULL, the JCT for the job is updated on the SPOOL volume by the addition of Output Processor start/stop times.

For each work JOE added to a class queue, a counter in the JQE for the job is incremented. During processor termination, the job is placed in the purge queue if the JOE counter is zero; otherwise, the job is placed in the hardcopy queue while awaiting print/punch processing. The control block used to access SPOOL data and the string of buffers used in reading the JCT and IOT(s) are returned via the $FREUNIT and $FREEBUF macro instructions. $DORMANT is used to indicate an inactive processor to the HASP Dispatcher, and control is returned to the "cold start" entry of the Output Processor.

JOB OUTPUT TABLE (JOT) ACCESS MACROS

Five macros are provided for accessing entries in the class queues of the JOT:

1. $#ADD - add a JOE to the class queues.
2. $#REM - remove a JOE from the class queues.
3. $#GET - select a JOE for processing by a Print/Punch Processor.
4. $#PUT - release a selected JOE for subsequent processing.
5. $#CAN - remove all nonselected JOEs for a specific job from the class queues.

The subroutines that supply the processing function for each macro can be found in the HASP Output Processor assembly at entry points which are labeled the same as the macro names. Each subroutine uses the $WAIT HASP macro to ensure that the Checkpoint Processor is not currently writing the JOT onto the SPOOL volume, because changes to the JOT during this operation could make the checkpoint record invalid. Subroutines, which require allocation of JOEs from those available in the JOT, determine that the free queue is large enough to ensure completion of the request. If the free queue is too low, a return code indicating "no process" is set, and return is made to the macro caller.

The $#ADD subroutine first scans the queue of characteristics JOEs to determine if a JOE which matches the prototype characteristics JOE being added already exists. If no match is found, a JOE is acquired from the free queue and the setup data from the prototype JOE is copied into it. Having either found a match or built a new characteristics JOE, the use count in the JOE is incremented by one to count the number of simultaneous users of the block. A second free JOE is then acquired from the free queue, and the data from the prototype work JOE is copied into it. The work JOE is added to the top of the appropriate class queue and, if a characteristics JOE was built, it is added to the characteristics queue. $POST is used to invoke a checkpoint of the altered JOT and to alert Print/Punch Processors that work is available, before returning to the macro caller.

The $#REM subroutine decrements the active device counters and use counter in the characteristics JOE associated with the request and returns it to the free queue if the use counter becomes zero. If a checkpoint JOE was assigned, it is returned to the free queue. The work JOE is returned to the free queue, and the JOE counter in the HASP Job Queue Element is decremented. If the JOE counter for the job becomes zero and the job was in the hardcopy queue, it is moved to the purge queue since no further output processing is required. The $POST is used to invoke a checkpoint of the altered JOT and to alert the Output Processor that space in the JOT has become available.

The $#GET subroutine uses the Device Control Table (DCT) of the calling Print/Punch Processor as a parameter list while trying to select the best work element from the JOT for processing.

Each class queue that the device is eligible to process is searched to find the highest priority entry. Calculation of the priority for an element is based on the following factors in order of their importance:

1. Device setup exactly matching characteristics JOE.

2. Characteristics JOE representing a setup not currently found on any other similar local device.

3. A value of 36 decremented as each eligible class queue is searched.

4. HASP job queue priority.

Note: Operator-controlled devices require that item 1 above is true. Automatic local devices require that item 2 above is true unless the forms ID for the JOE is that designated as standard during HASP generation.

If no work is found, a return code is set and control is returned to the macro caller. The work JOE with the highest selection priority is marked busy, and the device use counter in its related characteristics JOE is incremented. A JOE from the free queue is acquired and assigned as a checkpoint JOE in support of "warm start" and interrupt ($I

device). Before returning to the macro caller, $POST is issued to request a checkpoint of the altered JOT.

Note that the $#GET subroutine can be used to determine if work is available without selecting it by using the 'HAVE=NO' operand on the macro call.

The $#PUT subroutine decrements the active device counter of the characteristics JOE to ensure accurate operation of subsequent $#GET calls. If print/punch checkpoint data was supplied in the macro call, it is copied into the related checkpoint JOE and the checkpoint data valid flag set in the work JOE; otherwise the checkpoint JOE is returned to the free queue. After resetting the work JOE busy flag, $POST is issued to request a checkpoint of the altered JOT and to alert Print/Punch Processors of available work.

The $#CAN subroutine scans each of the 36 class queues for work JOEs that belong to to the job supplied in the parameter list. For each work JOE found in the scan, the use counter of the related characteristics JOE is decremented. If the use counter becomes zero, the characteristics JOE is no longer needed and is returned to the free queue. The checkpoint JOE pointer is checked and, if nonzero, indicates that the assigned JOE should be returned to the free queue. Finally, the work JOE is removed from the class queue and returned to the free queue. As each work JOE is removed, the JOE counter in the JQE is decremented. After all available JOEs have been examined, the job is moved to the purge queue if its JOE counter is zero or to the hardcopy queue if nonzero. The $POST macro is issued to request a checkpoint of the altered JOT and to alert the Output Processor of available space in the JOT.

THE JOB OUTPUT TABLE (JOT)

The JOT is located in the Output Processor at entry point $JOT. It contains several queues of JOEs, which describe current system output requirements. The 36 class queues that contain work JOEs, the queue of characteristics JOEs, and the queue of free JOEs are at the beginning of the table with other maintenance data. The bulk of the JOT is then composed of JOEs which are in one of the above queues and whose number is controlled by the HASP generation parameter &NUMJOES.

Each JOE in the JOT can serve one of three functions:

1. Work JOEs are chained in the class queues and are the primary representatives of output processing work. The JOE counter in the JQE counts the number of work JOEs for the job it represents.

2. Characteristics JOEs are chained in the characteristics queue and are pointed to by a field in the work JOE. The setup of a device before it can process a given work JOE is described by a characteristics JOE. Since many work JOEs may require the same setup, a characteristics JOE may be used or pointed

to by more than one work JOE.

3. Checkpoint JOEs are pointed to by a field in the work JOE which is currently being processed by a device or whose processing on a device was interrupted. Data necessary to restart the output process is saved in a checkpoint JOE.

JOEs which are not currently in use to represent output work are chained in the free queue.

Since the data in the JOT is necessary for warm start of the HASP subsystem, a checkpoint is requested each time a change is made. During warm start, a subroutine in the HASP Initialization Processor ensures that all checkpoint JOEs receive appropriate restart data and that all work JOE busy flags are reset.

HASP PRINT/PUNCH PROCESSOR

The HASP Print/Punch Processor is a reenterable program capable of supporting simultaneously multiple local and remote printers and punches. In addition, messages routed to remote terminals which are not available will be saved and transcribed later if the &SPOLMSG option was set at HASP generation. At generation time also, the number of simultaneous users of the Print/Punch Processor is defined by the sum:

&NUMPRTS+&NUMPUNS+&NUMTPPR+&NUMTPPU

Depending on the number of available local devices and the number of lines, this maximum may or may not be reached.

Control flags and work space for each print/punch user are contained in a PCE related to the user and attached to the HASP Dispatcher chain.

All PCEs for the Print/Punch Processor are initially waiting to be informed that JOT services are available. When the $POST is issued by the HASP Output Processor, each print/punch user via the $LINK HASP macro begins execution in overlay module HASPPPI1. The HASP macro $GETUNIT is issued by each processor in turn to get an output device. All local output devices requested by processors will be assigned unless drained by the operator. Remote devices are automatically drained until module HASPRTAM determines that there is work available in the JOT for them. Each Print/Punch Processor which is not successful in getting an output device uses the HASP macro $WAIT to give up control until a unit becomes available. The Print/Punch Processor then stores the address of its event wait field in the DCT for its output device so that subsequent I/O activity can be related to the proper user. $ACTIVE is issued to alert the HASP Dispatcher that the system is not dormant. If the processor is a first remote printer (RMF.PR1), setup is made to print SPOOLed messages if any exist (bypassing normal work selection). All processors which are not printing SPOOLed messages use the HASP macro $#GET to scan the JOT for output work. Unless the operator has altered their setup, all output devices are assumed to be in automatic mode with

standard forms mounted; preference is thus given to jobs which require matching setups.

If work is available, the $#GET macro will return the addresses of the work JOE, the characteristics JOE, and the JQE. If no work is available the processor will issue a message indicating that the device is idle, free the output device DCT, indicate that it is dormant, and wait for JOT work available to be posted. Having obtained a direct-access DCT via $GETUNIT for communication with the SPOOL volume, the processor gets one or two HASP buffers depending on the options chosen during HASP generation. Using the track address in the JQE, the JCT is read from SPOOL and checked. If an error is detected, the work JOE is purged and the processor returns to module entry HASPPPI1.

A successful read of the JCT is followed by a message indicating the job number and device name for operator information. Having initialized the PCE channel program work area, entry is made via a $LINK to the second initialization overlay module HASPPPI2.

SPOOLed remote message processing bypasses most of this module with the exception of one section, which uses the main print loop as a closed subroutine until all messages routed to the device are printed.

Since the JCT is not retained in a HASP buffer during the entire print/punch operation, fields which will be needed are copied into the PCE work area for later use. The processor next scans the Print/Punch Checkpoint element table (PRC) to locate an available slot for recording interim work progress. If a "warm start" is indicated by the work JOE, restart data is copied from the checkpoint JOE into the PRC and $POST is issued to request a checkpoint of system status. Punch Processors next build the punch-laced separator card if requested; otherwise, all processors return to the resident module at entry PRINTSEP.

Having completed all processor initialization, the characteristics JOE is referenced as a setup descriptor, the 3211 index value is forced to 1, and the device setup verification subroutine is called. After setup is finished, Print Processors produce a job header page, if requested, which either indicates a new work item (by START) or a continued work item (by CONT). Both header and trailer pages contain job name, job number, and SYSOUT class in block letter format if at least 30 lines of separator page print is requested.

The next section of the Print/Punch Processor has the task of examining each PDDB provided in the IOT(s) by the Execution Processor and selecting those which match the work and characteristics JOEs. For continuation of output which has been partially completed, the first PDDB examined is the one representing the interrupted data set. After checkpointing the current IOT track address, the PDDB offset in the IOT, the first track address of the data set, and the initial Record Control Block (RCB) offset in the first data buffer, the processor clears the warm start flag and calls device setup verification to prepare the device for output. As each data buffer belonging to the selected data set is read, it is checked for validity. A failure of this check

results in termination of that data set and selection of the next data set.

If selection of the PDDB causes a change in any of the non-count information of the type-6 SMF record, then a record is written describing the previous data sets by calling the SMF type-6 record-writing routine.

The main print/punch loop steps from one RCB to the next (throughout each buffer belonging to a data set) building a CCW for each output record. The number of CCWs in a chain is limited by either of the HASP generation parameters &NUMPRCCW or &NUMPUCCW or by the number of output records available in one buffer--whichever is smaller. Automatic page overflow is provided for data sets when the lines-per-page parameter on the JOB card is not zero. This does not, however, prevent printing over the page perforation since the line counter is reset whenever a skip to any channel is encountered in the data set. For checkpoint purposes, the PRC is updated at every skip issued to printer, thus allowing rapid restart on a warm start. If requested, each print line not directed to a 3211 or 3800 is translated to remove unprintable bit patterns. At the end of each data set, the current IOT is read to prepare for selection of the next data set by PDDB scan. If the data set just finished was the HASP Job Log, data collected from the JCT is printed as the HASP job statistics block. Data set copy count greater than one results in the PDDB selection of the same data set until the requested number of copies has been printed or punched. If OPTCD=J was specified, then the table reference character that is the first character of each print line is processed. For 3800 printers, this means conversion of the character to a Select Translate Table command whenever a change of character arrangement tables is indicated. For non-3800s, this means dropping the table reference character from the final output.

When all data sets belonging to the current work JOE have been processed, the JCT is read and validity checked in preparation for processor termination. If the termination is abnormal, messages are written both to the operator and to the output device (if a printer), giving the reason. Fields in the JCT which reflect print/punch time, line/card counts, etc., are updated, and the JCT is written back to SPOOL. If SMF data recording has been requested, an SMF type-6 record is built in an SMF buffer and scheduled for writing using $GETSMFB and $QUESMFB HASP macros. Punch Processors punch a blank card to clear the last valid data set card record. All processors, using data from the characteristics JOE as setup descriptor, call device setup verification in preparation for the trailer page on printers. Punch Processors also follow this path to ensure correct device setup for the next $#GET request.

The last requirement is that a macro indicating end of processing be issued to release the work JOE. Since some operator commands (such as cancel ($C), restart ($E), and interrupt ($I)) release the work JOE early, this section is skipped in those cases. If a repeat ($N) has been issued but could not be immediately honored, the macro $#PUT is used to place the work JOE back into the JOT for the requested copy. Otherwise, $#REM is issued to dispose of the completed work JOE. Control blocks and buffers required for processing are released and the Print/Punch Processor returns to its primary entry point to select a new job.

DEVICE SETUP VERIFICATION SUBROUTINE

This subroutine checks the device type. If the device is a 3800, then the subroutine goes to the 3800 device setup and verification subroutine.

The task of this subroutine is to ensure that the device owned by the Print/Punch Processor is set up with the requested forms and, when appropriate, the Forms Control Buffer (FCB) and Universal Character Set Buffer (UCSB). The FCB must also include the proper index byte if the device is a 3211. A parameter list is supplied containing the required setup, which is compared with the DCT that specifies actual physical setup. If a mismatch requires operator intervention, a console message is issued and processing is halted until the operator indicates otherwise. The most obvious response is for the operator to perform the requested setup and then use the start ($S) command to continue processing. Optionally, the operator can override any of the setup requirements before issuing the start ($S) or can suspend processing completely by using cancel ($C), interrupt ($I), or restart ($E). Next, the UCSB is loaded (if supported by the device), and any of the following is true:

1. This is the first use of the printer since HASP was started.
2. A change in UCSB identification has occurred.
3. The operator has issued a set ($T) command to this device with the operand 'T=' or 'U=' since the last UCSB load was done.
4. The last attempt to load the UCSB failed.

If the requested UCSB image is not in the library, a message is issued, a flag is set requesting that a subsequent load is needed, and the previous UCSB ID is moved into the DCT.

After UCSB loading, an attempt is made to load the FCB and index value for 3211 printers if any of the following is true:

1. This is the first use of the 3211 Printer since HASP was started.
2. A change in FCB identification or index value has occurred.
3. The operator has issued a set ($T) command to the device with the operand 'C=' since the last FCB load was done.
4. The last attempt to load the FCB failed.
5. The last load of the FCB was done with an index value different than that contained in the system copy of the FCB image.

If the requested FCB image is not in the library, a message is issued (followed by a setup message) and the device is stopped to allow specification of a valid FCB ID. Having completed the above task, the device setup verification subroutine returns control to the calling point in the Print/Punch Processor.

NOTE: When the device setup verification area of the HASP print-punch processor loads the UCSB on a 1403 or a 3211 printer, it is loaded with folding.

3800 DEVICE SETUP AND VERIFICATION SUBROUTINE

This subroutine verifies that the forms, flash, burst, and FCB settings of the device (in the DCT and DCTE) match those required by the caller. If there is a mismatch, then a request is made to the operator to set up the device with the proper forms, flash, burst, and FCB. The most obvious response is for the operator to issue start, indicating continue processing. Optionally, the operator can override any of the setup paramenters using the set ($T) command before issuing the start command. The operator can suspend processing completely by issuing cancel ($C), restart ($E), or interrupt ($I).

Next the subroutine must verify that the device is set up properly according to FCB, character arrangement tables, copy modifications, and copy numbers. If the requested values match the current setup, then no other processing is performed.

If the internal setup does not match the request, HASP attempts to satisfy the request without issuing SETPRT. Issuing the SETPRT SVC involves a task switch and other relatively high overhead processes. HASP can perform any of the following tasks without issuing SETPRT:

- Loading of a HASPGEN-defined FCB
- Changing to the hardware default character arrangement table (when this change does not cause a change in some other factor requiring SETPRT).
- Changing to having no copy modification in effect.
- Changing the current copy numbers, flash counts, and starting copy numbers.

HASP will issue SETPRT to accomplish any of the following:

- Load an FCB defined in SYS1.IMAGELIB.
- Load any character arrangement table other than the hardware default.
- Load a copy modification module.

If a SETPRT is required, then the subroutine transfers control to the STPT subtask to issue the SETPRT.

If and error occurs in SETPRT processing, the operator is informed and has the same options as described for the setup message, though in this case the normal action is to override or suspend processing for the JOE.

HASP PURGE PROCESSOR

The Purge Processor frees the job's acquired HASP direct-access space, creates a type-26 SMF record, and removes the Job Queue Element from the system. The processor acquires a Job Queue Element and issues the $ACTIVE macro to inform the HASP Dispatcher that the processor is active. Then, a direct-access Device Control Table (DCT) and two HASP buffers are acquired and initialized so that the job's Job Control Table (JCT) and Input/Output Table(s) (IOT(s)) may be read into the buffer from the SPOOL disk. If a DCT or buffer is not available, this processor will be placed in a HASP $WAIT state until a DCT or buffer can be acquired. If no permanent I/O errors occur while reading the JCT, a $PURGE macro instruction is issued to return the job's direct access input tracks. If no permanent I/O errors occur while reading the IOT, a $PURGE macro instruction is issued to return the job's direct access output tracks. Any additional IOTs for the job will then be read into the IOT buffer, and their direct-access output tracks will be freed unless a permanent I/O error occurs. If a permanent I/O error occurs while the JCT or an IOT is being read, the Disastrous Error routine is called and the $PURGE macro instruction is not executed. Once all of the job's direct-access tracks have been freed, a buffer is obtained using the $GETSMFB macro. If user exits are to be taken, a second SMF buffer is obtained. The common exit parameter area, the JMR portion of the JCT, is preserved in one buffer and a type-26 SMF record is created and saved in the second buffer. The second buffer is chained to the first buffer and the $QUESMFB macro is issued to put the first buffer on the busy queue of SMF buffers and to POST the HASPACCT subtask. If user exits are not to be taken (EXT=NO), only the type-26 record is queued up to be written out. Next, the job queue Element is removed from the HASP Job Queue and the following message is issued to the operator:

```
JOB xxxx IS PURGED
```

Finally, the buffers and DCT are freed, and the $DORMANT macro instruction is issued to indicate to the HASP Dispatcher that the processor is inactive and control is returned to the start of the routine for the processing of the next job to be purged.

HASP COMMAND PROCESSOR

The HASP Command Processor receives all HASP commands entered from acceptable local or remote HASP input sources. The processor is responsible for decoding each command and performing the processing necessary to cause appropriate action to the operator's request.

The HASP Command Processor is initially entered at the beginning of the control section (CSECT) HASPCOMM. Subsequent reentries are returns from the various command subprocessors with optional requests for the displaying of the "OK" message or other message contained in the COMMAND area of the PCE. After displaying any requested replies, the HASP Console Message Buffer (CMB) queue $COMMQUE is examined for the presence of the next command to process. If no CMB is queued, the Command Processor waits on work. When $POSTed or if a CMB is present upon entry, the Command Edit routine is entered via $LINK macro.

COMMAND EDIT ROUTINE - HASPCOME

Verb Conversion

The Command Edit routine optionally converts the command text from the long form to the standard single-character verb form. The data portion of the CMB, up to the first comma (,) or apostrophe ('), is made upper case and nonblank characters are shifted to the left. The resulting text is compared against arguments in the Verb Conversion Table contained within the routine. If a match is found, the corresponding standard form of the command is substituted.

Command Edit And Break Out

The information in the HASP CMB is moved to the COMMAND field in the PCE work area. The two bytes CMBFLAGS and CMBCONS of the CMB are moved to the COMFLAGS and COMROUTE fields of the PCE work area. These two bytes, when combined with the two succeeding bytes in the PCE, (COMLENGTH and COMCLASS) form the list form of the $WTO used for all responses to the operator from the Command Processor.

The COMMAND area of the PCE is primed with blanks, and the buffer is scanned. Characters are ORed (moved with upper casing) into the COMMAND area. Blanks encountered in the CMB will normally be skipped (blank elimination); however, if an apostrophe is encountered, blanks will not be skipped until the next apostrophe. Double apostrophe characters will cause the blank compression status to remain as previously set; however, the second apostrophe of the pair will be eliminated.

As each comma is encountered, an entry of the next available character position is made in the COMPNTER area of the PCE. (The first entry is the address of the character after the verb. The second is the address of the second operand, etc.) When the COMPNTER area is full, recording is discontinued. Upon completion of the scan, the CMB is released, the count in $COMMCT is incremented (POSTing CMB if necessary), the "L=cca" operand (if present and valid) is removed from the command, and the "cc"

value is placed in the COMROUTE field and the "a" value is converted and placed in the area bits of the COMCLASS field. The COMNULOP field in the PCE is set to the address of the second character beyond the last solid character (null operand), and the operand pointers are shifted down adjacent to the COMNULOP field. Operand control registers are set as follows:

WD = address of the first operand pointer in the COMPNTER field
WE = 4
WF = address of the last operand pointer in the COMPNTER field.

Selecting The Command Subprocessor

The Command Selection Table is used to determine the appropriate command subprocessor which must be entered. Starting with the first element, the Selection Table is scanned for a matching verb. When the verb is located, the first character of the first operand is then used for comparing. If a match is found on the operand or if the table entry contains an X'FF' for operand argument, the table entry for the command is considered "located". If the end of the entries is encountered for the verb or table, the command is considered invalid and the edit routine returns to the main processor with INVALID COMMAND message in the COMMAND area for display. (See macro for format of the Selection Table entry.)

Validating The Source And Entering The Subprocessor

Each entry of the Selection Table may have restriction indicators as follows:

COMRMT = 1 - Reject remote sources
COMS = 1 - Reject consoles which are restricted from entering system commands
COMD = 1 - Reject consoles which are restricted from entering device commands
COMJ = 1 - Reject consoles which are restricted from entering job commands

The selection indicators correspond with the restriction indicators which appear in the COMFLAGS field. The COMFLAGS indicator is previously set from the CMBFLAGS field of the HASP Console Message Buffer, which in turn is set by other HASP processors as follows:

1. CMBFLAGS when set by the remote console processor or remote reader processors will contain the remote indicator. This indicator corresponds to COMRMT bit in the Selection Table.

2. CMBFLAGS when set by the OS console interface is the OS authority indicators inverted with the Exclusive or Immediate (XI) instruction.

The restriction indicators are used as the second operand of a Test Under Mask (TM) instruction. If any restriction indicator in the COMFLAGS field corresponds to any restriction indicator in the Selection Table entry, the command is rejected as invalid. If the operand "L=CC" has not been accepted and the source console is an Operating System console, the Selection Table entry is examined for the presence of an automatic redirection index. If the offset exists, the COMROUTE field is adjusted with a new "cc" value and the area ID portion of the COMCLASS field is adjusted with a new "a" value (converted) if "L=a" or "L=cca" had not been accepted. A special check is made to ensure that any redirection of command response to an out-of-line area is not for a command that could cause console lockout (area z is forced if required). Register 1 is set with the value in the Selection Table entry COMTOFF field, and control is passed to the CSECT indicated by the selection title entry element via the $XCTL macro.

COMMAND SUBPROCESSOR CONTROL SECTIONS

The entry routine of each command subprocessor control section will, if applicable, use the offset value in register 1 (set by the edit routine) to determine the relative entry point for the designated subprocessor. Normally the subprocessor is entered directly by the special Command Processor macro: Branch Relative Register on R1 ($BRR R1). However, some control section entry routines will preprocess the operands of the command prior to entering the subprocessor. Each subprocessor performs the desired functions and returns to the main Command Processor for the next command.

HASP COMMAND PROCESSOR ORGANIZATION

The HASP Command Processor is created by a single assembly with multiple control sections (CSECTs). The main CSECT HASPCOMM is the only portion of the Command Processor that is always part of the HASP load module. It contains all V-type address constants required by the subcommand processors and all "BASE2" service routines. The Command Edit routine HASPCOME receives control from the main processor and determines which command subprocessor CSECT to enter for processing of the command entered. One or more of the various command subprocessor CSECTs are used in processing each HASP operator command. Although the physical CSECTs are organized in accordance with the size of the overlay work area, the logical organizational groupings are as follows:

1. Job Queue Commands
2. Job List Commands
3. Miscellaneous Job Commands
4. Device List Commands
5. System Commands
6. Miscellaneous Display Commands
7. Remote Job Entry Commands.

HASP COMMAND PROCESSOR WORK AREA

The HASP Command Processor PCE work area is the primary work area for the processor and is the only area which may be used to save information When a $WAIT is issued by the processor or by any of the "BASE1" service routines on behalf of the processor. The fields are generally used as described in the following paragraphs.

These fields are set by the Command Edit routine and are used to locate the beginning of each of the specified operands in the command currently being processed. COMNULOP contains a pointer to the second character beyond the last operand specified, i.e., points to a nonexistent or null operand. Operand 1 through n pointers are right-adjusted in COMPNTER so that the operand n pointer is adjacent to the null pointer. Command subprocessors use these areas for additional work space after the operand pointers are no longer needed. Examples of other uses are listed as follows:

1. Job queue commands $DN and $DQ place queue scanning control elements in the COMPNTER area.

2. Job list commands place the job range number (j-jj) in the corresponding operand pointer element area.

COMFLAGS to COMCLASS

This field contains a list form of the $WTO macro. The $WTO is referred to by a single-execute form of the $WTO (located within the HASPCOMM CSECT of the Command Processor) which is used for all operator messages generated by any routine within the processor. The CMBFLAGS and CMBCONS fields of the HASP Console Message Buffer for each command are used to construct the list form of the $WTO and provide correct route codes for replies. The three low-order bits of COMFLAGS are restriction indicators and are set to zero prior to each $WTO reply.

COMEWORK

This field is used as a work area and is used by function routines identified by the macro instructions as follows:

Macro	Contents Upon Exit From Routine
$CFCVE	Last character is blank
$CFDCTL	First four characters of requested device name
$DFJDCT	Address of HASP Job Queue Element for requested job
$CFJMSG	Same as $CFCVE

This field is aligned on a double-word boundary and is used as a work area and by function routines identified by the macro instructions as follows:

Macro	Contents Upon Exit From Routine
$CFCVE	Five-character number in EBCDIC with leading blanks
$CFDCTL	Last four characters of requested device name
$CFJMSG	Same as $CFCVE

COMMAND

This field contains the compressed form of the operator command with trailing blanks at the time each command subprocessor is entered. The command is overlayed by the reply message text for all $WTO messages issued by any Command Processor routine. Some command subprocessors use the area as a scratch area, and in some cases subprocessors use the right end for storage of critical information while message replies are generated in the left end of the area.

CODING CONVENTIONS

The symbols within the Command Processor conform to the following conventions:

1. All main processor, Edit routine, and PCE work area symbols start with the characters "CO"

2. All function macro-generated symbols start with "COF".

3. All command subprocessors have entry point symbols of the following form:

Form	Example	Command	Comments
Cvo	CDN	$DN	v = the verb of the command o = the first operand character
Cv	CB	$B device	Single-character identifier
Cvxx	CD7D	$D'jobname'	Apostrophe is hexadecimal 7D

4. All symbols created for the support of the command will start with characters which identify the entry point (CDNxxxx identifies a location which was originally written for the $DN command). Commands with no unique operand character symbol have the character "X" as the third character. (CBX.... identifies a location which was originally written for the $B device command.) These conventions may be altered in cases where the command identification characters are redefined after original development.

5. The main processor CSECT is HASPCOMM, all other CSECTs are defined via the symbol field of the $COMGRUP macro, specified starting with the characters "HASPC".

REGISTER CONVENTIONS

The Command Edit routine passes control to the control section (CSECT) which contains the appropriate command subprocessor. When the command group entry routine receives control, the registers will contain the following:

Reg	Contents
R0	Unpredictable
R1	Entry offset from the command entry offset
WA	Unpredictable
WB	Unpredictable
WC	Unpredictable
WD	First operand pointer (zero if no operand)
WE	4
WF	Last operand pointer
BASE3	Base for CSECT
BASE1	HCTDSECT address
BASE2	Beginning of main Command Processor
SAVE	PCE address
LINK	Unpredictable
R15	Unpredictable

If more than one command subprocessor appears within the group, register R1 will be set by the $COMGRUP entry routine so that a $BRR R1 will enter the command subprocessor.

HASP COMMAND PROCESSOR MACROS

To facilitate flexibility in the development and possible modification of the Command Processor, a macro package is included within the assembly source deck. This section is intended to supplement the HASP Command Processor source listings obtained from the HASP generation and assembly process in assisting the user to understand the generated code as specifically used in the current HASP.

Each HASP Command Processor macro may be dependent on the definitions contained within the Command Processor source as well as other members of the HASP source library. These macros are catagorized as follows:

ORGANIZATIONAL - Macros which provide basic definitions and are closely associated with the organization of the processor.

BASE2 SERVICES - Macros which call upon the main Command Processor to perform a service (display a reply).

CONDITIONAL IN-LINE FUNCTIONS - Macros which perform the function in-line or link to a routine which performs the desired function.

RELOCATABILITY AIDS - Macros which assist in keeping the overlay CSECT relocatable around $WAIT or implied $WAIT situations.

The following conventions are used in specifying parameter requirements:

"parameter=** -" - keyword parameter is required

"parameter=text -" - the assumed value if the keyword parameter is not specified

"parameter -" - The parameter is an optional positional parameter

"parameter - Required" - the parameter is a required positional parameter.

COMMAND PROCESSOR MACRO SUMMARY

Opcode	Definition
ORGANIZATIONAL:	
$COMWORK	Command Processor Work Area (symbolic definitions)
$COMGRUP	Define Group Of Command Subprocessors
$COMTAB	Define Command Table Element
BASE2 SERVICES:	
$CRET	Return To Main Command Processor
$CWTO	Write To Operator
CONDITIONAL IN-LINE FUNCTIONS:	
$CFCVB	Convert To Binary
$CFCVE	Convert To EBCDIC
$CFDCTD	Device Control Table Display
$CFDCTL	Device Control Table Locate
$CFINVC	Reply Invalid Command
$CFINVO	Reply Invalid Operand
$CFJDCT	Find Job's Device Control Table
$CFJDCTC	Continue Find Job's DCT
$CFJMSG	Display Job Information Message (Conditional)
$CFJSCAN	Scan Job Queue Assistance
$CFSEL	Select A Routine Based On Character
$CFVQE	Verify Console Control Over Job
RELOCATABILITY AIDS:	
$ARR	Add Relative Register
$BRR	Branch Relative Register
$SRR	Subtract Relative Register

ORGANIZATIONAL MACROS

$COMWORK - Command Processor Work Area (symbolic definitions) - This macro adds to the PCEDSECT definitions for fields located in the Command Processor PCE work area. Additional symbolic constants for BASE2 services and some externally defined parameters are defined.

$COMGRUP - Define Group Of Command Subprocessors - This macro defines the Command Processor overlay control section via the $OVERLAY macro. It provides an optional entry point routine which locates the command subprocessor for the commands which belong to the group and sets register R1 to the relative address. (The symbol field must be specified for this macro.)

n positionals - Each positional specifies the command identification characters for the corresponding command subprocessor located within the group. Example:

Specification	Command	Subprocessor Entry Point Name
AA	$AA	CAA
DA	$DA	CDA
B	$B device	CB
C	$C device	CC
P40	$P	CP40
S40	$S	CS40
D7D	$D'jobname'	CD7D

PRTY=** - Priority of the HASP overlay defined by the macro.

DELAY=NO - The subprocessor will be entered via $BRR R1 macro instruction. If "YES" is specified, R1 will contain the appropriate relative entry point address and control will be given to the statement following the macro statement. (More than one positional must be specified if R1 is to be set or the branch is to be executed.)

$COMTAB - Define Command Table Element - This macro defines an element in the Command Selection Table which is used by the Command Edit routine for identifying legal commands, eliminating unauthorized input sources, and entering the correct command group CSECT.

verb - Required - Command identification character(s) corresponding to the $COMGRUP positional parameter specification for the command. No two $COMTAB macro statements may specify the same identification character string. All macro statements creating entries for the same command verb will appear in consecutive statements with the statement which specifies a single identification character last.

group - Required - Exact characters used in the specification in the symbol field of the appropriate $COMGRUP macro statement.

REJECT= - Command source rejection mask. One or more of the following symbols may be specified as follows:

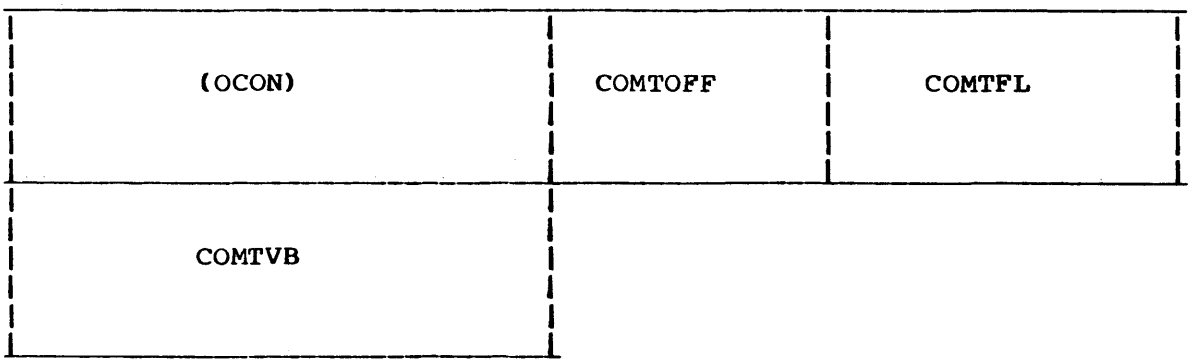
"COMRMT" - reject command if entered from a remote
"COMS" - reject command if entered from a console not authorized for system control
"COMD" - reject command if entered from a console not authorized for device control
"COMJ" - reject command if entered from a console not authorized for job control

Rejection of either a remote or a console not authorized for system appears as follows:

"REJECT=COMRMT+COMS"

REDIR=0 - Response to commands entered via Operating System Console are not to be automatically redirected. If the entry console is an Operating System console and values 1-15 are specified, the appropriate entry in the apparent source console redirect response table is used to redirect responses.

SELECTION TABLE ELEMENT



Symbol	Offset	Length	Description
(OCON)	0	2	Overlay constant used to identify the command subprocessor control section.
COMTOFF	2	1	Offset in the command subprocessor control section of "where to go" constant.
COMTFL	3	1	Redirection index and restriction flags.
			0000 xxxx - No automatic redirection of response.
			nnnn xxxx - n=1-15 index in the Redirect Response Table for the console

of "apparent entry".

xxxx 1... - COMRMT - Reject command if from remote work station.

xxxx .1.. - COMJ - Reject command if source not authorized for job control.

xxxx ..1. - COMD - Reject command if source not authorized for device control.

xxxx ...1 - COMS - Reject command if source not authorized for system control.

COMTVB 4 2 Command verb plus additional qualifier:

1. First character of first operand.
2. X'FF' indicator signifies that if previous entries for the verb resulted in a "no match," use this one.

Note: Although the Edit routine allows entry to the command subprocessor each command subprocessor may reject the command due to restricted operands.

BASE2 SERVICES

$CRET - Return To Main Command Processor

MSG= - Address of the message to be moved to COMMAND area for display. (L=operand if a non-register form is required.) "MSG=OK" indicates that the main processor is to display the OK message.

L= - Value representing the length of the message that is to be moved or has already been moved.

$CWTO - Write To Operator

Registers Used - R0, R1, WA, LINK, R15

MSG= - Address of the message to be moved to COMMAND area and displayed. (If this operand is specified, a non-register form of the ... L=operand must be specified.)

L=** - Value representing the length of the message that is to be moved or has already been moved.

CONDITIONAL IN-LINE FUNCTIONS

The HASP Command Processor (as distributed) provides the ability for the author of the command subprocessor to specify whether or not the code which performs the function is in-line or out-of-line. If an out-of-line routine is used, the name and location of the subroutine must be defined. This is accomplished with parameters standard for all function macro instructions (with the exception of $CFJSCAN) as follows:

1. TYPE=CALL - The macro statement is not a definition form of the macro. For "TYPE=DEF", the macro statement defines the subroutine form of the function and return linkage must be provided.

2. SYMBOL=address - The address of the "TYPE=DEF" version of the macro instruction. This indicates that only linkage to the "TYPE=DEF" version is to be provided. If neither "TYPE=DEF" or "SYMBOL=" parameters are specified, the code will be generated in-line with no return linkage.

$CFCVB - Convert To Binary - This macro converts the numeric portion of a command operand to one or two numeric values.

Registers Used - R0, R1, LINK, R15

R0 - contains the last number converted

R1 - contains the next to last number converted (last number if the only one or if the last is smaller than the previous).

POINTER=(R1) - Address of the COMPNTER field which addresses the operand containing one or more numerical values separated by a dash (-).

NUM=2 - Returns two values.

NUM=1 - One value is sufficient (R1 will be unpredictable on return).

NOK=** - Address of the error exit routine if the operand does not contain a number or if the number is too large.

$CFCVE - Convert To EBCDIC - This macro converts the number in register R0 to printable EBCDIC and sets the five resulting digits in the first five characters of the PCE area COMDWORK.

Registers Used - R0, LINK

VALUE=(R0) - The positive binary half-word value to convert to EBCDIC. If the register form is not used, the value is contained within the addressed half word.

$CFDCTD - Device Control Table Display - This macro displays the device name, unit address, and status of the DCT requested.

Registers Used - R0, R1, WA, LINK, R15

DCT=(R1) - Address of the DCT to be displayed.

$CFDCTL - Device Control Table Locate - This macro converts the abbreviated form of the device name to the long form (if abbreviated form is specified) and searches the DCT chain for a matching device.

Registers Used - R0, R1, R15, LINK

R1 - contains the address of the DCT found or zero if no DCT found.

POINTER =(R1) - Address of the COMPNTER field which addresses the operand containing the device name (abbreviated).

$CFINVC - Reply Invalid Command - This macro returns to the main Command Processor and causes the display of "INVALID COMMAND".

$CFINVO - Reply Invalid Operand - This macro moves eight characters, starting with the first character of the "current" operand to the Command area and returns to the main Command Processor, causing the display of "operand INVALID OPERAND".

OPERAND=(R1) - The address of the operand to display.

$DFJDCT - Find Job's Device Control Table - This macro searches the DCT chain for an active printer, punch, or reader DCT which is assigned to a processor whose PCE contains a pointer to the HASP Job Queue Entry belonging to the desired job. If the device is not found, exit will be to the instruction immediately following the $CFJDCT statement (in-line code version); otherwise, exit will be to that location plus four.

Registers Used - R1, LINK, R15

JOBQE=(R1) - Address of the HASP Job Queue Entry for the desired job.

CONT= - Name to be used by a $DFJDCTC macro to continue searching for additional devices.

$DFJDCTC - Continue Find Job's Device Control Table - This macro enters the named $DFJDCT macro expanded routine for the continuation of the scan of DCTs operating on a named

job. Registers and work areas must be the same as they were previously when entering the $DFJDCT routine. Exits and registers are defined under $DFJDCT.

SYMBOL=** - Address of the continue entry in the $DFJDCT macro-expanded routine (defined using the CONT= parameter).

$CFJMSG - Display Job Information Message - This macro sets into the COMMAND area of the PCE the information required for the Job Information Message and displays the message.

Registers Used - R0, R1, WA, LINK, R15

JOBQE=(R1) - Address of the HASP Job Queue Entry for the desired job.

JDCT = Address of the $CFJDCT TYPE=DEF macro, which may be used to locate the job's DCT. Register form is prohibited.

CVE= - Address of the $CFCVE TYPE=DEF macro, which may be used to convert numeric information to EBCDIC. Register form is prohibited.

JOB= - May be ignored by the macro; however, if specified as "JOB=SET", the text "JOBj" is assumed (by the expanded routine) to have been set in the COMMAND area for the desired job.

OPT= - If specified, causes the message to be displayed only if the job is active (OPT=A) or queued (OPT=Q), and exits to the next instruction. If job not displayed, exit will be to the next instruction plus four bytes.

$CFJSCAN - Scan Job Queue Assistance - This macro is used to assist in scanning the job queue. As each entry is located the user's Process routine is entered. The user examines the entry, performs the function desired on the entry, and returns to the symbol specified by the "NEXT=" operand. When the end of the queue is encountered, control is given to the instruction following the macro instruction. An optional feature of the macro is to allow the Process routine an "IGNORE" entry to the generated code to indicate that the current job entry is not acceptable to the Process routine. If the "IGNORE=" option is specified, the corresponding "EMPTY=" option is required. Register 1 is the scan register and is assumed to be unaltered by the user's Process routine. The "TYPE=DEF" option is not permitted for this macro.

Registers Used - R1, BASE2

R1 - scan register

BASE2 - found/not found switch (in addition to processor base).

PROCESS=** - Address of the user's Job Queue Element Processing routine. Register form is prohibited.

IGNORE= - Symbol used to define the entry where the scan will continue when the current job entry is not of the desired type.

NEXT=** - Symbol to be used to define the entry where scan will continue when the current job entry is of the desired type.

EMPTY= - Name of the user exit routine to be entered when the job queue is found empty of jobs of the desired type. Register form is prohibited.

$CFSEL - Select A Routine Based On Character - This macro matches the designated input character against a list of arguments and transfers control to the routine designated by the corresponding address. If no match is found, the next sequential instruction is entered.

Registers Used - R1, LINK, R15

n positionals of form: (character, address) - Each positional "character" subparameter specifies an argument. The corresponding address subparameter indicates the address of the routine to be entered if the character matches the argument. Register form is prohibited.

OPERAND=(R1) - Address of the designated input character to be examined.

$CFVQE - Verify Console Control Over Job - This macro tests the COMFLAGS field of the PCE to determine if the input source is a remote. If the source is a remote, the "Not OK" routine will be entered unless either the print or punch route codes for the indicated job specify the remote. Otherwise, the OK routine will be entered.

Registers Used - R1, LINK

JOBQE=(R1) - Address of the HASP Job Queue Entry for the desired job.

OK= - Address of the routine to be entered if the console has control over the job. The address may be the symbolic register containing the address if specified as "OK=(register,BCR)" or "OK=(relative register,$BRR).

NOK= - Address of the routine to be entered if the console does not have control over the job. The address may be the symbolic register containing the address if specified as "NOK=(register,BCR)" or "NOK=(relative register,$BRR). Either "OK=" or "NOK=" parameters must be specified.

RELOCATABILITY AIDS

$ARR - Add Relative Register - This macro instruction is used in conjunction with $SRR to restore the specified register to refer to the true address of relocated information.

Register - Required - Symbolic register containing the address to made true.

$BRR - Branch Relative Register - This macro instruction is used in conjunction with $COMGRUP to enter a subprocessor routine using the offset provided by the $COMGRUP routine.

Condition - Necessary condition for branch. If this parameter is omitted, no comma should be written to signify its omission. "Condition code" may be specified by the character strings: E, NE, H, L, NH, NL, Z, NZ, P, M, NP, NM, O or NO.

Register - Required - Symbolic register containing the offset.

$SRR - Subtract Relative Register - This macro instruction is used to make an address pointer relative for possible relocation before next referral to the information contained at the address.

Register - Required - Symbolic register containing the address to be made relative.

HASP CHECKPOINT PROCESSOR

This processor writes on disk the information necessary to effect a subsequent restart of the system. It writes the information at a predefined time increment and at the completion of each stage of each job.

The first entry into the Checkpoint Processor is to a section that initializes the processor. This section issues a $GETUNIT macro instruction to obtain a DCT for a disk and completes this DCT by inserting the Event Wait Field address, track to be written, and the buffer address.

Some miscellaneous checkpoint variables of the information to be checkpointed describe the following:

- Status of the system.
- Job queue which contains the status of each job in the system.
- Job output table (JOT) which allows warm start for jobs printed and punched output.
- Job information table (JIT) which contains additional job information.

The job queue and the JIT reside within the checkpoint area, but the miscellaneous variables and the JOT must be moved into this area.

The miscellaneous checkpoint variables are moved into the checkpoint area followed by the JOT. The processor then determines whether the JIT and/or the JOT need to be checkpointed and sets up the CCW chain accordingly. A $EXCP is issued to write the necessary records and a $WAIT on this I/O is initiated.

Next the checkpoint time interval is reset and the previous I/O is checked for errors. If no errors exist, the processor waits for job movement or for the time interval to expire. If errors do exist, a message is issued to the operator and the processor is permanently waited.

Since the completions of all HASP I/O operations are signaled asynchronously with HASP operation via IOS channel-end appendages, these completions must be queued by the appendage until all HASP processors can be synchronized to receive the notification. The purpose of the Asynchronous Input/Output Processor ($ASYNC) is to, at noninterrupt time, notify all processors of their I/O completions which were indicated by the OS I/O Supervisor at interrupt time.

The buffers (and respective IOBs) associated with I/O channel ends are chained, by the HASP channel-end appendages, for later processing by $ASYNC. In addition to the post of the HASP task by IOS on any I/O completion, the channel-end appendages also $POST the Asynchronous Input/Output Processor to initiate its processing when the HASP task receives control.

When $ASYNC receives control, it dequeues the first buffer from its chain of work (operating disabled, for this operation only, since its chain is updated at interrupt time). The master I/O count in the HASP Control Table (HCT) is reduced by one and catastrophic error code E01 is indicated if it becomes negative. The I/O completion code is moved from the OS IOB to a buffer control field. The Device Control Table (DCT) entry associated with this buffer is located and the active I/O count for the device is reduced by one. Catastrophic error code A01 is indicated if this count becomes negative.

Next the user's EWF address is extracted from the buffer and interrogated, and action is taken according to the following algorithm:

EWF = 0 — User does not want notification of completion of I/O operation (always a write). The buffer will be returned to the HASP buffer pool by $ASYNC.

EWF > 0 — $POST the I/O bit in the EWF specified and take no further action.

EWF < 0 — Enter a user-provided routine at the address specified by the absolute value of the EWF field. Addressability for the processor routine is established and the address given is entered via the Branch and Link instruction with the buffer address in register R1. No further action is taken upon return.

After performing the indicated action, $ASYNC returns to dequeue the next buffer from its chain and the above procedure is repeated. When the end of the chain is reached, $ASYNC enters the $WAIT state until additional I/O completions occur.

$ASYNC also performs, under the HASP task, certain functions for locally attached card readers, that were requested at interrupt time by the HASP attention exit routine (see description under HASP Initialization SVC). Such a request is recognized if the $ASYNC work chain pointer is negative. When this occurs, all local reader DCTs are scanned. If the UCB pointed to by a DCT has been flagged for attention by the attention exit, the DCT's HOLD bit is cleared and the Dispatcher's Event

Completion Field is $POSTed to indicate to other processors that a unit is available.

HASP TIMER PROCESSOR

The function of this processor is to reset the OS interval timer after a timer interrupt has occurred. This processor calls the IPOSTIT and ISETINT subroutines in the $STIMER/$TTIMER Interval Timer Supervisor, which causes the expired TQEs to be posted and the time interval specified in the first TQE in the TQE chain to be set into the OS interval timer. The processor then waits for another timer interrupt to occur. When the next timer interrupt is processed, the asynchronous exit routine posts this processor and the above procedure is repeated.

HASP MULTI-LEAVING LINE MANAGER

The function of this processor is to control all line activity with remote terminals. This includes line initiation/termination, remote terminal synchronization, line error recovery, and sign-on/sign-off processing. This processor interfaces very closely with the Remote Terminal Access Method.

When this processor receives control from the Dispatcher, it first determines whether an I/O operation has completed. If not, it scans each line (via the line Device Control Tables) to check for requested processing. When all processing has been completed, the processor returns control to the dispatcher ($WAITs) until more work becomes available.

When a channel end is detected, the channel end routine determines the sequence type of the Channel Command Word chain and branches to the appropriate section to analyze the channel end and initiate any error recovery procedures required.

The line Device Control Tables (DCT) are scanned and when one is found to be available, the Line Initiation routine is entered to acquire the DCT, to acquire a TP buffer, to construct an initial CCW chain, and to initiate I/O activity.

A single timer queue element is maintained by the Line Manager to initiate delays in line processing. This facility provides the capability of delaying a null response to a remote terminal and decreases the associated degradation. Various other Timer Queue Elements are maintained by individual line processors to initiate other delays of varying intervals.

The code in this processor is assembled conditionally, so only the instructions required to process a given configuration will be generated.

HASP TSO SUPPORT PROCESSOR

The Time Sharing Option (TSO) of the STATUS and CANCEL Command Processors use the TSO service SVC to make requests to HASP. When entered, the Exit and Support Processor performs the following functions:

1. The processor searches the HASP job queue for the named job setting and appropriate error response information for TSO in the event the job is not found or multiple jobs with the given name are found. If no errors were encounted, processing continues; otherwise, a return to TSO is taken.

2. If the request is to CANCEL the job, the processor ensures that the job is in a preexecution state. If not in a preexecution state, indicates that the job is not found and returns to TSO. Otherwise, the $P job command for the job is simulated.

3. If the request is for STATUS of the job, the processor determines which general queue type the job is in. If the job is in a queue prior to execution, the preexecution indicator is set; if the job is in execution, the execution indicator is set; or if none of the above, the post-execution indicator is set. The HOLD indicator is set if the job is in an operator releasable state. Because of synchronization problems with HASP processors using the HASP job queue, all of the functions required by the HASP TSO STATUS/CANCEL Exit are performed under the HASP task by the HASP TSO Support Processor (entry HASPTSOS). The exit, however, runs under the TSO user task (entry $TSOCOMM). The coordination interface procedures followed by the HASP task and user task routines follow:

   a. The user task routine copies the request information into a work area and activates the HASP task routine by $POST and OS POST.

   b. The HASP task routine performs the functions, fills in the feed back information, and OS POSTs the user task.

   c. The user task routine moves the feed back information into the callers parameter list and returns.

HASP PRIORITY AGING PROCESSOR

The function of the Priority Aging Processor is to regularly increase the priority of a job in such a way that its position in the HASP Job Queue is enhanced with the passage of time. This is accomplished by regularly passing through the HASP Job Queue and incrementing the priority field of all Job Queue Elements whose priority falls between upper and lower limits. These limits, as well as the time interval, are HASPGEN parameters and can be specified to fit the needs of an installation.

When this processor is dispatched it searches through the HASP Job Queue until it encounters a Job Queue Element whose priority field "QUEPRIO" is less than the HASPGEN parameter &PRIHIGH. For that Job Queue Element and every Job Queue Element after that (until the HASPGEN parameter &PRILOW is reached), the priority field is incremented by one. The Interval Timer is then reset, and the processor enters a HASP $WAIT until the timer interval expires.

Since the priority of the Job Queue Element is represented by the four high-order bits of QUEPRIO, adding one to this field has no immediate effect on the priority. After repeating this operation 16 times, however, the actual value of the priority will be increased by one. The value of the time interval is actually 1/16th of the interval implied by the HASPGEN parameter &PRIRATE. This effect tends to smooth out the process of Priority Aging by creating less impact when an interval expires.

To minimize CPU utilization, this processor discontinues operation whenever the HASP Job Queue is empty and does not continue until a new job enters the system.

HASP INPUT/OUTPUT SERVICES

The HASP Input/Output Supervisor ($EXCP) is used to interface all HASP Input/Output requests with the Operating System Input/Output Supervisor. Through the use of $EXCP, the HASP processors can achieve a certain degree of device independence for direct-access devices through the use of the track and sector conversion functions contained within the $EXCP routine. In addition, $EXCP also provides all I/O appendages required by the OS Input/Output Supervisor (IOS) and provides for the posting of I/O completions to each processor.

The interface between the HASP Input/Output Supervisor and the using processors is the Device Control Table (DCT), which is passed via the $EXCP macro instruction when I/O is requested. Upon entry to $EXCP, the address of the buffer to be used is obtained from the DCT, and the IOB (appended to the front of every buffer) is initialized. The user's Event Wait Field (EWF) address is moved from the DCT to the buffer and a pointer to the DCT is placed in the buffer. If the DCT is a direct-access type, the coded track address from the DCT is used to compute MBBCCHHR.

IF the HASPGEN parameter &RPS is set to YES and if the referenced device supports Rotational Position Sensing, the appropriate sector number is computed and stored in the IOB and the first CCW is set to a Set Sector command.

The IOB is now scheduled for I/O through the use of the standard OS Execute Channel Program macro instruction (EXCP), and immediate return is made to the caller. Each I/O request issued by HASP has an I/O appendage list specified which causes the appendages in $EXCP to be entered at various stages of I/O processing.

A page fix appendage, entered at EXCP time, fixes one page beginning with the first byte of the IOB. This page fix will not only fix the entire IOB but will also fix the data area and any CCWs within the IOB and/or data area. The appendage returns to IOS with an indication that no additional pages should be fixed since the DCB, DEB, ECB, appendages, and any CCWs which have been constructed in the PCEs will already be fixed.

Abnormal and normal channel end appendages are provided to signal the end of the I/O operation. Since these appendages are entered asynchronously with HASP operation, the buffer associated with the completed I/O is scheduled for synchronous HASP processing by either the Asynchronous Input/Output Processor or the MULTI-LEAVING Line Manager Processor (RJE operations only). The HASP task is posted, and immediate return is made to IOS.

If the cause of entry to the abnormal channel end appendage is a paper jam on a 3800 printer, the approximate number of pages lost is obtained from the sense bytes and placed in the DCTE for use by the Print/Punch Processor.

HASP BUFFER SERVICES

The Buffer Management routines are responsible for the allocation of the dynamic storage area (buffer pool) of HASP. Fixed-size buffers in this area are allocated and deallocated to HASP processors and routines via the $GETBUF and $FREEBUF macro instructions.

The $GETBUF routine consists of two programs which allocate HASP buffers or RJE buffers, respectively. Both programs function identically as follows: The appropriate free buffer pointer is tested and if no buffers are available, control is returned to the caller with the condition code set to zero. If a free buffer is present, the free buffer pointer is updated to point to the next free buffer; or, if this is the last available buffer, the pointer is zeroed. Then, if the debug indicator is on, a buffer validity checking routine is entered to ensure that the buffer is within the buffer pool. If it is not in the pool, the catastrophic error routine is entered; otherwise, control is returned to the $GETBUF routine. The condition code is set nonzero and control is returned to the caller with the buffer address in register R1.

The $FREEBUF routine enters the buffer validity checking routine if the debug indicator is on, if the buffer to be freed is inserted back into the appropriate free buffer pool (depending on whether the buffer is a HASP buffer or an RJE buffer), and if the IOBSTART field is updated with the address of the buffer's channel program, IOBCCW1. The HASP Dispatcher's Event Control Field is posted to show that a buffer is available, and control is returned to the caller.

HASP SMF SERVICES

The HASP SMF service routines are responsible for obtaining HASP SMF buffers from a free queue and for placing allocated HASP SMF buffers on a busy queue and POSTing the HASPACCT subtask.

The SMF Buffering Routine, $GETSMFB, is used to obtain a HASP SMF buffer from the $SMFFREE cell in the HCT. If no buffers are available, register R1 is checked for zero. If zero, the routine returns to the caller. If nonzero, the routine $WAITs for SMF and then loops back to try to obtain a buffer again. Once a buffer is obtained, its address is placed in R1 and then control is returned to the caller.

The $QUESMFB routine places a HASP SMF buffer, whose address is in register R1, on the end of the queue of busy HASP SMF buffers. The busy queue is pointed to by the $SMFBUSY cell in the HCT. Then the HASPACCT subtask is POSTed for work and control is returned to the caller.

HASP JOB QUEUE SERVICES

Jobs being processed or awaiting processing by a HASP phase are represented in an ordered queue by a Job Queue Element (JQE). The Job Queue Management routines are used by the HASP processors to insert, alter, locate, and remove Job Queue Elements. The queue elements are maintained in priority at all times with the highest priority element at the top of the active chain. There are six Job Queue Element routines which are called by issuing the following macros: $QADD, $QREM, $QGET, $QPUT, $QLOC, and $QSIZ. The Job Queue Elements are arranged in two chains. The active chain contains the Job Queue Elements for all the jobs in the system at a given time. The free chain contains all the queue elements which are not in use.

The $QADD routine is called whenever a queue element is to be added to the active queue. If the Checkpoint Processor is waiting for the checkpointed information to be written on the primary SPOOL volume, this routine enters a HASP $WAIT state. Whenever the Checkpoint Processor's I/O is complete, the free queue chain is tested to see if any free queue elements are available. If none are available, control is returned to the caller with a condition code of zero. If a queue element is available, the correct slot within the active queue chain is located by comparing the priority of the element to be added with the priorities of the elements in the active chain. When the priority of the new element is higher than the priority of the element in the active chain, the free Job Queue Element is extracted from the free queue chain and is inserted into the active chain. All the information for the new Job Queue Element is moved from the location pointed to by register R1 into the new Job Queue Element. Then the HASP Dispatcher's Event Control Field is posted to indicate that a Job Queue Element is available. The Checkpoint Processor's PCE is also posted so that it will be given control to write the updated Job Queue onto the primary SPOOL volume. The condition code is set nonzero, and control is returned to the caller. Upon return, register R0 contains the address of the associated Job Information Table entry.

The $QREM routine is entered to remove a Job Queue Element from the active chain. It will enter the calling processor into a HASP $WAIT state if the Checkpoint Processor's I/O is not complete. When the Checkpoint Processor's I/O is complete, the Job Queue Element that is to be removed is located by comparing its job number with the job numbers of the queue elements in the active chain. If an equal comparison is not found, control is returned to the caller with the condition code set to zero. If a match is found, the Job Queue Element is removed from the active chain and added to the top of the free chain by updating all the chain pointers. The Checkpoint Processor's PCE is posted so that it will be given control to checkpoint the Job Queue. Then control is returned to the caller with the condition code set nonzero to indicate that the queue element was successfully removed.

The $QGET routine is entered to acquire a Job Queue Element in a specified queue so that the job may be processed. The active queue chain is searched for a Job Queue Entry of the specified type (e.g., execution, output, or purge) that is not in HOLD status and is not presently acquired. If such a job is not present, control is returned to the caller with the condition code set to zero. If an acceptable

queue element is found, the QENTBY bit is turned on in the queue element to show that the element has been acquired, and control is returned to the caller with the condition code set nonzero, register "R1" pointing to the job queue element that was acquired, and register "R0" pointing to the associated Job Information Table entry. Whenever the system is in a drained status, this routine will be crippled so that control will always be returned to the caller with the condition code set to zero (to indicate that no available Job Queue Elements are present).

The $QPUT routine is entered to return a previously acquired Job Queue Element to the active chain, but with a new queue type. It will enter the calling processor into a HASP $WAIT state if the Checkpoint Processor's I/O is not complete. When the Checkpoint Processor's I/O is complete, the job number of the queue element to be returned is compared with the job numbers of the queue elements in the active queue. If the job number is not found, control is returned to the caller with the condition code set to zero. If a match is found, the new queue type is set, the HASP Dispatcher's Event Control Field is posted to indicate that a Job Queue Element queue is available to be acquired, and the Checkpoint Processor's PCE is posted so that it will be given control to write the updated job queue onto the primary SPOOL volume. The Job Queue Element is placed in the queue indicated by register R0 upon entry to this routine. The QENTBY bit is set as indicated with the queue type in register R0. The condition code is set nonzero and control is returned to the caller. Upon return, register R1 contains the address of the Job Queue Element just returned, and register R0 contains the address of the associated Job Information Table entry.

The $QLOC routine is entered to obtain the Job Queue Element address when the job number is known. The job number is compared with the job numbers in the active chain. If a match is not found, control is returned to the caller with the condition code set to zero. If a match is found, the condition code is set nonzero, and control is returned to the caller with register R1 containing the located Job Queue Element's address and register R0 containing the associated Job Information Table entry address.

The $QSIZ routine is entered to obtain the number of Job Queue Elements in a given queue type and routing. The number of jobs of the specified type (excluding jobs in HOLD status) are counted, and control is returned to the caller with register R1 containing this count. If register R1 is nonzero, the condition code is set nonzero, and if it is zero, the condition code is set to zero. Whenever the system is in a DRAINED status, this routine is crippled so that control is always returned to the caller with register R1 zeroed, and the condition code is set to zero to indicate that no jobs are available in the specified job queue.

HASP UNIT SERVICES

The Unit Allocation routines are responsible for the allocation and deallocation of the input/output units which have been assigned to HASP. Device Control Tables (DCTs) are allocated and deallocated to HASP processors and routines via the $GETUNIT and $FREUNIT macro instructions.

The $GETUNIT routine scans the Device Control Table (DCT) chain in an attempt to find an available DCT of the requested type. If none are found, control is returned to the caller with the condition code set to zero. If an available DCT of the requested type is found, it is set "in use" and control is returned to the caller with the condition code set nonzero. The address of the DCT is returned in register R1.

The $FREUNIT routine first examines the Active Buffer Count field of the DCT to see if there are any buffers involved in active I/O with the associated unit. If the Active Buffer Count is nonzero, the processor is placed in a HASP $WAIT state until this count is reduced to zero. When the count is zero, the "in use" indication is reset and if the DCT is now available, the HASP Event Control Field is $POSTed to activate other processors that might be waiting to acquire the DCT. If the device has been drained, a message is issued to that effect. Return is then made to the caller.

HASP TIMER SERVICES

The Interval Timer Supervisor is used by the various HASP processors to record the passage of a specified period of time and to notify the requesting processor when the interval expires. This routine uses the standard OS timer features (STIMER and TTIMER) but has the additional capability to simultaneously monitor an unlimited number of intervals.

All uses of the Interval Timer Supervisor are through the HASP macro instructions $STIMER and $TTIMER. Each user of $STIMER is required to provide a 12-byte (3-word) HASP Timer Queue Element (TQE), passed via parameter register R1. $STIMER maintains a chain of all active TQEs in ascending order of interval magnitudes, with the shortest requested interval (first TQE) set on the OS STIMER queue (via a normal STIMER macro). When entered with a new interval request, $STIMER cancels the active OS timer element with a TTIMER CANCEL and reduces the interval specified in all chained TQEs by the elapsed portion of this interval. The requestor's TQE is then, after converting the requested interval to OS timer units (26 usec units), inserted into the appropriate place on the TQE chain using the first word of the TQE as a chain field. The OS timer is now reactivated with the interval in the first TQE in the chain, and return is made to the caller.

When the current OS interval elapses, the Asynchronous Exit routine in $STIMER is entered to record the expiration. The asynchronous routine reduces the intervals of all queued TQEs by the size of the just-elapsed interval, posts the Timer Processor, posts the HASP task, and returns to OS. The Timer Processor, when dispatched, will post the appropriate processors and reset the OS timer to the interval specified in the first TQE in the chain by issuing an STIMER macro.

HASP processors which have previously set an interval through $STIMER may obtain the time remaining in the interval and optionally cancel this interval through the use of the $TTIMER macro. When entered, $TTIMER cancels the active OS interval and reduces all queued TQE intervals by the elapsed portion of that interval. The requestor's TQE is then located in the queue by comparing the address of the TQE passed by the macro in register R1 to each TQE in the chain. When the correct TQE is found, the remaining time in the interval is loaded in register R0 for return to the caller. The use of the CANCEL option on the $TTIMER macro, which is indicated by register R1 containing the complement of the TQE address rather than the true address, causes the TQE to be dequeued from the chain. The OS timer is reactivated with the interval from the first TQE on queue, and return is made to the caller. Note that $TTIMER for a TQE which is not active has no effect, and a zero value is returned in register R0 as the time remaining.

This routine allocates tracks for the SPOOL volumes that were online at IPL time. The track information is stored in the caller's direct-access allocation map, in his JCT or IOT, and is also returned to the caller in register R1. The track allocation algorithm is designed to reduce seek time as much as possible.

The status of each SPOOL volume is recorded and maintained in master track group bit maps. A master map is present for each module (available SPOOL volume). Each bit in the master track group bit map represents a track group. If the bit is on, the track group is available to be allocated, and if the bit is off, the track group has already been allocated. Track group bit maps are also maintained in each JCT or IOT, but the bit definitions are opposite. Thus, if a bit is on in the JCT or IOT, the track group has been allocated to the JCT or IOT.

Track groups on the SPOOL volumes are allocated whenever the JCT or IOT has not previously acquired any tracks, or whenever all the tracks in the current track group that is allocated to the JCT or IOT have been acquired. If the JCT or IOT has already been allocated to a track group, but all the available tracks in that track group have not been acquired, the next available sequential track in the track group is allocated to the requestor. When this happens, the track information in the JCT or IOT is updated and loaded into register R1, and control is returned to the caller with the condition code set to one. This track information is recorded in the JCT or IOT in the following format: MTTR, where M is the module number (one byte), and R is the record number (one byte). The JCT or IOT track group bit map is also updated whenever a new track group is acquired. The update consists of ORing in the appropriate bit for the acquired track group in the JCT or IOT track group bit map.

When a new track group has to be acquired, seek time is reduced by searching for the nearest track group plus or minus eight track groups from the last-used track group. The last-used track group for each track group bit map is updated each time a $EXCP is issued to the volume. Each track group bit map is searched for an available track group at the last-used track group. Then, each track group bit map is searched for an available track group minus one track group from the last-used track group, then plus one from the last-used track group, and this progression continues until an available track group is found or the plus eight track group is searched. If an available track group is found, the JCT or IOT track information is updated and loaded into register R1, and control is returned to the caller with the condition code set to one. The JCT or IOT track group bit map is also updated. If an available track group is not found, the operator is notified of the out-of-track condition by the following message:

SPOOL VOLUMES ARE FULL

Then, control is returned to the caller with the condition code set to zero, and register R1 zeroed.

The Direct-Access Storage Purge Routine frees all of the SPOOL volume tracks that the job has acquired and informs the system that these tracks are available to be reacquired.

The track group bit map in the job's Job Control Table or Input/Output Table is ORed into the main track group bit map to return the job's tracks back to the system. Then the track group bit map in the JCT or IOT is zeroed to indicate that this job does not have any tracks allocated to it. The HASP Dispatcher's Event Control Field is posted to show that tracks are available to be acquired, and control is returned to the caller.

With the help of subroutine NGMAP, to obtain the number and start of track groups on each SPOOL volume, and subroutine NGBITMAP, to construct from this information a track group map segment, direct-access initialization constructs the master track group as if all available track groups were unallocated. For warm starts, overlay IOVQ will remove from the master map those bits representing track groups allocated to jobs currently in the HASP job queue (except jobs which were reading; they will be purged) and will inform the operator of the status of each active JQE. During a failure, IOVQ will give the operator a choice between rerunning the job and printing its output.

HASP TRACE SERVICES

The Trace Program is a debug facility used in HASP that is completely independent of the OS trace facility. Each time this program is called, it will insert the contents of the general purpose registers into a special Trace Table (assembled into the HASP module) to aid in the determination of HASP problems.

The Trace program is called by any routine or processor in HASP by the insertion of a $TRACE macro instruction. If the HASPGEN parameter &TRACE is set nonzero, the macro instruction will expand into an instruction that will cause a unique specification program interrupt. All program interrupts are intercepted by the HASP Trace program and the instruction that caused the interrupt is tested to determine if it is the unique instruction inserted by the $TRACE macro instruction. If the interrupt was caused by a true program interrupt, the request is sent to the first level interrupt handler, to be handled in the normal way. Otherwise, a 16-word trace entry is inserted into the HASP Trace Table.

After the Trace Table entry has been inserted and the pointers have been updated, the count of the number of times this particular $TRACE macro instruction has been executed is inserted into the first byte of the first word of the trace entry and also into the last half of the $TRACE instruction. All registers are then restored, and return is made by loading the program old PSW, which restores the condition code to its original value before the $TRACE macro instruction was executed.

The operator response of "NOTRACE" to HASP's initialization WTOR allows all tracing to be deactivated or allows only selected $TRACE macros to cause entries in the table. With this option, when a $TRACE interrupt occurs, the count field in the $TRACE macro is interrogated. If it is zero (the normal value as assembled), the $TRACE macro is replaced by a NOP so that the interrupt will not occur again from that same location. If the count field is nonzero (accomplished using a HASP REP card), normal tracing and counting are performed for that $TRACE as described above.

The symbolic location "$TRACETB" in HASP identifies a 3-word table with the following format; the first word is the address of the last entry that was made in the Trace Table; the second word is the address of the first byte of the Trace Table; and the third word is the address of the last byte of the Trace Table +1.

Each 16-word trace entry contains the following information: the first word contains a 1-byte count of the number of times this $TRACE macro has been executed and the 3-byte address of the location of this $TRACE macro; following the first word, general purpose registers 0 through 10 and 12 through 15 are stored, in that order.

HASP CONSOLE SERVICES

HASP Console Services consists of routines throughout the HASP System that are involved with the reception and delivery of messages to and from the operator. Most of these routines are concerned with the handling of the various Console Message Buffer (CMB) queues. Routines that contain significant logic for performing console services are discussed in the following paragraphs, with the exception of $HASPWTO which is discussed earlier in this manual.

CONSOLE MESSAGE BUFFER (CMB) QUEUEING ROUTINES

Routine Entry	Assembly	Functions
HASPRCC1	HASPRDR	Recognize entry of a HASP command card, copy the command into a CMB (acquired from the $FREEQUE queue) setting appropriate routing and entry source information, queue the CMB to the command processor $COMMQUE queue, and repeat the message on an Operating System console via $WTO macro.
HASPMCON	HASPRTAM	Recognize the existence of a remote console message when signaled by routines within the RTAM deblocking routines, read the command using RTAM, copy the command into a CMB (gotten from the $FREEQUE queue) setting appropriate entry source information, queue the CMB to the Command Processor $COMMQUE queue, and repeat the message on an Operating System console via $WTO macro.
$MGCRSVC	HASPCON	Recognize HASP commands when entered via the SVC 34 CVTHJES exit interface, copy the command into a CMB (gotten from the $FREEQUE or $WRESERV queue), set appropriate entry source information, and queue the CMB to the Command Processor $COMMQUE.
HASPCOMM	HASPCOMM	Recognize the queueing of HASP commands in the $COMMQUE queue, free the CMB via the $FREEMSG service routine, perform the requested function or cause it to be performed, and make appropriate responses to the console of entry or other console.
$WTO	HASPNUC	Receive control when the $WTO macro is executed by a HASP processor and provide an interface with the Console Buffering routine.

$MGCRSVC	HASPCON	Recognize the entry of the short form reply when entered via the SVC 34 CVTHJES exit interface, expand the reply, locate the TCB, and enter the SVC 35 exit routines for job association and logging on the HASP Job Log data set for the job (causing $WRESERV queue to be used if no CMBs in the $FREEQUE queue).
$WTOSVC	HASPCON	When entered via the SVC 35 and 36 CVTHJES exit interface recognize messages that are associated with HASP-controlled jobs and cause logging on the HASP Job Log via the Console Buffering routine.
HASPCBUF	HASPCON	Filter out $WTO requests of low level importance based upon values set by the $TCON command, add time stamp and job number as appropriate, set the message in an available CMB, and queue the CMB to the $LOGQUE queue for HASP Job Log data set logging or to the $BUSYQUE queue for display at a HASP remote work station or at Operating System consoles.
HASPLOG	HASPXEQ	Recognize the queueing of a CMB in the $LOGQUE queue, search the HASP execution PCEs for the one controlling the job associated with the message. Through the use of Execution Processor service routines cause the message to be included in the HASP Job Log for the job, and queue the CMB to the $BUSYQUE queue for display (if required) via the $WQUEBUF subroutine of the Console Buffering routine or free the CMB via the $FREEMSG routine. A secondary but extremely important function of this routine is to recognize the "JOBj END EXECUTION" message and signal the Execution Processor to terminate execution of the associated job.
HASPMCOM	HASPRTAM	Recognize the queueing of operator messages for display at remote work stations and either transmit the message to an online MULTI-LEAVING work station console via RTAM, SPOOL the message on the primary SPOOL volume, or discard the message as appropriate freeing the CMB via the $FREEMSG service routine. Messages queued on the SPOOL volume are transmitted to the remote work stations

		by the Print Punch Processor between printing of job output.
HASPWTO	HASPCON	Recognize the queueing of operator messages for display on Operating System consoles, convert the message into a recognizable Operating System WTO parameter list, issue the SVC 35 to display the message, and either free the CMB via the $FREEMSG service routine, queue the CMB to the $DOMQUE queue, or queue the CMB to the $WCOMRES queue for continuation of Command Processor multiline WTO requests. If a $DOM macro is executed against a message prior to actual queueing to the $DOMQUE, an Operating System DOM macro is executed and the CMB is freed. If the end line of a MLWTO is encountered, the $WCOMRES queue is emptied and the CMB is freed.
$DOM	HASPNUC	Receive control when the $DOM macro is executed by a HASP processor, turn off $DOMACT flag in the CMB (addressed by register 1), remove the CMB from the $DOMQUE queue, and execute an Operating System DOM macro to delete the message.
$FREEMSG	HASPNUC	When entered, place the CMB in the $WRESERV queue (if queue is empty) or in the $FREEQUE queue.
$WTOSVC2	HASPCON	When entered via the SVC35 (second exit) CVTHJES exit interface, edit the Operating System's normal WQE control block, adding time stamp and job number as appropriate.

HASPMCON - HASP REMOTE CONSOLE PROCESSOR

This processor processes all console messages to and from remote terminals. The routine optionally saves messages to remotes which are not signed-on MULTI-LEAVING terminals for later printing on the remote terminal printer.

The processor receives control whenever a Console Message Buffer (CMB) is placed in the $BUSYQUE queue for a remote terminal or whenever a console message is received from a remote terminal. Processing is as follows:

1. The processor first examines the output queue of messages and upon encountering a message queued for a remote terminal examines the current status of the terminal. If the terminal is not an

active BSC MULTI-LEAVING terminal, the CMB containing the message is freed, via the $FREEMSG Service routine (if $SPOLMSG=0).

2. If the message is to be written to a remote console device, a Remote Console Device Control Table is constructed for the specific remote terminal, the DCT is chained onto the other DCTs for this remote, the DCT is OPENed by calling the Remote Terminal Access Method, all queued messages are written to the terminal, and the DCT is CLOSED and unchained.

3. If the message to be written is for a currently inactive or for a non-MULTI-LEAVING active remote and if HASP operator message SPOOLing space is specified ($SPOLMSG ≠ 0), an attempt to save the message on the primary SPOOL volume for later printing at the remote by printer support routines is made. SPOOLing of messages is accomplished as follows:

   a. The remote Message SPOOLING Queue ($MSPOOLQ) element for the designated remote is examined for a queue header entry of zero. If zero, a record is allocated from the Message Allocation ($MSALLOC) Table, and the corresponding MTTR for the record is placed in both header and trailer entries for the remote. (Nonzero but equal header and trailer entries signify that the queue exists; however, since the last record of each remote element is always empty, no data is currently queued).

   b. A record is allocated from the $MSALLOC Table to represent the new end of message queue, and the associated MTTR is placed in the chain field of the current HASP buffer. The HASP buffer is then filled with the operator message, along with any more messages currently queued for the same remote, and is written on the primary SPOOL volume at the record location designated by the trailer MTTR for the remote. As each CMB is emptied, it is freed via the $FREEMSG Service routine.

   c. Upon completion of I/O, the buffer chain field replaces the trailer MTTR, indicating that the queue is not empty and providing chaining information.

   d. The above process is repeated for additional CMBs, as required to empty the $BUSYQUE queue of messages for the remote.

   In the process of allocating message records the $MSALLOC Table bit map is used. Each bit in the map, when on, represents a free record on the primary SPOOL volume. Allocation consists of finding the highest numbered bit that is on, turning the bit off, and converting to a corresponding MTTR. When all bits in the map are off, indicating that no records are available, all CMBs with messages to be SPOOLed are discarded.

4. If an input message is to be read, two CMBs are gotten from the $FREEQUE queue, a Remote Console Device Control Table is constructed, and the Remote Terminal Access Method is utilized to GET the message. The message is written to the local console, using one CMB and $WTO macro, and then is queued for the Command Processor $COMMQUE, using the other CMB.

$MGCRSVC - SVC 34 Exit Routine

The Operating System SVC 34 module IGC0403D enters the HASP SVC 34 Exit routine at entry point $MGCRSVC. When entered the SVC 34 Exit routine performs the following functions:

1. A scan of the command text portion of the SVC 34 parameter list is performed. The scan locates the beginning and length of solid text as well as performing a backspace edit (the character defined by the HASPGEN parameter $BSPACE and the preceeding character are eliminated from the text). If the resulting command becomes all blanks the return to the Operating System indicates the command to be a HASP command.

2. The resulting command is examined for the presence of the numeric (short) form of the Reply to Information Request command. If the numeric form is recognized the following is performed:

   a. The short form is expanded to R XX,'text' format.

   b. The Operator Reply Element (ORE) chain is checked for a valid reply number. If the number is in error, the resulting command is passed to the Operating System for action.

   c. Job association is attempted for the Task Control Block (TCB) associated with the ORE for the reply. If no association is made the command is given to the Operating System.

   d. A check for the availability of a Console Message Buffer (CMB) is made on the $FREEQUE queue. If no CMB is available, an attempt to move the reserved buffer from $WRESERV queue to the $FREEQUE is made. If this fails, an error return is taken to the Operating System.

   e. The SVC 35 interface and Console Buffering routines are used to move the expanded form of the reply into the free CMB, adding time stamp and job number. The CMB is queued to the Log Processor for logging on the HASP Job Log. Each message logged will be tagged with the character "R" in front of the time stamp.

   f. The command is given to the Operating System for processing.

3. A check is made for the presence of a HASP command ("$" first character). If the command is not a HASP command, it is given to

the Operating System for processing; otherwise, the following is performed:

a. A check for an available CMB is made as with replies. If no CMB is available, an error exit to the Operating System is taken. The count in $COMMCT is reduced by 1, and if the result is zero, no CMB available situation is simulated.

b. The command text is copied into the available CMB and the CMB is placed on the $COMMQUE queue. The Command Processor is $POSTed for work, and the HASP task ECB is POSTed.

Returns to the Operating System are as follows:

R15 = 0 - Command is a HASP command.

R15 = 4 - Command is an Operating System command.

R15 = 8 - Command is a HASP command with no CMB available.

$WTO - HASP Write To Operator Service Routine

The $WTO routine is entered from the various HASP processors when the expansion of the $WTO macro is executed. Upon entry, the routine disables the CPU and checks for an available Console Message Buffer (CMB). Normally, if no CMBs are available on the $FREEQUE, a $WAIT macro is issued enabling the CPU for interrupts. If, however, the request is for a UCMID-specified console (restricted to the Command Processor), the $WCOMRES queue is checked for an available CMB. If no CMBs are available and the caller desires not to wait, a return with condition code set to zero and CPU enabled for interrupts is taken. If CMBs exist, a check is made to ensure that the caller did not make a $DOMACT request and the Console Buffering routine is called to fill in the CMB and queue the message. On return, the CPU is enabled, and condition codes are set to a nonzero condition before returning to the caller. If $DOMACT is requested, the count in $COMMCT is reduced by 1 and is checked against a minimum value. If the resulting count is not below the minimum, normal processing occurs; otherwise, actions to indicate that no CMBs exist are performed.

$WTOSVC - HASP SVC 35 And SVC 36 Exit Routine

Operating System SVC 35 module IEAVVWTO and SVC 36 module IGC0303F enter the HASP SVC 35 and SVC 36 Exit routine. When entered, the routine performs the following functions:

1. The type of entry is determined, and flags are set so that SVC 36 entries in the HASP Job Log will be identified by the character "L" in front of the time stamp and all others will have a blank character.

2. Multiline WTOs (MLWTOs) are examined for the connect ID of x'FFFFFF', and if encountered, control is returned to the Operating System with the deletion return code. Otherwise, the number of lines to display is determined (one assumed for all non-MLWTO entries).

3. Line one of the request is checked to determine possible deletion. If the message is to be deleted, control is returned to the Operating System with the deletion return code. Otherwise, the job association routine is entered; this routine attempts to associate the message with a HASP-controlled job, either through the TCB or job name in the text of the message. If no association is made, control is returned to the Operating System for normal processing.

4. A check is made to ensure that Console Message Buffers (CMB) are available. This is done by disabling the CPU and checking the $FREEQUE queue. If no CMBs are available, a check is made to determine if the task is in a WAITable status. If it is not WAITable, control is passed to the Operating System and the message will be omitted from the HASP Job Log. Otherwise, an ENQ enqueues on a "qname" of "SYSHASP" and "rname" of "CMB". When the Operating System gives control to the task, a check is again made to see if CMBs are available; if not, a WAIT macro is issued (a wait element is filled out for HASP to POST). When POSTed the routine loops back to the second CMB check. If CMBs are available, the enqueue is released by the DEQ macro and the entire function is repeated.

5. When CMBs are available, the Console Buffering Routine is entered to copy the message into a CMB (adding time stamp and job number) and to queue the CMB to the $LOGQUE queue for inclusion in the HASP Job Log for the job. On return, the HASP task is POSTed, the CPU is enabled, and a check is made for additional lines (MLWTO). If there are more lines, steps 4 and 5 are repeated until all requested lines are queued to the $LOGQUE queue. Otherwise, control is returned to the Operating System for processing.

This interface with the Operating System uses registers of the calling modules during enabled state processing. Register usage in the enabled state is as follows:

0 = UCMID/CONNECT ID/WORK	8 = START OF MESSAGE
1 = PARAMETER LIST/WORK	9 = NOT USED
2 = END OF MESSAGE POINTER	10 = SAVE FOR REG 0
3 = CVT ADDRESS	11 = NOT USED
4 = TCB ADDRESS	12 = NOT USED
5 = NOT USED	13 = SAVE FOR REG 1
6 = NUMBER OF LINES	14 = RETURN AND FLAGS
7 = LENGTH OF MESSAGE	15 = BASE

$WTOSVC2 - HASP SVC 35 EXIT 2 ROUTINE

The Operating System SVC 35 modules IEAVVWTO and IEAVMWTO enter the HASP SVC 35 Exit 2 Routine. When entered, the routine performs the following functions:

1. Edits the Operating System's WQE for single line messages that are not identified as HASP messages (TCB is not the HASP communications task), aligning the start of text lines.

2. Inserts time stamp to all edited messages.

3. Inserts the HASP job number if the message is associated with a HASP-controlled job.

HASPCBUF - CONSOLE BUFFERING ROUTINE

The Console Buffering Routine fills out an available Console Message Buffer (CMB) with control and message text information for inclusion in the HASP Job Log and/or display at a HASP work station or an Operating System Console. It is entered from the $WTO Service routine or the SVC 35 and 36 Exit routine (indirectly from SVC 34 Exit routine through the SVC 35 Exit routine). When entered the routine performs as follows:

1. If the console type is not a remote, Operating System UCMID log only, or logical routing with $DOMACT flag set, the logical routing and list levels are compared with elements in the WCNLSTBL Table in an attempt to eliminate the message. The logical console is left included if the list level of the message is higher than the corresponding list level in the logical console entry in the table. If the resulting logical consoles indicate no output is required and job numbering is not desired the $WTO request is ignored and the routine returns to the caller. (The WCNLSTBL Table is maintained by the operator via the $TCON command).

2. The first CMB in the $FREEQUE queue is addressed. The HASP time stamp is placed into the CMB (branch entry to the Operating System Time routine is used.) Identifications are inserted into the first text character as setup by callers:

   $ - HASP $WTO R - SVC 34 Exit

L - SVC 36 Exit Ø - SVC 35 Exit

3. If job number is desired, the job number is moved into the CMB from the HASP Job Control Table (JCT). (Register 10 must address the JCT at the time of execution of the $WTO if job numbering is requested.)

4. The message text is moved into the CMB and if $DOMACT is requested the return save area is set so that R1 will address the CMB upon return to caller.

5. The CMB is removed from the $FREEQUE queue and the CMB is queued to the $LOGQUE queue (if job or log only is requested) or to the $BUSYQUE queue (otherwise). Control is returned to the caller.

HASPLOG - HASP LOG PROCESSOR

On initial entry from the HASP Dispatcher the Log Processor enters the overlay CSECT HASPXLOG via the $LINK macro. The routine examines the $LOGQUE queue for the existence of Console Message Buffers (CMB) containing messages for a HASP controlled job's HASP Job Log. If no CMBs are queued the processor $WAITs for work. The Console Buffering Routine causes the Log Processor to be $POSTed for work whenever a CMB is queued to the $LOGQUE queue. When entered with work to do the Log Processor performs the following functions:

1. The processor searches the Execution Processor Control Elements (PCEs) for a processor that is currently controlling the job identified in the message job number field. If no PCE is found, logging functions are skipped (steps 2 and 3).

2. If the job allows HASP job logging, the jobs Data Definition Block (DDB) for the log is located and preparations for writing are made as required:

   a. A buffer is assigned to the DDB if not already available.

   b. A title is moved into the buffer if not already done for the job.

   c. The Execution Processor is instructed to write the buffer if full or if the buffer is to be truncated.

3. The message is moved to the buffer. If the message is a HASP "JOB j END EXECUTION" message, the DDB is flagged for termination and the Execution Processor is notified allowing termination of the execution phase for the job.

4. The CMB is removed from the $LOGQUE queue and placed in the $BUSYQUE queue if logical consoles identifications are present; otherwise, the CMB is freed via the $FREEMSG routine. Control is returned to the beginning of the processor.

$HASPWTO - HASP Communications Subtask

Described earlier in this manual.

$DOM HASP Delete Operator Message Service Routine

The $DOM routine is entered by a processor that has previously executed a $WTO with the class parameter $DOMACT specified. Register R1 contains the address of the Console Message Buffer (CMB) that contains the message displayed by the $WTO and, after issuance of the Operating System WTO SVC, contains the DOM ID for an Operating System DOM SVC. When entered via execution of a $DOM macro (register R1 contains address of the CMB) the $DOM routine performs the following functions:

1. The $DOMACT flag in the CMB is turned off.

2. The count in $COMMCT is incremented, and if the count goes from a minimum value to one above minimum, CMB is $POSTed.

3. The $DOMQUE queue is searched for the CMB addressed by register R1 on entry to the routine. If found, the DOM ID is removed, an Operating System DOM SVC is issued for the message, and the CMB is freed via the $FREEMSG routine. Control is returned to the caller. (If $DOMQUE queue does not contain the CMB the HASP Communication subtask will issue the DOM.)

$FREEMSG - HASP FREE CONSOLE MESSAGE BUFFER SERVICE ROUTINE

The HASP Free Console Buffer (CMB) Service routine is entered (with the CPU in a disabled state) from various console service routines whenever a CMB is to be made available. When entered, the $FREEMSG routine performs the following functions:

1. The $WRESERV queue is tested for the presence of a reserved CMB. If there is no reserved CMB, the CMB to be freed is placed in that queue and control is returned to the caller with condition codes set nonzero.

2. The CMB is LIFO queued to the $FREEQUE, and a test is made to see if any routine could be waiting to use a CMB. If not, control is returned to the caller with condition codes set nonzero. If waiting HASP processors are possible, a general $POST for CMB is executed and control is returned to the caller with condition code set to zero.

3. If a task is waiting via the $WTOSVC routine, the waiting ECB is validated and POSTed using the branch entry to the Operating System Post routine (entry through CVTOPT01). Control is return to step 2 above (no task can be waiting now).

DISASTROUS ERROR HANDLER

This routine is entered from a processor whenever a critical SPOOL disk error is detected. The operator is notified of the error, and processing continues, although the operator should re-IPL the system with a cold start as soon as possible.

When this routine is entered, a $WTO is issued to notify the operator of the error, and control is returned to the calling processor. The message to the operator is as follows:

```
DISASTROUS ERROR - COLD START SYSTEM ASAP
```

CATASTROPHIC ERROR HANDLER

This routine is entered whenever an unrecoverable error is discovered by HASP. The operator is informed of the error and given an error code, and the system enters a 1-instruction enabled loop. The error codes and their meanings are listed in the HASP Operator's Guide, Appendix A.

When this routine is entered, register R0 contains the address of a 4-byte field containing the 3-character error code left-justified. The 4-byte error code field is moved into the operator message. This message is then written on the operator's console using an ordinary OS WTO macro:

```
$ HASP SYSTEM CATASTROPHIC ERROR.  CODE = xxx
```

After this message is typed a 1-instruction loop is executed. If HASP is abended by OS for any reason, a STAE exit (previously set by HASP Initialization) is entered. This exit in turn calls the Catastrophic Error Handler which issues the above message with the 4-byte code "ABND". The registers at this time are as documented (in appropriate OS/VS2 documentation) for entry to STAE exits, except that register R0 is moved to register R2.

INPUT/OUTPUT ERROR LOGGING

This routine is entered whenever an unrecoverable input/output error occurs on a HASP direct-access intermediate storage device, or whenever line errors occur which may require the attention of the operator. A message is generated describing the error, and this message is routed to the operator via the operator's console. The routine then returns without taking any further action.

When this routine is entered, register R1 contains the address of the Input/Output Block (IOB) which is associated with the input/output operation in error. The channel status, channel command code, sense information, track address, and line status are retrieved from the IOB and are formatted; the unit address and volume serial are obtained from the Unit Control Block (UCB); the device name (if applicable) is

acquired from the Device Control Table (DCT); and the message is written to the operator's console.

The formats of the two messages are described in the HASP Operator's Guide, Appendix A.

HASP OVERLAY SERVICES

These routines, together with the Overlay Roll Processor, respond to calls from other HASP processors when the macros $LINK, $LOAD, $XCTL, $RETURN, and $DELETE are executed in HASP coding. This enables certain executable and table portions of HASP coding (assembly control sections created by use of the $OVERLAY macro) to be brought into main storage from their normal direct-access residence for use during HASP execution.

Major objectives of Overlay Service and Roll logic are: to allow multiple processors to use a single copy of the same overlay routine simultaneously and to prevent any system lockout due to $WAITs in overlay routine coding.

The overlay data set is constructed as part of HASP installation by the HASP Overlay Build utility and is referred to by the DDname OLAYLIB in the job which invokes HASP.

All Overlay Service and Roll Processor coding is located in module HASPNUC. Service entry points are addressable by register BASE1 and are referenced by macro expansions through the HASP Communication Table.

$LINK SERVICE

On entry, register R15 contains the address of the next instruction after $LINK, and register LINK contains the called routine's OCON. An OCON is an index into the HASP Overlay Table, which is the control section HASPOTAB created by the HASP Overlay Build utility. HASPOTAB's individual entries are defined in OTBDSECT, created by the $OTB macro.

The calling processor's registers R0-WC are saved in the caller's PCE. Overlay Service base address is established in register WC. Register R15 is saved in PCEORTRN. R15 is set to the relative displacement of the called routine entry point from the beginning of an Overlay Area IOB, i.e., OACEPROG-BUFDSECT. The called routine OCON is saved in PCEOCON and is then used to compute the address of the Overlay Table entry for the called routine. If &DEBUG is set to YES, field OTBCALLS is incremented by one. The called routine's priority is moved to PCEOPRIO.

If the Overlay Table indicates that the called routine was made a permanent part of the HASP Load Module at Overlay Build time, register BASE3 is loaded with the address of a theoretical overlay area containing the resident routine (BUFSTART-BUFDSECT bytes prior to the routine itself), caller's R0-WC are reloaded, and control is passed to the called routine at its entry point.

If the called routine is not permanently resident, a search is made of all overlay areas in the system. If the called routine is found in an area (PCEOCON equal to area's OACEOCON), the caller's PCE is added to the chain of all active users of the area. This chain begins at OACEPCE and continues through PCEOPCE of each PCE (if several users are on the chain) and ends with a zero chain word. A test is made for illegal nested $LINK if &DEBUG is set to YES; see The HASP Operator's Guide for error message. If the called routine is in process of being read into

the area from direct access, the calling processor is made to $WAIT on OLAY, to be later activated by the Overlay $ASYNC Exit. Otherwise, caller's R0-WC are reloaded and control is passed to the called routine entry point, with register BASE3 containing the address of the Overlay Area IOB for use as the overlay routine base address.

If the called routine is not found while searching all overlay areas, the search attempts to find an overlay area which is not currently in use. It may contain an overlay routine but may not have active users (OACEPCE must be zero). The inactive area containing the routine of lowest priority (OACEPRIO) will be used, subroutine OLOD will be called to start reading the called routine from direct access, and the calling processor will be $WAITed on OLAY, to be later activated by Overlay $ASYNC Exit.

If no inactive areas are found, the calling PCE is placed on a queue waiting for an overlay area. The Queue begins at the word $WAITACE, continues in descending priority order by PCEOPRIO using chain word PCEBASE3, and ends with a zero chain word. If several PCEs are on the queue requesting the same overlay routine (PCEOCONs equal), only the first PCE is on the above chain, the others are chained from it using word PCEOPCE. All PCEs in the queue are $WAITed on OLAY. This queue is emptied by the Overlay Roll Processor or by the OEXIT subroutine.

$LOAD SERVICE

$LOAD shares almost all logic with $LINK. Entry register conditions are identical to those for $LINK.

R15 is not saved in PCEORTRN. R15 is not set to the relative entry point of the called routine.

When the called routine is found in an overlay area or is read into one by later system actions, R15 still contains the address of the next instruction after $LOAD. Subsequent use of R15 as an absolute entry point results in control being returned to the caller following the $LOAD macro, with the routine in an actual or theoretical area, addressable by BASE3 as with $LINK.

$XCTL SERVICE

$XCTL logic shares almost all logic with $LINK. Entry register conditions are identical to those for $LINK.

R15 is not saved in PCEORTRN. $XCTL is legal only when it logically follows another $XCTL or an original $LINK. Subsequent $RETURN uses PCEORTRN as stored by the original $LINK to return control from Overlay Service to the original caller.

Before doing entry actions for the new called overlay routine, the OEXIT subroutine is called to remove the calling processor's PCE from the chain of users of the current overlay routine.

$RETURN SERVICE

On entry, register LINK points to the next instruction after $RETURN and also contains the condition code and program mask as set by a BAL instruction. BASE3 points to an actual or theoretical area containing the current overlay routine.

Caller's R0-WC are saved in the PCE. Overlay Service base address is established in WC.

The OEXIT subroutine is called to remove the caller's PCE from the chain of users of the current overlay routine.

Returned condition code is reestablished using an SPM instruction. Caller's R0-WC are reloaded. Control is returned to the address previously saved in PCEORTRN by $LINK.

$DELETE SERVICE

$DELETE is nearly identical to $RETURN, except that it is used to release control of an overlay routine previously $LOADed.

On entry, register LINK points to the next instruction after $DELETE. This is stored in PCEORTRN, and all actions described for $RETURN are performed.

OEXIT Subroutine

This subroutine is used by service routines for $XCTL, $RETURN, and $DELETE to release use of the current overlay routine by the calling processor. On entry, register WA contains the subroutine return address and register BASE3 contains the address of an actual or theoretical (permanently-resident routine) overlay area containing the current overlay routine.

If the current overlay routine is permanently resident, OEXIT returns immediately. Otherwise, the chain of all users of the area (beginning at OACEPCE and continuing through PCEOPCE) is searched and the caller's PCE is removed. If other processors are still using the area, OEXIT returns.

If the above actions result in the overlay area becoming inactive (OACEPCE equal zero), the $WAITACE queue is inspected. If PCE(s) are waiting, the top priority group of one or more requesting the same overlay routine is dequeued, the address of the first such PCE is placed in register R1, and OEXIT simply falls through to the OLOD subroutine, which eventually returns to the caller of OEXIT.

OLOD SUBROUTINE

This subroutine is used by service routines for $LINK, $LOAD, and $XCTL; by the Overlay Roll Processor; and indirectly by users of the OEXIT subroutine. Its purpose is to start a read for a requested overlay routine from the direct-access device containing the overlay data set. On entry, register WA contains the subroutine return address, register BASE3 contains the address of an actual overlay area to be used, and register R1 contains the address of the first of a group of one or more PCEs requesting the same overlay routine, chained from the first PCE by PCEOPCE.

OACEPCE of the Overlay Area is pointed to the first PCE. OACEPRIO and OACEOCON are set to indicate the routine that will reside in the area. The Overlay Table entry for the requested routine is accessed and, if &DEBUG is set to YES, field OTBLODS is incremented by one.

The relative T and R in the overlay data set of the requested routine is obtained from the Overlay Table. The address of the Overlay DCT is loaded into register R1. If the overlay data set is on any SPOOL volume (device type DA in the DCT), an absolute form of MTTR is computed and stored in DCTSEEK. This conforms to $EXCP requirements for SPOOL volumes and allows $EXCP to remember SPOOL arm positions. If the overlay data set is on a non-SPOOL direct-access volume, the standard OS form of MBBCCHHR is computed and stored in IOBSEEK.

Hardware read operation is requested by using the $EXCP macro. The Overlay DCT specifies that when the read operation is complete, Overlay $ASYNC Exit is to be entered. All PCEs chained from OACEPCE are already $WAITing OLAY, to be later activated by Overlay $ASYNC Exit. OLOD then returns to its caller or caller of OEXIT.

OVERLAY $ASYNC EXIT

This routine is entered when (under control of the Asynchronous Input/Output Processor ($ASYNC) PCE) an overlay read operation (started by OLOD subroutine) is posted complete. On entry, register R1 points to the overlay area. BASE2 is set to the base value for the Overlay Roll Processor, which is used for local addressability. R15 contains the return address to $ASYNC.

The chain of all users of the overlay routine just read (begins at OACEPCE, continues through PCEOPCE) is processed. Each PCE's reentry address (R15, now stored in PCER15) is made absolute by adding the address of the overlay area, if the value in PCER15 is determined to be relative. The address of the overlay area is also stored in each PCEBASE3, to provide addressability when the Dispatcher activates each processor. The function $POST for OLAY is performed on each PCE to make it dispatchable.

If OS IOS has posted the read complete with a permanent I/O error, each PCE's reentry address (PCER15) is pointed to a routine which types the message "UNREADABLE OVERLAY - ..." and enters a permanent $WAIT. The

overlay area is freed for other use, and the OEXIT subroutine is called to start any queued requests.

If &OREPSIZ is set to zero, this exit returns to $ASYNC. Otherwise, the overlay REP storage area is examined to see if any REPs that apply to this overlay routine were read during HASP Initialization. REPs whose CSECT name (last four characters) match OACENAME are applied. The assembly origin (OACEASMO) of the routine is subtracted from the REP address and the BUFSTART address of this overlay area is added to determine the storage location to be patched.

Return is finally made to $ASYNC to allow other processing to continue. The Dispatcher will enter each processor using the overlay routine just read.

HASP REMOTE TERMINAL ACCESS METHOD

The Remote Terminal Access Method provides an interface between the HASP Processor and the remote terminal. RTAM provides blocking/deblocking, compression/decompression, and synchronization with the remote terminal in such a way that the processor need not be concerned with the characteristics of the remote with which he is communicating. The MULTI-LEAVING Line Manager interfaces very closely with RTAM through a series of subroutines, the more important ones of which are briefly described below.

The Remote Terminal Access Method consists of four main sections and some miscellaneous subroutines. This section discusses the four main sections: OPEN, GET, PUT, and CLOSE.

OPEN

The OPEN routines convert the line from an idling mode of operation to a transmit or receive mode of operation. In the case of the MULTI-LEAVING interface, this routine also generates the request or permission to begin a new function.

GET

The GET routines convert data received from the line into EBCDIC images suitable for processing by the HASP processors. This conversion includes deblocking, decompression, and conversion from line code to EBCDIC.

PUT

The PUT routines convert data from EBCDIC into a form ready to be transmitted to the remote terminal. This conversion includes compression, blocking, and conversion from EBCDIC to line code.

CLOSE

The CLOSE routines convert the line from a transmit or receive mode of operation to an idling mode of operation.

The following sections describe the primary subroutines used by the Remote Terminal Access Method and the MULTI-LEAVING Line Manager.

MSIGNON -- SIGNON CARD PROCESSOR

This subroutine receives the address of a /*SIGNON card in register R1. If the line used to read the SIGNON card was defined as a dedicated line, the SIGNON card is ignored and the subroutine returns immediately. If the line is a nondedicated line, the MABORT and MDISCON subroutines are called to disconnect any other remote that may have been attached to this line. The password is then checked, and if it is not valid, an error message is issued and the subroutine returns. If the password is valid the specified Remote Terminal's DCTs are located and examined. If the specified remote is already attached to another line or if the specified remote is not locatable, the subroutine issues an error message and returns. Otherwise, the specified remote is attached to the line, an SMF record is written, and a confirmation message is issued.

MINITIO -- MULTI-LEAVING INPUT/OUTPUT INTERFACE

This subroutine analyzes the status of a MULTI-LEAVING remote terminal and takes appropriate action to minimize degradation while ensuring maximum line throughput. The subroutine first establishes the status of every processor currently active on the MULTI-LEAVING line. Then, based on the active input processor count, the active output processor count, the status of the remote terminal, and the status of input and output buffers queued within HASP the subroutine either transmits an ACK0 to the terminal, transmits a text buffer to the terminal, or initiates a 1-second delay.

MEXCP -- REMOTE TERMINAL INPUT/OUTPUT INTERFACE

This subroutine interfaces the Remote Terminal Access Method with the standard HASP "$EXCP" Input/Output Interface. In addition to initiating I/O, this subroutine also provides the MULTI-LEAVING Block Control Byte sequence count, and the BSC 2270/2780/3780 parity check (ACK0-ACK1) conversion.

MCCWINIT -- CHANNEL COMMAND WORD SEQUENCE SETUP SUBROUTINE

This subroutine is passed a sequence type in bits 24-27 of register R1. The subroutine then constructs a CCW chain, based on this value, and returns. The following table depicts the various CCW sequences which can be constructed by this subroutine.

HASP REMOTE TERMINAL CCW SEQUENCES

BSC Prepare Sequence (Code = C)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	DISABLE	0	60	C0	1
IOBCCW2	SET MODE	LCBMCB	60	C1	1
IOBCCW3	ENABLE	0	60	C2	1
IOBCCW4	NOP	MBSCSYN	60	CA	4
IOBCCW5	NOP/WRITE	MBSCENQ/MBSCEOT	60	CA	1
IOBCCW6	READ	TPBUFST	20	C4	&TPBFSIZ

BSC MULTI-LEAVING Terminal Sequence (Code=9)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	92	1
IOBCCW2	NOP	MBSCSYN	60	99	4
IOBCCW3	WRITE	LCBRCB	60	99	2
IOBCCW4	READ	TPBUFST	20	94	&TPBFSIZ
IOBCCW5	NOP	MBSCSYN	60	98	4
IOBCCW6	WRITE	TPBUFST	60/A0	98	*-*
IOBCCW7	WRITE	METBSEQ	60	98	2
IOBCCW8	READ	TPBUFST	20	B4	&TPBFSIZ

HASP REMOTE TERMINAL ACCESS METHOD

BSC Hardware Terminal Read Sequence (Code = 8)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	82	1
IOBCCW2	NOP	MBSCSYN	60	89	4
IOBCCW3	WRITE	LCBRCB	60	89	2
IOBCCW4	READ	TPBUFST	20	84	&TPBFSIZ

BSC Hardware Terminal Write Sequence (Code = A)

CCW	Command	Data Address	Flags	Internal Code	Byte Count
IOBCCW1	ENABLE	0	60	A2	1
IOBCCW2	NOP	MBSCSYN	60	AA	4
IOBCCW3	WRITE	MBSCENQ	60	AA	1
IOBCCW4	READ	LCBRCB	20	A6	2
IOBCCW5	NOP	MBSCSYN	60	A8	4
IOBCCW6	WRITE	TPBUFST	60	A0	*-*
IOBCCW7	WRITE	METBSEQ	60	A8	2
IOBCCW8	READ	LCBRCB	20	A5	2

HASP INITIALIZATION

HASP Initialization builds control blocks and performs all other preparations necessary for HASP job processing. Initialization is designed to provide either a cold or warm starting capability. A cold start is one which starts the system anew. Only those jobs entered after a cold start will be processed. A cold start does not have any configuration requirements except as defined in the HASP generation parameters. A warm start is a restart. Checkpointed information is read from the SPOOL1 volume and queued jobs and data from the last processing are recovered. This type of start requires, as a minimum, that the SPOOL volumes used during the previous execution be online. Extra SPOOL volumes, up to a total of &NUMDA volumes, may be added.

ENTRY ACTIONS

Since HASP Initialization resides in the same area as the main HASP buffer pool (as designated by the HASP parameter &NUMBUF) and portions of the initialization routines are executed from overlay control sections, all HASP processors except those required for initialization and console processing are placed in the HOLD status. The Command Processor PCE is altered to refer to the root segment of HASP Initialization, which resides in the data portion of the first buffer.

The HASP Initialization WTOR is then displayed via OS WTOR facilities. Initialization waits for the operator to respond with the desired options. The options are compared against the Initialization Options Table, and the appropriate bits in the $OPTSTAT field in the HCT are set or reset in accordance with the options specified. If any option is incorrectly entered, an error message is issued and the $OPTSTAT field is set to the default option configuration. (Refer to Starting The HASP System in the HASP Operator's Guide.)

The HASP Initialization SVC is then invoked using an assigned code for the OS type 1 extended SVC router facility. The address of the HASP Vector Table ($HVT) is passed to the SVC as a parameter. The SVC stores this address in the CVT (at CVTHJFS) to activate OS/VS2 exits to HASP. The SVC places the HASP task in supervisor mode and returns a list of OS Nucleus addresses which HASP saves in the HCT for later use. After return from the SVC, HASP establishes a STAE so that the Catastrophic Error Routine will be entered if the main HASP task is abended.

If requested by the operator, the HASP REP Routine is entered for optional alteration to the resident portions of OS or any part of HASP.

MANDATORY FIXING

The HASPNUC CSECT contains all coding and tables which must be fixed in real storage during HASP processing. This includes the HCT, Dispatcher, I/O Appendages, and all PCEs. Other frequently used services subroutines (e.g., Queue, Buffer, Unit Services) are included even though it is not mandatory that they be fixed. The OS FIX SVC is called to fix each 4K page of HASPNUC.

BUILDING DCBS AND DEBS

The amount of space necessary for DCBs and DEBs for all unit record devices, internal readers, and RJE lines is determined from the HASPGEN parameters. Space for one direct-access DCB and DEB, with &NUMDA+1 extents, is added to the total. This total amount of space is gotten from OS subpool 254, which is in LSQA and long-term fixed. The DCBs and DEBs are initialized, chained together as if OPENed, and connected to the HASP DCTs for the appropriate devices.

Because the HASP ECB and I/O Appendages are fixed as part of the HASPNUC CSECT and because DCBs and DEBs are built in fixed LSQA, the only things which must be fixed when HASP requests an I/O operation are the IOB, CCWs, and data area. These are discussed later under Buffer Building.

PREPARATION OF OVERLAY SERVICE

The Overlay DCT is prepared by indicating that it is in use, that it is used only for reading, that Overlay $ASYNC Exit is to be entered on completion of any operation started by using Overlay DCT, and that the Overlay Roll Processor is the owner of the DCT.

The overlay data set is described by a DD card with ddname OLAYLIB. DEVTYPE and OPEN macros are used to determine the number of tracks/cylinder of the overlay volume and data set extent, which is placed as the last (&NUMDA+1) extent in HASP's single multi-extent direct-access DEB. The overlay data set is closed, since HASP uses its own constructed I/O control blocks.

The overlay data set UCB address is stored in a table used to withdraw or abort HASP, and the UCB is made allocated, permanently resident, and private.

The number of tracks/cylinder and extent are used to compute a beginning absolute TT of the overlay data set, which is stored in the Overlay DCT for later use by the OLOD subroutine.

LOCATING SPOOL VOLUMES

All OS UCBs are searched via the UCB lookup table, and direct-access volumes with volume serials of SPOOLx are examined for use for HASP SPOOL volumes. As each device is examined, the UCB is allocated by turning on the private, reserved, permanently resident, and allocation indicators. The UCB locations and the sixth volume serial character are saved in a temporary work area for later reference. If, during the UCB search, multiple volumes with the same serial or too many SPOOLx volumes are found, an error message is displayed, SPOOL volume UCBs are deallocated, and the HASP job is terminated. Upon completion of a successful allocation of SPOOL volumes, control is passed to Direct-Access Initialization.

DIRECT-ACCESS INITIALIZATION

Direct-Access Initialization (NGDAINIT) gains control after all SPOOL devices have been found by initialization; initialization has built a table of 6-byte entries (NSPOOLL1) describing the direct-access devices on which SPOOL disks are mounted. Each table entry appears as follows:

0	1	2	3	4	5
dev	vol	ucb		unused	

where:

dev = the low-order byte of the direct-access device type

vol = the low-order byte of the volume serial number

UCB = the device's UCB address

Before checking for warm start, NGDAINIT establishes where the checkpoint record is to be placed on SPOOL1. To do this, it first calls the DEB/TED setup routine to establish certain statistics about all mounted SPOOL volumes and then issues an OBTAIN macro instruction for SYS1.HASPACE on SPOOL1. The checkpoint information will reside on the first three tracks of this data set; accordingly, NGDAINIT sets up the necessary channel programs using the OBTAINed information.

WARM START

If the operator requested a warm start, NGWARM reads the checkpoint information directly into the area from which the Checkpoint Processor will write it; the information consists of the HASP job queue, printer/punch checkpoint information, miscellaneous status information (including direct-access checkpoint information), the Job Information Table (JIT), and the Job Output Table (JOT). The direct-access checkpoint information, $DACKPT, consists of &NUMDA 6-byte entries of the following form:

0	1	2	3	4	5
dev	vol	ssss		eeee	

where:

dev = the low-order byte of the direct-access device type

vol = the low-order byte of the volume serial number

ssss = the starting absolute track number of data set SYS1.HASPACE on the indicated SPOOL volume

eeee = the ending absolute track number of the first extent of data set SYS1.HASPACE on the indicated SPOOL volume

For SPOOL1, the starting track number excludes the checkpoint tracks.

NGWARM ensures that each volume specified in the direct-access checkpoint is mounted and, with the help of subroutine NGALLOC, that its extents are unchanged. If not all volumes are mounted, if any extents have been changed, or if a cursory check of a volume shows that it is not properly formatted, NGWARM writes a message and sets a quit switch to cause HASP to quiesce.

If all volumes specified by the direct-access checkpoint are correct, NGWARM checks for (and formats if necessary) newly-mounted volumes. Then it again calls subroutine NGDEBSET to allow for the possibility that the order of SPOOL volumes in NSPOOLL1 (by unit address) may not have been the same as in $DACKPT; the final order is that of $DACKPT.

Now NGWARM relocates the HASP job queue, if necessary. The job queue as recorded in the checkpoint record contained main storage addresses; if HASP does not now occupy the same storage locations as it did before, each main storage address in the HASP job queue (and in pointers to the job queue) must be adjusted to reflect the current main storage location of the job queue.

The subroutines NGMAP and NGBITMAP are called, to compute the total number of track groups in all SPOOL extents, and initialize the track group bit map as if all groups are available. Track groups belonging to jobs in the warm started queue are claimed as allocated by the HASPIOVQ overlay, described later.

Finally, NGWARM gives control to NGEXIT. NGEXIT assembles and format-writes the checkpoint information; restores the HASP appendage table pointer in $DADEB1, the HASP multi-extent direct-access DEB; counts the number of allocated track groups (one-bits) in the track group map. If RPS support was specified at HASPGEN time, a table is constructed in subpool 254 which will be used by HASP I/O Services ($EXCP) to determine the sector numbers of records in SPOOL and overlay extents. Control passes to Activation of Overlay, described later.

COLD/FORMAT START

If the operator specified cold or format start, NGCOLD first zeros out the track group map. Then NGCOLD processes each mounted SPOOL volume.

For each volume, NGCOLD uses subroutine NGALLOC to process the DSCB for SYS1.HASPACE. This subroutine issues the OBTAIN macro instruction to retrieve the DSCB. If OBTAIN's return code is not zero, an appropriate error message is printed via WTO. If the return code is zero, NGALLOC computes and saves lower and upper absolute track numbers.

If NGALLOC operated normally, NGCOLD now tests for an operator specification of COLD; if the test is positive, NGCOLD calls subroutine NGREADCT to read and validate the count field of the first record of the last track of the first extent of SYS1.HASPACE on the volume. If the count field is invalid or if the operator specified FORMAT, NGCOLD calls

NGFORMAT to format the first extent. NGFORMAT issues an unconditional GETMAIN for storage to be used in building a formatting channel program and data. NGFORMAT then builds the program and data, and formats each track by calling NGEXCP, which issues an EXCP, issues a WAIT and checks the post code.

After the volume has been inspected (and formatted if necessary), NGCOLD calls NGMAP to calculate the number of track groups in this volume and the track group number of the first track group. NGCOLD increments the overall number of track groups available for allocation by the quantity returned from NGMAP and calls NGBITMAP, which turns on in the master track group map the bits corresponding to available track groups on this volume. Then NGCOLD processes the next volume.

When all volumes have been processed, NGCOLD refreshes certain checkpoint information (the HASP job queue, the print checkpoint information, and some miscellaneous checkpoint information) and gives control to NGEXIT, as described previously at the end of Warm Start.

The DEB initialization subroutine, NGDEBSET, initializes certain HASP and OS control blocks and allows a great degree of SPOOL device independence.

When called, NGDEBSET first puts into $DADEB1 the address of the HASP TCB; it also changes the DEB appendage address to point to the standard IOS appendage. (The appendage address is restored by NGEXIT.) NGDEBSET checks for SPOOL1 and quiesces HASP if SPOOL1 is not found. Then NGDEBSET processes the SPOOL volumes.

For each volume, NGDEBSET calculates the number of records per track using information from the device characteristics table IECZDTAB in the OS nucleus and the formula given with the DEVTYPE macro instruction in the OS/VS Data Management for System Programmers. Then it sets up certain information in an entry of the Table of Extent Data (TED).

After setting the UCB address in $DADEB1, NGDEBSET performs the same functions for the remaining volumes and returns to the caller.

ACTIVATION OF OVERLAY

If the overlay data set is contained on a SPOOLx volume, the Overlay Device Control Table is adjusted so that $EXCPs done by the OLOD subroutine will use MTTR addresses and M, which refers to the DEB extent for the SPOOLx volume rather than the overlay data set extent. The first Processor Control Element (PCE) in the HASP chain is connected to the OS save area chain and, with register 13 pointing to the first PCE, Initialization enters the HASP Dispatcher as though the first processor had executed a $WAIT macro. The HASP Dispatcher will run the PCE chain and dispatch the initialization root segment. The root segment will $LINK to the first overlay control section, HASPIOVA.

UNIT RECORD INITIALIZATION - HASPIOVA

HASP INITIALIZATION

The OS UCBs are scanned for unit record devices. HASP pseudo devices are identified by a low order one bit in the UCBATI byte. Otherwise, the devices are considered to be real devices.

Pseudo Device Initialization

Pseudo devices are varied online. Pseudo 2520 devices are identified and matched with an Internal Reader (INTRDR) Device Control Table which is initialized for processing as with real devices

Real Unit Record Device Initialization

Each device is matched with a corresponding Device Control Table which is initialized for processing. If the device is allocated by OS, busy, or if a TIO indicates not operational or CSW stored, the DCT will remain in the DRAINED status, causing HASP not to use the device unless the operator starts the device by command. Automatic starting reader attention index (UCBATI) values are set to enter the HASP Attention Exit (described as part of HASP Initialization SVC). If more real unit record devices of each particular type are found than there are available DCTs, an error message is displayed and the additional devices are ignored.

If &NUM3800 is greater than zero, then the STPT subtask is ATTACHed. If there is an error in the ATTACH processing, the operator is informed and processing continues. For each 3800 printer, a Device Control Table Extension (DCTE) is GETMAINed from the HASP region. If no MDR buffer exists for the 3800 UCB, then one is GETMAINed and its address placed in the UCB extension. If an error occurs in the GETMAIN for the DCTE, the operator is informed and processing continues.

Control is then passed to the Remote Job Entry or Miscellaneous Initialization routines, as appropriate, via a $XCTL macro.

REMOTE JOB ENTRY INITIALIZATION - HASPIOVR

Line Initialization

The OS UCBs are scanned for Binary Synchronous Communication Adapter devices. The UCBs found are first matched with one or more DCTs and corresponding line descriptions (LINEmm HASPGEN parameter). Any DCT with a line description which specifically designates the UCB will be initialized for the UCB. If no line description designates the UCB, a DCT with a line description of "***" will be located and initialized. Line devices will not automatically be started.

Remote Device Initialization

Remote Device Control Tables are connected and initialized with information contained in the corresponding remote description (RMTnn HASPGEN parameter). Each group of RMr.RDn,...,RMr.PRn,...,RMr.PUn,... for a given remote are chained together for control by the MULTI-LEAVING Line Manager and RTAM. In addition, the printer and punch DCTs are removed from the chain of all HASP DCTs and are reinserted directly behind the reader DCT for the corresponding terminal. The device description is converted to internal flags and placed in each of the corresponding DCTs. If the line number is designated in the

description, the line DCT is located, DCTs are chained together, and flags are set to indicate non-signon remote.

The HASP Remote Job Entry Buffer Pool is initialized, and control is passed to the Remote Console Initialization routine or Miscellaneous Initialization routine, as appropriate, by $XCTL.

REMOTE CONSOLE INITIALIZATION - HASPIOVS

The Operator Message Space is allocated, and control blocks are initialized. The Remote Console Processor PCE and a direct-access DCT are connected (the DCT is flagged IN USE). The origin of the first available track in the SYS1.HASPACE data set of the SPOOL1 volume and the base track address for operator message record allocation is set into the MSAMTTR field of the Message Allocation ($MSALLOC) Table in the form: 0TT1 (TT is the first track available for messages). The number of records per track for the mounted SPOOL1 volume is inserted into the MSARPTRK field. If cold start was performed by direct-access initialization, the cylinder map for SPOOL1 is altered to reflect the allocation of sufficient adjacent track groups, starting with the group of the base track. The number of the last group is saved in the checkpoint records for future warm starts. If a warm start was performed by direct-access initialization, a check is made against the checkpoint record to ensure that the space required is within the allocated space. Control is given to the Miscellaneous Initialization routine by $XCTL.

MISCELLANEOUS INITIALIZATION - HASPIOVB

The dispatching priority of HASP is set at 255 using a CHAP macro. The version of HASP is moved to the first eight bytes of the HCT for identification in a storage dump.

The Execution Processor PCEs are numbered in descending sequence from &MAXXEQS to one in the first byte of PCEID.

Several "S INTI.HOSINIT&OSC(n),,,&OSC(n)" commands are issued internally to start initiators for each of &MAXPART HASP logical partitions.

The entry points for the three HASP subtasks (HASPWTR, HASPWTO, HASPACCT) are IDENTIFYed in the HASP load module and ATTACHed.

Control passes to Buffer Building.

BUFFER BUILDING - HASPIOVC

An OS variable GETMAIN is issued to obtain storage for the alternate buffer pool. The actual amount of storage gotten is reduced by the amount of reserved storage (&RESCORE*1024) and is used to determine the number of buffers that may be created in the alternate buffer pool. Extra storage, if any, is released via OS FREEMAIN. The number of

buffers which may be created in the alternate pool is compared against the expression of generation parameters:

&MINBUF - &NUMBUF

where:

&NUMBUF = number of buffers in the main buffer pool

If the alternate buffer pool will not contain at least the number of buffers specified by the expression, a warning is issued.

The origin of the main and alternate buffer pools are examined to determine which has the lower storage address. The pool with the lowest address is created and chained to the $BUFPOOL chain of buffers. The pool remaining is then created and chained to the end of the first.

HASP builds all buffers (overlay areas, TP buffers as described previously for RJE Initialization, and the main and alternate pools described here) such that the IOB, CCWs, and data area are all within the same 4K page. Therefore, only one page must be fixed to do a HASP I/O operation.

JOB QUEUE WARM START - HASPIOVQ

This overlay is called only if the job queue contains jobs from a Warm Start as previously described.

If a job was reading, the operator is informed via WTO and the job queue element is removed from the queue using $QREM.

If a job was executing, the operator is asked via WTOR if HASP should rerun the job. If the response is yes, the job is left in the $XEQ queue. If the response is no, the job is placed in the $OUTPUT queue so that partial results will be printed/punched.

For all jobs which are in the $XEQ queue, track groups occupied by input data sets (indicated by the bit map in the JCT) are marked as allocated in the master track group bit map. For all other jobs, track groups occupied by output data sets (indicated by the bit map in the first IOT) are marked as allocated in the master track group bit map. The number of available track groups in the master map is determined and saved.

All print/punch restart data is moved from the print/punch checkpoint elements to checkpoint JOEs in the Job Output Table (JOT).

Active print/punch device counters are cleared in the characteristics JOEs in the JOT.

All work JOEs are scanned. For those JOEs which were busy printing or punching, the operator is informed via WTO and the busy bit is cleared so that the JOE may be acquired when processing resumes.

ACTIVATION OF NORMAL PROCESSING

The operator is asked to ENTER HASP REQUESTS (if REQ was specified) and the root segment of HASP Initialization is entered via the $RETURN macro.

The root segment returns the PCE to the Command Processor and, if the operator specified REQ in the WTOR, enters the Command Processor. If NOREQ was specified by the operator, all HASP Processors are $POSTed and the Command Processor is entered.

HASP INITIALIZATION SVC

This program is a Type-I SVC routine which resides in the Operating System nucleus and must be link-edited with the nucleus to resolve the external address constants required for HASP processing. It provides the following basic functions:

1. For HASP:

   a. Places HASP in supervisor state.

   b. Returns the address of key symbols in the nucleus which are required for HASP processing.

   c. Guards against recursive entries in order to prohibit multiple copies of HASP from being initiated.

   d. Provides the address of an entry which will cause the SVC routine to be reset for HASP withdrawal and cause the PSW to be reset to its initial value.

   e. Provides entry points for the I/O Supervisor (IOS) to HASP exit coding for the pseudo device exit and the attention exit.

2. For the non-HASP program: to indicate to any other program whether HASP is currently active or not.

Upon entry, register 1 is compared with the EBCDIC characters "HASP". If the register does not compare, a condition code is returned to the user in register 15 as follows:

R15=0 - HASP has not been initiated and is not currently active.

R15≠0 - HASP has been initiated and is currently active.

If Register 1 contains "HASP", further tests are made to validate the caller as HASP. If HASP has previously been invoked or if the caller's protect key is nonzero, the caller is abnormally ended with a S16D abend code.

If the above tests are passed, the caller's register 0 is assumed to be the address of the HASP Vector Table ($HVT). This is stored in the OS/VS2 CVT (in word CVTHJES) to indicate that HASP is active. The left half of the SVC-OLD PSW is saved for the reset routine. The PSW is then modified so that the return to HASP will place HASP in the supervisor state. Register 1 is then loaded with the address of a table of address constants of key nucleus addresses, and return is made through the OS SVC FLIH.

One of the addresses in the Nucleus Address Table is the address of the SVC reset routine. When this routine is entered, it clears CVTHJES to indicate that HASP is no longer active. It then returns to the caller by loading a PSW constructed by concatenating the left half of the original PSW with register 14.

HASP INITIALIZATION SVC

Although not logically part of the SVC logic described above, the SVC routine physically contains two entry points for IOS to HASP exit coding.

IOS transfers to the HASP pseudo device exit (IECHASPE) when a problem program issues EXCP and the UCB is a pseudo device (UCBATI X'01' bit set). The coding in the SVC simply looks at CVTHJES and transfers control to the exit coding in HASP (using the address in $HVT), if HASP is active. If HASP is not active (CVTHJES zero), the exit returns to IOS at +4 from the normal return, causing IOS to abend the caller with a S100 code.

IOS transfers to the HASP attention exit (IECHASPA) when an unexpected device-end interrupt occurs for a locally attached card reader and the UCBATI is X'24'. The SVC contains all coding which is executed directly as part of the exit. If HASP is not active (CVTHJES zero), the exit returns. Otherwise, UCBATI is flagged, the HASP Asynchronous Input/Output Processor and the HASP task are posted, and the exit returns to IOS.

HASP REP ROUTINE

This routine gives the system programmer the capability of applying absolute or relocatable value patches to HASP, at absolute or relocatable storage addresses, as part of the HASP Initialization process.

The REP card format is:

Columns	Contents
1	Any identification - ignored by REP routine
2-5	CSECT name, "REP", or "ABS"
6	Blank
7-12	Address at which to apply patch (6 hex digits); or blank
13-16	Blank
17-blank	Half-word absolute value patches, 4 hex digits each, separated by commas, patch data terminated by first blank; or one full-word (8 hex digit) relocatable value patch, followed by a comma and the name of the resident CSECT which defines the relocatable part of the value.

The above format allows patches to be applied at any absolute storage location (by use of REP or ABS beginning in column 2) or at addresses in HASP CSECTs (resident or overlay), subject to relocation. Relocatable addresses should be taken directly from a HASP assembly listing containing the CSECT to be patched. A blank address field is interpreted as one greater than the last address patched by the previous card, but the card will be used only if columns 2-5 match those of the previous card.

The patches may be absolute values or one relocatable word per card, whose value is relative to any resident HASP CSECT. Relocatable values should be punched as if they were the assembled value of an A-type constant in the CSECT which defines the referenced relocatable symbol.

Use of the term "CSECT name" in the above description means the fifth and following characters of a HASP CSECT name, as taken from the External Symbol Dictionary of a HASP assembly listing.

A deck of one or more REP cards should be terminated by a card having "/*" punched in columns 1-2.

REP cards are read from the card reader, whose address is given by the HASPGEN parameter $REPRDR, immediately after the operator replies to HASP's initial WTOR (if the operator specifies "REP" in the reply options). Each card is listed on the printer, whose address is given by the HASPGEN parameter $REPWTR, unless the operator specifies "NOLIST" in the reply options. All I/O is performed using CPU instructions SIO and

TIO with the CPU disabled for all interruptions. Fixed real storage in LSQA (subpool 255) is used and real CCW data addresses are determined by using the LRA instruction. Cards are read and processed until a card having "/*" in columns 1 and 2 is encountered or until the card reader signals unit exception.

The value or data portion of each card is processed first. If the value is relocatable (indicated by comma in column 25), eight hex digits beginning in column 17 are converted to a binary value. The CSECT name (last four characters beginning in column 26) is located in an internal table of standard resident module names. A value is taken from this table which is the storage address minus any nonzero assembly origin at which the resident module is loaded. This value is added to the value taken from the card.

If the value portion is absolute, groups of four hex digits (separated by commas) beginning in column 17 are converted to binary values until a blank is encountered instead of an expected comma. The values are concatenated to form a single variable length binary value.

The address portion of the card is processed next. If nonblank, six hex digits beginning in column 7 are converted to a binary address. An attempt is made to locate the to-be-patched CSECT name (last four characters beginning in column 2) in the standard resident module name table. If located, the loaded storage address minus any nonzero assembly origin of the resident module is added to the address taken from the card. If the CSECT name is not in the standard resident module name table, the overlay table is searched to determine if the CSECT is an overlay which was made permanently resident. If so, the nonzero assembly origin of the overlay CSECT is subtracted from and the loaded storage address is added to the address taken from the card. In both of the above cases, the patch value, as previously computed, is applied by moving it to the storage address determined by one of the two methods described.

If the CSECT name is not located by either search just described, it is assumed to be an overlay CSECT which is not permanently resident. The name, unrelocated address, and value are saved in a reserved area, to be applied each time the overlay is read from direct-access during HASP operation.

If the address field of the card is blank, the to-be-patched CSECT name is compared with that from the preceding card. If they are not equal, the card is ignored. Otherwise, the card is considered to be a continuation of the preceding card and the patch value is applied at the next higher storage address or is saved, as appropriate.

If no area was reserved to save patch information for application to nonresident overlays (HASPGEN parameter &OREPSIZ=0) or if the capacity of the reserved space is exceeded, the operator message "OVERLAY REPPING ERROR" is issued and HASP operation is abortively terminated.

HASP OVERLAY BUILD PROGRAM

The purpose of this program is to process the object deck output from the 11 HASP assemblies, which make up the HASP load module. Overlay CSECTs are extracted and written (each as a single record) to the sequential overlay data set (ddname OLAYLIB), all references to overlays from resident and overlay routines are resolved, and all resident CSECTs (even if programmed as overlayable) are passed to the OS Linkage Editor in a sequential data set (ddname SYSLIN). Optional control cards are processed to allow changing the status of any overlayable CSECT from actual overlay to permanently resident or vice versa.

On initial entry, the time is sampled. A truncated "time-like" value is saved. This value will be placed in one resident CSECT and one overlay CSECT. During HASP Initialization, if these two values do not match, an error message is produced and HASP terminates.

All data sets are opened and the listing title line is printed. If the control card data set is present (ddname SYSIN), cards are read, printed, and processed until end-of-file is encountered. Each card contains an overlayable CSECT name beginning in column 1, which must begin with "HA$". A SYM table entry is made for each such name. An OCON (index into the Overlay Table, HASPOTAB) is assigned and a priority, if present in column 16 of the card, is remembered. This information is later used to override the normal processing of that CSECT when it is encountered in the object decks. A listing header line is printed at the end of control card processing.

All object decks are processed as a single sequential input data set (ddname SYSOBJ). Only the four object card types ESD, TXT, RLD, and END, as documented in OS/VS2 Loader PLM are processed. All other cards are written directly to SYSLIN. If an object card with a valid ESID number greater than the program's table limits (internal assembly variable &MAXESID) is encountered, the program abends with a U0101 code.

ESD card processing is essentially the construction of two tables from ESD information. The SYM table contains the names of and information about any external names under overlay control (i.e., beginning with "HA$"). It is a global table covering all object decks together. A name is entered when a reference to it or a CSECT definition of it is first encountered, or during control card processing as previously described. An overlay name in an ESD card item is first searched for in the SYM table. If it is found, changes are made to the existing entry. An error message is produced for each duplicate definition of a previously-defined overlay CSECT name, and only the first definition is used. An OCON is assigned to each entry. When a name becomes a defined CSECT, if the fourth character is "0", the overlay routine is actually to be made disk resident, and storage is assigned to load its text.

The ESID table is cleared at the beginning of each object deck and is constructed as ESD items are encountered under control of SYM table contents. It is a table of words ordered by ESID number. TXT card processing and RLD card processing access this table only. It contains relocation values, OCONs, and flags controlling the disposition of text and RLD items.

ESD items (for references to overlays or for definitions of overlay CSECTs which are to be disk resident or are duplicated) are eliminated from an ESD card when processed, before the ESD card is written to SYSLIN. This elimination is done by changing them to type NULL or, if type LD, by physically removing them and compacting the card.

TXT card processing has three possible results. Text belonging to an actual overlay is loaded into storage, subject to relocation according to storage assigned by ESD processing. Text of any overlay CSECT which is a duplicate of one encountered previously is discarded. Text of nonoverlay CSECTs or overlays being made permanently resident is written unaltered to SYSLIN.

RLD card processing concerns individual RLD items as follows. If an item applies to a discarded duplicate overlay CSECT, it is eliminated. If an item references a nonoverlay CSECT, it is left unaltered. An overlay reference item describes a 2-byte Q-type constant assembled in the expansion of the $LINK, $LOAD, $XCTL, and $OCON macros. The reference is resolved by substituting the OCON value assigned to the referenced overlay routine, and the item is eliminated. If the Q-constant exists in an actual overlay routine, the OCON value is simply moved to the proper address of the text already loaded into storage. If the Q-constant exists in a nonoverlay CSECT or overlay being made resident, a new TXT card containing the OCON value is created and written to SYSLIN. Eliminated items are physically removed and the RLD card is compacted before writing to SYSLIN.

END card processing is really end-of-object-deck processing. The card is written unchanged to SYSLIN. The entire SYM table is then scanned for selected processing. Each actual overlay whose text was loaded from the most recent object deck is written to OLAYLIB as a fixed length record of length &OLAYSIZ (internal assembly variable set to 1280 bytes in unmodified HASP). A listing line is printed for each overlay CSECT defined in the most recent deck, with its length and assigned OCON value. Priority and disk address in two forms are printed for actual overlays. An error message is printed if an actual overlay length exceeds &OLAYSIZ.

Processing of multiple object decks continues as above until end-of-file for SYSOBJ is signaled. The entire SYM table is then processed to produce the Overlay Table, which is written to SYSLIN as a new object deck (did not exist in the input) containing a single resident CSECT, HASPOTAB. An error message is printed for any name in the SYM table which is still not defined as a CSECT.

Each entry in HASPOTAB is 4 bytes or, if &DEBUG is set to YES, 12 bytes. The last 4 characters of the CSECT name are included if entries are 12 bytes, to facilitate identification in a storage dump. If a routine is actual overlay (disk resident), the TR (relative form) of disk address and the priority are placed into the table entry for that routine. If an overlay routine was written to SYSLIN by previous processing (to become permanently resident in the HASP load module), a V-type constant is created in its table entry. An appropriate RLD item referencing the CSECT name is created.

When HASPOTAB is complete, an END card for it is written to SYSLIN, all data sets are closed and the program terminates. Completion code 0 is returned normally; the completion code is 4 if duplicate CSECTs were encountered and 8 if any overlays were too long or undefined.

The HASP System/3 Remote Terminal Program is assembled under OS, using Assembler F. The advantages of assembling under OS are: the System/3 program can be assembled as part of a standard HASPGEN or RMTGEN; a System/3 program can be customized to the particular System/3 configuration and HASP System being generated, since Assembler F can handle conditional assembly statements; and macros can be used.

To allow assembly of System/3 code, a set of macros is included as part of the System/3 source code, HRTPSYS3. Most of these macros are designed to generate machine language code for the System/3; a few additional macros, such as $WAIT and $FB, provide for in-line functions and control blocks. The format macros will be discussed first; they are called the machine-language macros.

The machine-language macros consist of a set of macros whose names correspond to the mnemonic System/3 operation codes defined in the publication IBM System/3 Model 10 Components Reference Manual (Order Number GA21-9103) and the extended System/3 assembler mnemonics defined in the publication Disk System Basic Assembler Program Reference Manual (Order Number SC21-7509), with the following exceptions: each mnemonic operation code is prefixed by a dollar sign; no macros are provided for the instruction ZAZ, AZ, and SZ; additional extended mnemonics $NOPB and $NOPJ are provided; and the form and order of the operands is such as to be convenient to Assembler F.

When a machine-language macro refers to a location in core, the operand is coded either "address" or "(displacement,register)". Thus, "$MVC X'1234',(0,REG2),LENGTH" moves LENGTH bytes to core location X'1234' (and succeeding lower-addressed bytes) from the core location pointed to by REG2 (and succeeding lower-addressed bytes).

There are ten forms of machine-language outer macros. These are:

1. The 2-address form, exemplified by "$MVC adr1,adr2,length". The operands "adr1" and "adr2" are as explained above. The operand "length" is assembled as "length-1" unless it is omitted or is literally "*-*" (in which case it is assembled as zero) or the opcode is $MVX, in which case it is assembled as "length". The opcodes $MVC, $ALC, $ED, $ITC, $CLC, and $MVX belong to this form. The extended mnemonics $MZZ, $MZN, $MNZ, and $MNN may be used.

2. The 1-address form exemplified by "$L reg,adr" and including $L, $A, $LA, and $ST.

3. The 1-address form exemplified by "$MVI adr,immediate" and including $MVI, $CLI, $SBN, $SBF, $TBN, and $TBF.

4. The Jump instruction, written as either "$JC adr,cc" or "$Jxxx adr", where $Jxxx is one of the extended mnemonics. In this case, "adr" may not be specified as "(displacement,register)" and must be within a positive displacement of 256 bytes from the last byte of the Jump instruction.

5. The Branch instruction, written as either "$BC adr,cc" or "$Bxxx adr" where $Bxxx is one of the extended mnemonics.

6. The 1-address I/O forms, exemplified by "$LIO da,m,n,adr" and including $LIO, $TIO, and $SNS.

7. The instruction $SIO, written as "$SIO da,m,n,cc".

8. The instruction $APL, written as "$APL da,m,n".

9. The instruction "$HPL, written as $HPL cc" where each "c" is either the actual character to be displayed as a halt code or the character "*", indicating a byte of zeros. For example, "$HPL EJ".

10. The assembler instructions $DC and $DS, where the statement label (if any) is assigned the address of the last byte of the last operand specified.

In addition to the machine-language macros, a $USING and a $DROP macro are provided to enable Assembler F DSECTs to be used more easily. The form of the $USING macro is "$USING expression,register" where "expression" is a 1-to-8-character expression with the location counter reference symbol "*" either not used or used as the first character, and "register" is a 1-to-8-character absolute expression. No more than two different $USINGs (two $USINGs with different arguments "register") may be outstanding at any time. $USING works as follows: from the time the $USING is issued, for any address-type machine-language macro which contains an address specification of "(disp1,reg)", the character string "reg" is compared with the string "register" of each outstanding $USING. If no match is found, the displacement is assembled as YL1(disp1). If a match is found, the displacement is assembled as YL1(disp1-(expression)), where "expression" is taken from the corresponding $USING.

The form of $DROP is "$DROP register" where "register" is a character string that appeared as the second operand of a previous and outstanding $USING. The form "$DROP register,register" is also allowable.

The assembly listing generated by Assembler F contains the macro-expansion for each macro used, to provide a printed copy of the generated text of each machine instruction and the address at which it will be loaded in System/3 storage. The expansion of each of the machine-instruction macros is typically contained in one print line, and the text of the generated instruction is always contained in hexadecimal on one print line.

The object deck produced by Assembler F is used as input to the translation program SYS3CNVT, called automatically by RMTGEN. SYS3CNVT reads the object deck via either ddname SYSLIN, or ddname SYSGO if SYSLIN is absent. First, SYS3CNVT punches on SYSPUNCH a System/3 1-card loader. Then it reads from SYSLIN or SYSGO, ignoring all but TXT cards and the END card. For each TXT card, SYS3CNVT creates one System/3 96-column load-mode card image, suitable for reading by the System/3 1-card

loader. Each such 96-column card image contains 64 bytes of information as follows:

1. Bytes 1-5 contain a System/3 $MVC instruction of the form "$MVC load-adr,(column-number,1), length-1" length-1, where load-adr is the absolute load address of the rightmost byte of text on the corresponding 80-column Assembler F object deck TXT card, column-number is the number minus one of the 96-card column in which appear the low-order six bits of the rightmost byte of text, the digit "1" refers to the System/3's register 1, and length is the number of bytes of text on the card.

2. Bytes 6-61 contain a maximum of 56 bytes of text, starting in colume 6.

3. Bytes 62-64 contain a three digit card sequence number.

When the object deck's END card is detected, or when a TXT card appears that was generated by the $END macro (whose optional keyword operand START= specifies the starting execution address of a segment of text), a 96-column load-mode card image is constructed whose 64 bytes are as follows:

1. Bytes 1-4 contain a System/3 $B instruction of the form "$B address" where address is either the first byte of the text segment just loaded (if the $END macro does not specify START=, or if the END card of the assembly has no operand) or the address specified in the START= parameter of the $END macro or the operand field of the END card.

2. Bytes 62-64 contain a 3-digit card sequence number.

After the object deck's END card has been processed, SYS3CNVT creates a 96-column card image of which columns 2-4 are "EOR" (this is the rep terminator card, END-OF-REPS) and columns 62-64 contain a 3-digit card sequence number.

Certain of these 96-column card images contain descriptive information in bytes 33-64: these cards make up the 1-card loader, which is captioned "FIRST CARD", the card created from a $END macro, which is captioned "PSEUDO-END" and the card created from an END card, which is captioned "LAST CARD".

After it has created each 96-column card image (including that for the 1-card loader), SYS3CNVT breaks the image in half and punches two 80-column cards from it. Each 80-column card punched by SYS3CNVT contains the following fields:

1. Columns 1-2 are blank

2. Columns 3-50 contain the first (if column 80 is odd) or the last (if column 80 is even) 48 bytes of a System/3 card image

3. Columns 51-72 are blank

4. Column 73 contains the punch combination for X'80', an indicator to any System/3 Remote Terminal Program generated with &S396COL=1 that two 80-column cards are to be combined and punched as one 96-column card (the System/3 Starter System is generated with &S396COL=1)

5. Columns 74-80 contain the remote terminal identifier and card sequence number, in the form "Rmmnnnn", where nnnn is 0001 on the first card punched.

The punched output of SYS3CNVT may be routed directly to a System/3 which is running the Starter System or other suitable System/3 Remote Terminal Program; the resulting 96-column punched deck of cards is immediately ready for loading into a System/3 of the proper configuration. Alternatively, SYS3CNVT's punch output may be punched on 80-column cards for later transmittal to a System/3. Each 80-column card is suitable for data transmission in either transparent or nontransparent mode.

The HASP System/3 Work Station program consists of processors, interrupt routines, and system subroutines. There is a processor for each logical function to be performed by the program, each processor is controlled by a Function Block (somewhat analogous to a TCB in OS). Interrupt routines are provided for those devices (BSCA, 5471, and 5475) which are capable of interrupting the CPU; other devices are operated by processors. For example, the MFCU Processor operates a hopper of the 5424 MFCU; it becomes associated with either a logical reader processor or a logical punch processor, depending on the state of the hopper.

The various routines of the System/3 Remote Terminal Program are described in the order in which they appear in the listing.

IHEREP - HASP ENVIRONMENTAL RECORDING AND ERROR PROCESSOR

IHEREP prints at program load time the error statistics gathered from the previous running of the System/3 Remote Terminal Program. IHEREP is then overlaid and the Remote Terminal Program continues to load.

First, IHEREP loads the 5203 forms length register and selects the correct print chain image according to the printer's status information. Then it checks the log area for validity. If the log area is valid, the characters 'HASP' will appear immediately before the log area. If these characters do not appear, IHEREP prints the message:

HEREP COUNTERS HAVE BEEN ALTERED

and branches to zero to cause program loading to resume.

If the log area is intact, it contains eight 2-byte counters for each status byte which can contain unit check information for a device. IHEREP prints a title line and then, for each status byte, a subtitle

line and as many as eight detail lines. A subtitle line contains device description and status byte number. A detail line contains status bit description, bit number, and count of bit occurrences in decimal.

Control of IHEREP resides in the table of subtitles and detail descriptors, and control of the 2-byte bit counters is by a bit string (starting at symbol IHBIT) containing one-bits for the counters to which correspond detail descriptors. The table of subtitles and detail descriptors is made up of $IHMSG macros; if the first operand of this macro is 'T', the macro defines a subtitle, and if the first operand is an integer between 0 and 7, it specifies a detail descriptor for the bit whose bit number is the first operand. The table entries are used in order, and a byte of zero defines the end of the table.

When the HASP Environmental Recording and Error Printout is complete, the counters are zeroed out, and IHEREP branches to zero to continue program loading.

$COM - COMMUTATOR

The Commutator gives control in turn to the various processors which comprise the System/3.

If the Event Wait Field (FBEWF) of an FB has zeros in the bit positions defined by EWFALL, the function is said to be dispatchable. $COM loads register 1 from field FBREG1 of the FB (register 2 points to the FB) and gives control to the associated processor by loading the Instruction Address Register (IAR) from field FBENT.

When the processor has completed its work, it returns to the commutator with register 2 pointing to its FB. It may return to $COMRET, where $COM will save both the Address Recall Register (ARR) as the processor's next entry point and the value of register 1; $COMRETA, where $COM will save the value of register 1, or $COMRETB, where $COM will assume that both the value FBENT and the value FBREG1 are correct.

Then $COM chains to the next FB (or starts again with the first FB if the chain field FBNEXT is zero) and repeats the above process.

$MFCU - 5424 MFCU PROCESSOR

$MFCU operates under two FBs and two Hopper Control Areas (HCAs) - one for each MFCU hopper. The routine contains four levels of subroutines.

$MFCU begins by calling first-level subroutine HREAD to read a card. HREAD sets up a read $SIO instruction from information in the HCA and calls second-level subroutine HEXCP. HEXCP calls fourth-level subroutine HTIO, which returns condition code equal if the hopper described by the HCA is ready and condition code unequal if it is not. If condition code unequal is returned, HEXCP returns to the commutator; it will regain control again at the call to HTIO.

If the hopper is ready, HEXCP calls third-level subroutine HSIO to perform I/O on the hopper. HSIO first checks for various exceptional conditions. If error recovery is in progress (for the other hopper), HSIO returns immediately with condition code unequal. It returns similarly if the MFCU is busy reading, printing, or punching. If error recovery is not in progress and the MFCU is not busy, HSIO tests the "hurry" switch (which is set if one hopper is active and the other hopper becomes ready with a read $SIO pending for it). If the hurry switch is set and the current $SIO is not a read-only $SIO, HSIO returns condition code false.

If all the above tests are passed, HSIO checks the stacker request associated with the current $SIO. If the stacker request is different from that for the previous $SIO, the feed path is checked to make sure it is clear. If the feed path is not clear, HSIO returns condition false; in addition, if the $SIO is read-only, it sets the "hurry" switch. But if the feed path is clear, HSIO resets the hurry switch, sets the new stacker number, and proceeds as if the stacker request for the current $SIO were the same as that for the previous $SIO.

If no stacker change is indicated, HSIO moves the current $SIO to an in-line position from the HCA and examines it. If the $SIO indicates print (interpreting), HSIO attempts to select one of two print buffers into which to move the punch information for the $SIO. If unsuccessful, HSIO returns condition code unequal. But if one of the print buffers is free (as indicated by the MFCU print-buffer-busy status bits) HSIO copies the punch data into the print buffer and modifies the $SIO instruction to indicate the print buffer being used. Then, or if the $SIO is read-only, HSIO loads the MFCU's read and punch data address registers. After a call to HTIO to ensure that the hopper is still ready, HSIO issues the $SIO instruction, sets condition code equal, and returns to its caller, HEXCP.

HEXCP examines the condition code returned to it. If the condition code is unequal, HEXCP nonprocess exits, exactly as it did for HTIO above. But if the condition code is equal, HEXCP nonprocess exits to be entered again at a $STIO which continues to nonprocess exit until the MFCU ceases being busy; then HEXCP calls third-level subroutine HSNS to determine the completion of the I/O operation.

HNSN calls HTIO to see if a unit check condition exists. If that is the case, HSNS reads the MFCU status bytes. If all status bits in the error status byte are off (or if no unit check condition existed) HSNS returns condition code equal; if only the no-op status bit is on, HSNS returns condition code unequal.

If other error status bits are on, HSNS calls system subroutines $MSG and $LOG to add a message to the error trace table and to count the error bits for HEREP, respectively. Then HSNS checks the error bits further. If the only error bits on are punch invalid or print check, HSNS returns condition code equal; these are regarded as user data errors (punch invalid) or trivial errors (print check).

But if other error bits are on, HSNS sets the error-recovery-in-progress flag in HSIO (to prevent other $SIO instructions from resetting the error bits) and nonprocess exits until a SNS instruction shows that all error bits (except no-op) have been reset by the operator (who must do a nonprocess run-out on the MFCU). Then HSNS returns condition code unequal.

HEXCP returns to its caller (which was HREAD in this case) the condition code it received from HSNS.

HREAD examines the condition code returned to it by HEXCP. If unequal was returned, HREAD again calls HEXCP; otherwise first-level subroutine HREAD returns control to mainline $MFCU (in this case, at its second instruction).

Having read the first card from its hopper, $MFCU now tests that card for blanks, via first-level subroutine HBLANK. If the card is blank, the hopper is assumed to contain blank cards to be punched. Otherwise, the hopper is assumed to contain a job stream and the MFCU awaiting-read routine HAR attempts to associate the hopper with a free logical reader FB, using subroutine HGET. HGET returns condition code equal if it succeeds (it also posts the logical reader's FB for UNIT), and condition code unequal if the hopper becomes not ready (and therefore dormant rather than awaiting-read); otherwise, HGET nonprocess exits until one of the above two conditions happens.

If HGET returns condition code equal, the MFCU reading routine, HRD, signals to the now-associated logical reader that the read buffer for the associated hopper is busy; then HRD nonprocess exits until the logical reader frees the read buffer. When the read buffer is free, HRD checks the EOF flag, set by the logical reader when it encounters a /*EOF control card. If the EOF flag is on, HRD makes the hopper dormant by branching to the first instruction of $MFCU; otherwise HRD calls first-level subroutine HREAD as above to read the next card and, on return, again sets the read buffer busy.

If on the other hand $MFCU finds a blank card in a dormant hopper it gives control to HAP, the awaiting-punch routine, which tries to find (via HGET) a logical punch FB of which HASP has requested permission to send a punch stream. Having found such a logical punch, HAP gives control to HPU, the MFCU punch routine.

HPU nonprocess exits until the assoicated logical punch processor sets either the EOF flag or the punch-buffer-busy flag in the flag byte of its hopper control area. If the EOF flag is set, HPU makes the hopper dormant.

But if the punch-buffer-busy flag is set, HPU punches and prints a card and reads the next card (to ensure that only blank cards are punched). HPU sets up a read-punch-print $SIO and calls second-level subroutine HEXCP. If HEXCP returns condition code unequal and the MFCU status indicates any of the error no-op, punch check, hopper check, or feed check, the punch buffer is not marked free; otherwise, it is marked free and set to blanks. The MFCU status is checked again; if neither read

check nor no-op is indicated, the card is examined to determine if it is completely blank. Otherwise, or if the card now in the wait station is not blank, another card is read (via subroutine HREAD). When a blank card has been read successfully, HPU again checks for punch-buffer-busy as above.

$1442 - 1442 CARD READER - PUNCH PROCESSOR

The $1442 processor is assembled if RMTGEN parameter &S31442 has been set to 1. Its logic is similar to that of $MFCU but simpler, since only one hopper need be controlled. $1442 uses some of the subroutines of $MFCU; for this reason, and since its interface to the logical reader and logical punch is the same, the 1442 hopper control area is similar to (but not identical with) the HCAs of the MFCU.

$1442 starts by reading a card from the 1442 via entry point GSIORD of subroutine GSIO. If the card is blank, GAP (awaiting-punch) calls HGET just as does HAP in $MFCU; if the card is nonblank, GAR (awaiting-read) calls HGET just as does HAR in $MFCU.

When a logical reader or logical punch has been associated with the 1442, GRD or GPU gains control and proceeds with I/O as indicated by the read-buffer-busy and punch-buffer-busy flags. In addition to recognizing the EOF flag set by the logical reader, GRD also recognizes the last-card flag, recognized by the logical reader.

Subroutine GSIO performs I/O on the 1442. Entry point GSIORU sets a feed command in the $SIO and branches to common code. Entry point GSIORD sets a read-EBCDIC command in the $SIO and loads the data address register; it branches to common code. Entry point GSIOPU sets up a punch-and-feed command, loads the data address register and the punch count register, and falls through to common code.

GSIO's common code nonprocess exits on a $TIO until the hopper is ready. Then it issues the constructed $SIO and nonprocess exits until the 1442 is not busy. If entry was from GSIORU, GSIO returns condition code equal; otherwise it tests for unit check (via subroutine HTIO) and reads the 1442 status bytes. If no unit check occurred, GSIO returns condition code equal.

But if the 1442 had a unit check or otherwise became not ready, GSIO uses subroutines $MSG and $LOG to add a message to the error trace table and count the error bits for HEREP, respectively; then it checks the status bytes. If no error bit is on, GSIO returns condition code equal; otherwise GSIO returns condition code unequal.

$5203 - 5203 PRINTER PROCESSOR

The 5203 Printer Processor nonprocess exits until another processor has marked the printer data area busy. Then it completes the Q-byte and CC-byte of a $SIO instruction from an SRCB furnished it by either $PRINTER or $CONP. After a $TIO shows that the 5203 is ready, $5203 loads the

printer image address register and the printer data address register and issues the $SIO. $5203 then nonprocess exits until the printer is not busy.

When the printer operation has ended, $5203 checks for errors. If any of the error incrementer failure check, hammer echo check, or any hammer on check has occurred, $5203 attempts to reprint the line. Otherwise, it clears the print line to blanks, shows the print buffer free, and again nonprocess exits until a processor sets the print buffer busy.

Additionally, whenever a unit check occurs, $5203 calls subroutines $MSG and $LOG to produce an error message and to count the one-bits in the printer status bytes.

$READER - LOGICAL READER PROCESSOR

$READER waits for one of the physical reader routines to post it for UNIT. When posted, it sends to HASP a request-permission control sequence (via subroutine $REQ) and waits to be posted for PERM by $BSCA when the system receives from HASP the appropriate permission-granted sequence.

When it has received permission, $READER nonprocess exits unless the read-buffer-busy flag is on, indicating a card is ready to be processed. Then it examines the card. If the card's columns 1-5 are "/*EOF", $READER sends to HASP an end-of-file control sequence (via subroutine $LEOF), which is merely a zero-length record. It then waits again for UNIT, and continues as above when posted. The same end-of-file processing occurs if the reader is a 1442 and the last-card flag was set by the 1442 physical reader routine. 1442 code is absent unless &S31442=1.

If there is no end-of-file indication, $READER processes the card further. If object deck processing was not specified at RMTGEN time, $READER transmits the first 80 columns of the card to HASP by calling subroutine $CMPR. On return, $READER resets the read-buffer-busy flag of the appropriate hopper control area and nonprocess exits until the read-buffer-busy flag is again set by the physical reader routine. Then it continues as above.

However, if object deck processing was indicated at RMTGEN time by the specification &S30BJDK=1 and if the physical reader device is a 5424, $READER first checks column 81 of the 96-column card image for the character "1". If the comparison is unequal, the card is the first card of a 2-card hexadecimal image of a full-EBCDIC 80-column card. In this case, $READER compresses the first 80 columns of the card into the first 40 bytes of the same device's punch buffer, shows the read buffer free, and nonprocess exits until the read buffer is again busy. Then it checks the new card image for a "2" in column 81. If column 81 does not contain a "2", $READER treats the newly-read card as a normal card, and the previous card is lost. If the new card contains a "2" in column 81, $READER compresses its first 80 columns to the second 40 bytes of the same device's punch buffer and transmits the constructed card image to

HASP, using subroutine $CMPR. Then it resets the read-buffer-busy bit and nonprocess exits as above.

Subroutine RDSQUEZE performs the above-mentioned compression. It creates a single sink byte from a pair of source bytes each of which is assumed, without validity-checking, to contain the EBCDIC representation of one of the 16 hexadecimal characters. EBCDIC representation of one of the 16 hexadecimal characters. For example, it would compress the byte pair "F0C6" to the byte "0F".

$PRINTER - LOGICAL PRINTER PROCESSOR

$PRINTER waits for HASP to send a request-permission control sequence. When $BSCA finds such a sequence, it posts $PRINTER for permission. $PRINTER then checks the printer availability flag. It nonprocess exits until this flag becomes zero; then it sets this same flag to show that the printer is in use. It sends a permission-granted control record to HASP (via subroutine $PERM) and then, if the print buffer is free, calls subroutine $DCOM to request a print line be decompressed into the print buffer.

On return from $DCOM, $PRINTER recognizes two or three conditions: normal return, end-of-file return, and (optionally) forms mount message.

For the forms mount message case, the SRCB (carriage-control byte, in the case of print records) will be X'8E'. $PRINTER makes the carriage control byte a print-and-space-three, shows the print buffer busy, and nonprocess exits until the print buffer becomes free; then it sets a carriage-control byte of space-three-immediate (so that the forms mount message will be visible on the printer without operator intervention) and continues as in the normal case. This code is assembled only if &S35471=0.

For the normal-return case, $PRINTER moves the SRCB returned by $DCOM to the printer control area as the carriage control byte, sets the printer-buffer-busy bit, and nonprocess exits until the print-buffer-busy bit is off. Then it again calls $DCOM for the next print line.

For the end-of-file case, $SPRINTER resets the printer availability flag and checks to see if HASP had again sent a request-permission. If so, $PRINTER again sets the printer availability flag, sends to HASP permission-granted (via subroutine $PERM) and continues as above. Otherwise, $PRINTER waits for HASP to send request-permission.

$PUNCH - LOGICAL PUNCH PROCESSOR

$PUNCH waits for HASP to send a request-permission control sequence. When $BSCA finds such a sequence, it posts $PUNCH for PERM, whereupon $PUNCH waits for UNIT. When posted for UNIT by a physical device routine, $PUNCH sends a permission-granted control record to HASP (via subroutine $SPERM) and nonprocess exits until the appropriate punch

buffer is free. Then it calls subroutine $DCOM to decompress a card image into the punch buffer.

If $DCOM returned a card image (rather than end-of-file) the image is processed in various ways, depending upon the type of the punch device and options selected at RMTGEN time. If the punch is a 1442, $PUNCH calculates the number of bytes to punch, subtracts it from 128, places the difference in the 1442 hopper control area, and shows the punch buffer busy. It then nonprocess exits, as above, until the punch buffer becomes free.

If the device is a 5424, $PUNCH first checks column 1 of the card image.

If column 1 is X'6A', the card image is assumed to be a HASP job separator card. $PUNCH extracts the job number from columns 52, 62 and 75, ignores the rest of the image, and punches a card of which columns 1-32 are:

```
********** JOB nnn **********
```

It causes this card to be punched as usual, that is, by marking the punch buffer busy; then it nonprocess exits until the punch buffer becomes free.

If the device is a 5424 and RMTGEN specified &S396COL=1, $PUNCH checks column 73 of the card image. If that column is X'80', &PUNCH checks column 80. If column 80 is odd, $PUNCH saves in a work area in its Function Block the 48 columns starting at column 3 and again calls $DCOM to get the next card, as above. If column 80 is even, $PUNCH moves columns 3-50 of the card image to columns 49-96, moves the first 48 bytes from its work area to columns 1-48, and causes the card to be punched.

If the device is a 5424 and RMTGEN specified $S30BJDK=1, $PUNCH checks column 1. If that column is X'02' $PUNCH saves the rightmost 40 columns of the 80-column card image in its work area and expands the leftmost 40 columns to 80 columns by substituting for each byte two EBCDIC characters; for example, X'02' becomes C'02'. It sets the character "1" in column 81 and causes the card to be punched. $PUNCH then repeats this process for the saved 40 columns, sets the character "2" in column 81, and causes the card to be punched.

If none of the above situations apply, $PUNCH merely marks the punch buffer busy, nonprocess exits until it becomes free again, and then calls $DCOM to get the next card.

$DCOM may return an end-of-file indication rather than a card image. $PUNCH sets the end-of-file flag in the hopper control area and checks for a subsequent request-permission from HASP. If HASP has requested permission again, $PUNCH waits again for UNIT, as above; otherwise, $PUNCH waits for PERM, as above.

5471 CONSOLE INTERRUPT ROUTINE

CINT, the 5471 Console Interrupt routine, gains control upon an interrupt from either the 5471 Printer or the 5471 Keyboard. A keyboard interrupt may occur due to the END key, the RETURN key, the CANCEL key, the REQUEST key, or a DATA key. A printer interrupt may occur either after completion of printing a character or after a carriage return.

At an END key interrupt CINT starts a carriage return, posts the Console Processor, and exits by starting the keyboard. If a request is pending, the Start I/O instruction sets the request light on and disables interrupts from all keys; otherwise it sets both lights off and enables interrupts from the request key.

A RETURN key interrupt causes the same functions as an END key interrupt.

A CANCEL key interrupt causes CINT to print an asterisk and set a flag which will cause a carriage return at the next printer interrupt. CINT then resets the buffer pointer to point to the first byte of the buffer and exits by issuing a SIO which leaves the same lights on and interrupts enabled as before the interrupt.

For a DATA Key interrupt, CINT saves the keyed character in the buffer byte pointed to by the buffer pointer; then it increments the buffer pointer by one. It issues a SIO to the printer so that the keyed character will be printed. If the buffer pointer now falls outside the buffer, CINT turns on the carriage-return request bit and performs all the functions of the END key except for issuing a carriage return. Otherwise, it exits by issuing to the keyboard a SIO which leaves the same lights on and interrupts enabled as before the interrupt.

On a printer interrupt due to end of either printing or carriage return, CINT tests the carriage return request bit. If that bit is on, CINT resets it and exits by issuing a SIO for carriage return.

If there is no carriage return request pending, CINT tests the output-in-process bit. If output is not in process, CINT exits by disabling printer interrupts. But if output is in process, CINT checks whether the final output character has been printed. If so, it resets the output in-process flag, posts the Console Processor, and exits by restarting a carriage return. If not, it selects and loads the next character to print and exits by issuing a SIO to print that character.

Whenever CINT posts the Console Processor, it also turns on the action-required flag, CFACT. This flag is tested and reset by the Console Processor.

5471 CONSOLE PROCESSOR

The 5471 Console Processor, $CON, nonprocess exits until posted; then it checks to find what caused it to be posted.

If input is complete, $CON replaces in the MULTI-LEAVING buffer pool the buffer it stole when it acknowledged the request key. Then it sends the

operator command to HASP by calling subroutine $CMPR, unless the input length is zero. In any case, it continues by checking for request-pending.

If a keyboard request is pending, $CON first steals a buffer from the MULTI-LEAVING buffer pool, to avoid a potential buffer lockout problem. If no buffers are available, it leaves the request pending and checks for queued buffers containing messages to print on the 5471 Printer. But if the MULTI-LEAVING buffer steal was successful $CON resets the 5471 buffer pointer, resets the action-required and request-pending flags, sets the input-in-process flag, and issues a SIO which turns on the proceed light and enables all keyboard interrupts. Then it nonprocess exits until posted.

If $CON was not posted for the above reasons, it investigates output possibilities. If either input or output is in process, it cannot start output; it again nonprocess exits until posted. But if neither input nor output is in process, and if there is no end-of-forms indication from the 5471, $CON checks for output. First it checks the error message table, a circular table, to see if any error messages are outstanding. If so, it expands a 4-byte coded error message to the equivalent 8-character hexadecimal representation in the 5471 buffer, sets the output-in-process flag, and issues a SIO to start printing the first character; then it nonprocess exits until posted, while CINT prints the remaining characters.

If no error messages are outstanding, $CON checks for messages from HASP. If there are some, $CON calls subroutine $DCOM to decompress a message. In order not to be forced into a wait condition on subsequent calls to $DCOM, $CON then checks whether the MULTI-LEAVING buffer from which the message was decompressed contains more messages; if not, $CON frees it by calling subroutine $FREEBUF. Then $CON initiates printing of the message by setting the output-in-process flag and issuing a SIO to print the message's first character. Then $CON nonprocess exits until posted.

5475 CONSOLE INTERRUPT ROUTINE

Upon an interrupt from the 5475 Data Entry Keyboard, the 5475 Console Interrupt Routine (CINT) checks the cause of the interrupt. An interrupt may be caused by a DATA key, the FIELD-ERASE FUNCTION key, the RELEASE FUNCTION key, the error-reset function key, any other function key or switch, or the MULTIPUNCH key. A MULTIPUNCH key interrupt is treated as an error and requires the operator to depress the ERROR-RESET key; all function keys and switches other than those mentioned are treated as no-operation keys.

A DATA key interrupt causes CINT to place the keyed character in the 5475 buffer. CINT then increments the buffer pointer by one; if the buffer pointer now points outside the buffer, CINT performs the RELEASE key function. Otherwise CINT adds one to the column indicated and exits. The exit process consists of issuing a LIO for the column indicators and a SIO for the keyboard.

An interrupt from the RELEASE key causes CINT to post the 5475 Console Processor for work, set the SIO in CINT to disable the keyboard, and exit.

Any of several error situations causes CINT to turn on the error light. It does this by setting its SIO to X'23' which also locks all data keys. When an interrupt other than from the ERROR-RESET key occurs and the error light is on, CINT exits without further processing. But if the interrupt was from the ERROR-RESET key, CINT resets the SIO to its normal value of X'4F' and exits. Conditions which cause the error light to come on are a multipunch interrupt indication, no interrupt indication, or two or more of: the Interrupt Conditions Data key, the FUNCTION key, and the MULTIPUNCH key.

5475 INPUT CONSOLE PROCESSOR

When posted for WORK by CINT, the 5475 Input Console Processor ($CON) sends the operator command to HASP by calling subroutine $CMPR, unless the input length is zero. In any case, it resets the column indicator save area to "01", resets the 5475 buffer pointer, and sets to X'4F' the SIO in CINT. Then $CON turns off the column indicator display (to avoid burning out the lights), issues an SIO to unlock the keyboard and enable interrupts, and again waits for WORK.

$CONP - 5203 OUTPUT CONSOLE PROCESSOR

When posted for WORK, $CONP checks the printer-availability flag. This flag is on if $PRINTER is currently printing a job. If the flag is on and RMTGEN specified &PRTCONS=2, $CONP frees all MULTI-LEAVING buffers currently queued on its Function Block (using subroutine $FREEBUF) and again waits for WORK. But if RMTGEN specified &PRTCONS=1, $CONP checks to see if it should force messages to be printed on the 5203. It does this by comparing the number of MULTI-LEAVING buffers currently queued on its Function Block with a maximum number. If the comparison is low, it nonprocess exits until either the comparison is not low or the printer-availability flag is off; if the comparison is not low, it performs a page eject before starting to print messages.

To print messages, $CONP first prevents the logical printer routine $PRINTER from using the 5203 simultaneously; to prevent this, it sets the UNIT wait bit in $PRINTER's Function Block. Then $CONP attempts to find an outstanding 4-byte coded error message; if it finds one it expands the message to eight bytes and causes it to be printed.

If no error messages are outstanding, $CONP checks for messages from HASP. If there are some, it calls $DCOM to decompress a message. In order not to be forced into a wait condition on subsequent calls to $DCOM, $CONP then checks whether the MULTI-LEAVING buffer from which the message was decompressed contains more messages; if not, it calls $FREEBUF to free the buffer. Then $CONP causes the message to be printed, by marking the print buffer busy and nonprocess exiting until it again becomes free. All messages printed by $CONP are single-spaced.

Finally, if no messages remain to be printed, $CONP examines the printer-available bit to determine if it interrupted a job to print messages. If so, $CONP does a page eject. In any case, $CONP resets the UNIT wait bit to unlock $PRINTER and waits for work again.

BSCINT _ BSCA INTERRUPT ROUTINE

The BSCA Interrupt Routine, BSCINT, processes all interrupts and performs all error recovery for the Binary Synchronous Communications Adapter. Processing is always initiated by one of three types of op-end interrupts: end-of-transmit, end-of-receive, and 2-second timeout.

For an end-of-transmit interrupt, BSCINT gains control at BSXOPE. If no hardware errors have occurred, it starts a receive operation; otherwise, it uses subroutine BIDISCON to recover from a possible disconnect and, on return, attempts to retransmit.

For an end-of-receive interrupt, a great deal more is done. After having computed the number of received bytes, BIRCV checks for hardware errors; if any occurred, it uses subroutine BIDISCON and then transmits a negative acknowledgment (NAK) to HASP.

The section of code responsible for transmitting a NAK first checks whether the wait-a-bit (WAB) sequence had been transmitted most recently; if so, it transmits the WAB sequence again rather than a NAK. If not, it determines if more than five bytes had been received. Since the buffer used for a receive is the same as that used for a transmit, the receive operation may have overlaid some or all of the transmitted data; since the starting or ending sequence was incorrect or a hardware error occurred, BSCINT has not yet received a positive acknowledgment for the transmitted data. To alleviate this problem, the first five bytes of the transmit data were saved before the buffer was transmitted. If the receive operation overlaid more than these bytes, the buffer cannot again be transmitted; the first two saved bytes are replaced with a DLE-ACK0 and the transmit ending address is set to the starting address plus two. Then the routine transmits a NAK to HASP.

If the received starting sequence was a NAK, the interrupt routine sets up an error message of 02000000 (NAK received), refreshes the first five bytes of the buffer and the transmit ending address, and retransmits the buffer to HASP.

If the received sequence was DLE-ACKO, BSCINT sets flags to show $BSCA that a transmit/receive operation has completed; then it exits by starting a 2-second timeout. If the 2-second timeout completes before $BSCA has cancelled it, BSCINT sets the 2-second-timeout-complete flag and exits by disabling BSCA interrupts.

If the second byte of the received starting sequence was STX and the ending byte was ETB, BSCINT validates the Block Control Byte (a HASP control byte which contains a modulo-16 received-block count) and saves the 2-byte HASP Function Control Sequence. If the BCB is as expected, interrupt processing concludes as for DLE-ACK0. Otherwise, the STX is

changed to X'FF' as a signal to $BSCA to throw the buffer away and the difference between the received BCB and the expected BCB is examined. If the modulo-16 difference is -2 or -1, BSCINT tolerates the error; otherwise it sets up an error message of 02rree00 to display the received and expected BCBs and it builds and transmits to HASP a BCB-error control sequence.

$BSCA - COMMUNICATIONS ADAPTER PROCESSOR

$BSCA nonprocess exits until BSCINT posts it with an indication that either an error message awaits synchronous processing, a receive operation has completed without error, or a 2-second timeout has occurred.

If an error message was produced by BSCINT, it must be placed in the circular Error Message Trace Table by a synchronous processor rather than an interrupt routine, since the $MSG subroutine is not reentrant. $BSCA calls the $MSG subroutine to add the error message to the area table.

If a receive operation has ended without error, $BSCA processes the received buffer, which is always the first buffer on $BSCA's buffer chain. If the buffer does not contain text, $BSCA frees it immediately. Otherwise $BSCA inspects the buffer's first RCB (or first SRCB if the RCB indicates a MULTI-LEAVING control record). If the RCB is zero (typical when HASP sends wait-a-bit) $BSCA frees the buffer. Otherwise, $BSCA compares the RCB (or SRCB) with the field FBRCB in all FBs eligible to receive buffers; if there is no match, it frees the buffer. But if a match is found, $BSCA again determines if the first record in the buffer is a control record. If so, it posts the subject FB for PERM and resets its POST bit to indicate a possible early post (the POST bit is turned on by subroutine $PERM); then it frees the buffer. But if the buffer contains data records, $BSCA dequeues the buffer from its own FB and queues it onto the subject FB, in the process reducing its own buffer count by one, increasing that of the subject FB by one, and, if the subject FB's buffer count (FBBCT) becomes equal to or greater than the subject FB's maximum buffer count (FBBMX), resetting the appropriate bit in the master Function Control Sequence $FCS by using FBFCS.

If $BSCA turned off an FCS bit, it turns on flag BFCSOFF. Whether or not $BSCA turned off an FCS bit, it inspects the subject FB's flags; if flag BFCON is on in FBFLG, $BSCA resets that flag and its own BFCSON flag. Flag BFCSON expedites transmission of a response, and flag BFCSOFF delays transmission. The effect of the above manipulation is to avoid an unnecessary line turnaround when a printer or punch is temporarily at its buffer limit.

Having processed the received buffer, or if a 2-second timeout occurred, $BSCA determines what and when it is to transmit. It transmits a response immediately under any of the following conditions:

1. Wait-a-bit was received from HASP.

2. A text buffer is ready to send.

3. Flag BFCSON is set and there is a free buffer.

4. Text was received from HASP, flag BFCSOFF is not set, and there is a free buffer.

5. Two seconds have passed since end-of-receive.

The response transmitted is one of the following:

1. Text if a text buffer is ready to send

2. A Function Control Sequence if there is a free buffer and the FCS has changed

3. DLE-ACK0, if there is a free buffer and the FCS has not changed from when it was last transmitted

4. Wait-a-bit (including FCS) if there are no free buffers.

To get a free buffer $BSCA uses subroutine BSGBUF, which queues the buffer on $BSCA's buffer chain (FBBUF) in last-in, first-out fashion and increments its buffer count (FBBCT) by one. Additionally, a part of BSGBUF sets up the transmit starting address, receive starting address, and receive ending address, and may be called separately from BSGBUF.

$CMDSCAN - LOCAL COMMAND SUBROUTINE

If RMTGEN parameter &S3CMDS is set to 1, code is assembled to provide a local command facility. Code appears in four places:

1. $CMDSCAN, to process the commands

2. $CVB, used by $CMDSCAN to convert decimal command operands to binary

3. $MFCU, to allow $CMDSCAN to check nonblank cards from dormant hoppers ($CMDSCAN returns condition code equal if a card contained a command; the hopper remains dormant)

4. $1442, with the same functions as $MFCU.

$CMDSCAN receives a pointer to a card in index register 1. It examines the card for a valid command and branches to the proper command routine, or returns to its caller with condition code not-equal.

Each command routine processes the command's operands as necessary, and exits to one of three labels:

1. CMDEND (normal end) to print 'CODE0000'

2. CMDSYN (syntax error) to print 'CODE0001'

3. CMDOPD (operand error) to print 'COD␣0002'.

A command routine may use the $CVB subroutine to convert an operand from decimal to binary. Index register 1 must point to the decimal operand's high-order byte. If this byte is not numeric, $CVB will branch to CMDSYN: otherwise, on return from $CVB, the binary result will be right-justified in bytes $CVBANS and $CVBANS-1, and index register 1 will point one byte past the low-order digit of the decimal operand.

$LEOF, $PERM, $REQ - CONTROL SEQUENCE SUBROUTINES

These subroutines transmit to HASP certain control sequences required for proper operation of HASP MULTI-LEAVING Remote Job Entry: logical end-of-file, permission-granted, and request-permission.

$LEOF sends the sequence RCB, SRCB, SCB where RCB is taken from the FB pointed to by register 2 (FBRCB), SRCB is X'80', and SCB is X'00' (a string control byte of X'00' is an end-of-logical-record SCB; occurring immediately after an SRCB, such an SCB indicates a zero-length record).

$PERM sends the sequence RCB, SRCB, EOB where RCB is X'A0' (permission-granted for function described in SRCB), SRCB is taken from FBRCB of the FB pointed to by register 2, and EOB is X'00' (a zero RCB indicating logical-end-of-transmission-block). $PERM also sets the bit EWFPOST in the field FBEWF; this "early-post" bit is reset by $BSCA when it finds any permission-type control record whose SRCB matches FBRCB.

$REQ sends the sequence RCB, SRCB, EOB where RCB is X'90' (request-permission for function described in SRCB) and SRCB and EOB are as described for $PERM.

Code common to all three routines requests from $CKLEN three bytes of space in a MULTI-LEAVING buffer, moves the 3-byte sequence, and calls $BFLUSH to truncate the buffer and queue it on $BSCA's buffer chain.

$DCOM - DECOMPRESSION SUBROUTINE

$DCOM is called by one of the output processors (such as $PRINTER) to decompress a logical record from a MULTI-LEAVING buffer into an area whose starting address is supplied by the caller. (HASP transmits all data records to MULTI-LEAVING terminals in a compressed and truncated format). If decompression is successful, $DCOM returns to the caller at an offset of three bytes; if $DCOM recognized a logical end-of-file, it returns at an offset of zero.

To decompress a logical record, $DCOM first examines the address in FBCURL, 2-byte field in the caller's FB reserved for the use of $DCOM. If that field is nonzero, it has previously been set by $DCOM to point to the RCB following the last-decompressed logical record in the current buffer. If that RCB is not X'00', $DCOM decompresses to the caller's area (which must be two bytes longer than the maximum record length) the

record following the RCB, moves the SRCB to FBSRCB, saves the address of the next RCB in FBCURL, and returns to the caller as explained above.

But if FBCURL is zero, $DCOM checks if more buffers are queued on the caller's FB. (If FBCURL is nonzero but the RCB to which it points is zero, $DCOM first frees the current buffer and then proceeds as if FBCURL were zero.)

If one or more buffers are queued, $DCOM selects the first buffer, points to its first RCB, and decompresses a logical record as above. But if no buffers are queued, $DCOM waits for WORK, to be posted by $BSCA when the next buffer for the same output device is received.

The output buffer's address is specified by the caller in field FBAREA: on return, $DCOM replaces this field by the address of the last-plus-one output byte.

$CMPR - COMPRESSION SUBROUTINE

$CMPR compresses data from a user-specified input area to a local work area and transmits it to HASP by calling subroutine $CKLEN.

When called, $CMPR examines the status of its local work area. If the work area is busy, $CMPR has been called by some other processor and has in turn called $CKLEN; $CKLEN is nonprocess exiting until it can find sufficient bytes in a MULTI-LEAVING buffer to allocate to $CMPR. In this case, $CMPR nonprocess exits until its work area becomes free.

When the work area is free, $CMPR compresses into it the text pointed to by FBAREA. Compression consists of either full compression and truncation, only truncation, or neither compression nor truncation, as selected by the setting of the RMTGEN variable &COMP=. Once the record is compressed, $CMPR calculates its compressed length and calls $CKLEN with a request for the number of bytes it requires in a MULTI-LEAVING buffer. When $CKLEN returns, $CMPT moves the compressed record, shows its work area free, and returns to the caller.

$CKLEN - MULTI-LEAVING BUFFER ALLOCATION SUBROUTINE

$CKLEN returns to its caller the address in a MULTI-LEAVING buffer of the rightmost byte of an area whose length is specified by the caller.

The caller specifies a length in register one. If $CKLEN has a current buffer, its current buffer pointer points to the last-allocated byte. It adds to this the caller's specified length. If the resultant address is lower than two bytes before the end of the buffer, $CKLEN saves this address as its current buffer pointer and returns this address to the caller in register one. But if the resultant buffer address is not lower than two bytes before the end of the current buffer, $CKLEN truncates the buffer, queues it on $BSCA's buffer chain, and posts $BSCA by turning on flag BFPOST in byte BCF1. To truncate a buffer, $CKLEN moves the current buffer pointer to its first two bytes and the sequence

EOB, ETB (X'0026') to the two bytes after the byte pointed to by the current buffer pointer.

After having truncated and queued the current buffer or if on entry there was no current buffer, but not if entered via entry point $BFLUSH (in which case $CKLEN returns immediately after truncation and queueing), $CKLEN attempts to get another buffer to satisfy the caller's request. If no buffer is free, it nonprocess exits until one comes free. It initializes the current buffer pointer to point to what will eventually be the buffer's FCS2 byte. It initializes a pointer to the last byte available in the buffer, and it saves the address of the buffer's chain word in a third pointer. Then it allocates space for the caller and returns, as above.

$FREEBUF - MULTI-LEAVING BUFFER FREE SUBROUTINE

$FREEBUF dequeues the first buffer from the buffer chain word FBBUF of the FB addressed by register two upon entry; subtracts one from FBBCT, the count of buffers enqueued upon that FB; and compares the new count with FBBMX. If the compare is low, $FREEBUF ORs the 2-byte field FBFCS into the 2-byte field $FCS, posts the $BSCA processor, and sets flag BFCSON in both the subject FB's flags and the BSCA flag byte. (See the $BSCA processor description for a discussion of BFCSON.)

In any case, $FREEBUF queues the just-dequeued buffer on chain word $MLPOOL in last-in, first-out sequence. If the system was generated for a 5471 Console, $FREEBUF posts $CON, the Console Processor. Then $FREEBUF returns to its caller.

ABEND - CORE DUMP SUBROUTINE

ABEND produces a core dump on the 5203 Printer. The code for ABEND is assembled only if the RMTGEN specification &DEBUG=1 has been used. &DEBUG=1 also causes the generation of extra debugging code throughout the terminal program; some of the extra sequences of code generated contain conditional branches to ABEND. ABEND may also be called from the CE panel of the System/3 by setting the IAR to its address.

Each line produced by ABEND consists of a 4-character address, 64 characters representing the 32 bytes starting at that address, and their printable equivalent in 32 more characters, bounded at the left and the right by a single asterisk; or four asterisks in the address position followed by blanks, to indicate that all of core up to the next line's address or the end of core would have printed the same as the previous line. The ABEND dump routine requires a printer with at least 1200 print positions; if a 96-print-position printer is used, not all of the EBCDIC portion of the line will be printed.

The first six bytes of printed core contain the address recall register, register 1, and register 2 as of the time ABEND gained control; the remainder of core is intact.

$LOG - HASP ERROR RECORDING SUBROUTINE

$LOG is a reentrant subroutine which maintains in-core error recording counters. Each counter is two bytes long and has a maximum count of 65535. There are eight counters for each of the following bytes:

```
1442 Status Byte 2 (if &S31442=1)
1442 Status Byte 1 (if &S31442=1)
BSCA Status Byte 2
5203 Status Byte 2
5203 Status Byte 1
5424 Status Byte 1
```

The counters are captioned, printed, and reset by IHEREP at program load time and thus form a permanent record of unit checks associated with the above devices. Only those counters which represent unusual unit checks are printed by IHEREP.

$MSG - ERROR MESSAGE TRACING SUBROUTINE

$MSG adds the 4-byte coded entry addressed by register 1 to the circular trace table of error messages. This table is examined by the 5471 Console Processor and under certain conditions by the 5203 Output Console Processor; $MSG posts whichever of these processors has been generated.

The various error messages supplied to this routine by its callers are explained in the System/3 Operator's Guide.

$INIT - INTIALIZATION ROUTINE

$INIT gains control when program loading is complete. It sets the print chain image, reads and processes REP cards, sets the 5203 forms length register, sets the 5424 print buffer register, establishes communication, sets up buffers, and exits to the commutator.

To set up the print chain image, $INIT reads the printer status bytes. If the 48-character-set bit is on, it moves the LC image to the image area; otherwise, it moves the PN image. Then $INIT starts processing REPs.

The format of a REP card is:

```
column  2      9       17
        REP    addr    REP-data
```

where "addr" must be a valid hexadecimal core address of exactly four characters (or four blanks) and "REP-data" is a sequence of one or more replacement groups with the last group terminated by a blank and all other groups terminated by commas. A replacement group is a string of 2n (n any integer) hexadecimal characters. The blank after the last replacement group may be followed by comments.

Starting at the address specified by "addr" the REP routine will store bytes one at a time corresponding to byte pairs of the "REP-data" taken from left to right. If the "addr" specification is blank, bytes will be stored starting at the first byte after the byte last used by the preceding REP card (or at zero if there was no preceding REP card). A REP card whose "REP-data" field contains no data is valid; its "addr" field (if any) specifies the address of the first byte to be REPd if the next REP card's "addr" field is blank.

To process REPS, $INIT reads a card from the primary hopper of the MFCU; a read error will give an F3 halt. If the card image contains "REP" in columns 2-4, it is processed according to the above specifications, with absolutely no validity checking, and $INIT reads another card, as above.

If the card image contains "&MLBFSIZ=" starting in column 1, $INIT converts to binary the specified decimal buffer size (which must immediately follow the equal sign and be terminated by a blank) and substitutes the result for the default buffer size. Then $INIT reads the next card, as above.

If the card image contains "/*SIGNON" starting in column 1, $INIT overlays the default sign-on card with it and continues as if the card were an EOR card.

If the card image contains "EOR" (END-OF-REPS) in columns 2-4, $INIT terminates rep processing, loads the 5203 print forms length register and the 5424 print buffer address register and establishes communications.

To establish communications, $INIT first disables and then enables the BSCA. Next, it examines the sign-on card to see if dialing information was specified. If so, it determines the starting and ending addresses for the telephone number (which is not checked for validity) and loads these values into the current and stop address registers after first ensuring that the data line is unoccupied. (If the data line is occupied, $INIT assumes the operator dialed and waits for the data set to become ready.) After starting an auto-call operation and looping until an op-end interrupt occurs, $INIT checks for timeout status; if so, the auto-call unit returned an abandon-call-and-retry signal and a CA halt (call-aborted) occurs. When the operator resets the halt, the entire logic starting with disable-BSCA will be reexecuted. But if the timeout bit is off, $INIT assumes the call was successful and loops until a dataset-ready indication occurs, as above.

When the data set becomes ready, $INIT transmits the 2-byte sequence SOH-ENQ, a sequence recognized by HASP as a request from a MULTI-LEAVING terminal. If the receive part of this transmit/receive command ends with timeout, the operation is repeated; if it ends with any other abnormal status, one of two things occurs. If the system was generated with &DEBUG=1 and the address knobs on the System/3 console are set to any odd address, the System/3 halts; the halt indicators display a hexadecimal image of the BSCA error status byte. Otherwise, and when the operator resets the halt, the entire logic starting with disable-BSCA will be reexecuted.

If the receive operation ended normally, the two received bytes should be DLE-ACK0. If they are not, the transmit/receive operation is performed.

If DLE-ACK0 was received correctly, the message "COMMUNICATION ESTABLISHED" is printed on the 5203. If a 5471 was specified when the system was generated, its interrupts are enabled and the same message is printed on it. If a 5475 was specified, its interrupts are enabled. $INIT now performs buffer initialization.

Buffer initialization consists of three steps and overlays the initialization code with MULTI-LEAVING buffers. As the first step, the value of MULTI-LEAVING buffer size is set in the various locations throughout the program that requires it; it may have been changed by the &MLBFSIZ control card. Step two moves the actual buffer initialization code to low core, where it is executed as step three. Execution consist of chaining together all buffers but the first buffer (which contains the sign-on record and is afterward queued to the $BSCA processor) with the chain origin at $MLPOOL.

When buffer chaining is complete, the sign-on buffer is queued as mentioned and control passes to the commutator. $COM gives control in its turn to the $BSCA processor, which as a special, first-time function transmits to HASP the buffer containing the sign-on card image.

HASP IBM-1130 WORK STATION PROGRAM

The 1130 MULTI-LEAVING terminal program is designed to operate on a system with 8K words which contains the standard Binary Synchronous Communications Adapter.

The unit record equipment supported may include any or all of the following devices:

1. 1442 Reader/Punch or Punch
2. 2501 Reader
3. 1132 Printer
4. 1403 Printer
5. Console Keyboard/Printer.

Programs developed for the 1130 in conjunction with the HASP Remote Job Entry feature are assembled using the OS Assembler. The 1130 instruction set is generated through the use of macro-instructions corresponding to the actual 1130 hardware commands. Additionally, pseudo (assembler) operations are available to aid in the development of 1130 programs on System/360

The object decks produced by the OS Assembler are subjected to further processing by a program (LETRRIP) which condenses and changes the format of the EBCDIC decks to facilitate 1130 loading.

The remote terminal system for the 1130 is composed of several programs briefly described as:

RTPBOOT - A bootstrap loader consisting of a single "load mode" format card and several column binary and EBCDIC program cards. The function of RTPBOOT is to bootstrap an EBCDIC format loader (RTPLOAD) into 1130 core. RTPBOOT will load from either a 1442 or a 2501 Card Reader.

RTPLOAD - Loads into the upper segment of defined 1130 storage and then loads the main terminal program (RTP1130) into the lower extent of 1130 storage. RTPLOAD also processes REP cards and performs the initial processing of /*SIGNON control cards.

RTP1130 - The main terminal processing program which provides the MULTI-LEAVING support for the 1130.

The following sections provide more detailed information on the design and implementation of the above programs.

The subsequent sections present the basic structure of the terminal program for the 1130. Included are descriptions of the commutator logic and associated processors; system subroutines; processor subroutines; control block formats and data block general formats.

The documentation presented is introductory in nature. The user intending to modify the system should use the documentation in conjunction with a program listing that contains commentary in much greater detail.

COMMUTATOR PROCESSORS

Distribution of CPU time to the processors concerned with the functions necessary to support terminal devices is through programmed commutator logic. Each processor which needs CPU time and is dependent on external I/O device rates is represented by a commutator entry. The commutator entry consists of the following basic elements:

1. A named commutator "gate" which takes the form of a branch to the next commutator entry (gate closed) or a NOP if the entry is active (gate open)

2. A long-form branch to the active commutator main routine used if the gate is open

3. A named return point for reference by the main commutator routine

4. A named end to the commutator entry which is the address of the next commutator entry.

The basic structure as defined may also contain register save-restore sequences to be used for each entry-exit cycle through the commutator.

The processor entry from the commutator (gate open) usually provides for a method of setting a variable entry to the segments of the processor which are involved with waiting for I/O to complete or some system resource to become available. The general operation of the commutator involves the opening and closing of processor gates, the setting of variable entry points within the processors, the initiation and associated wait period for I/O operations, and the return to the commutator to share the CPU during wait periods. The last instruction in the commutator is a branch to the first instruction in the commutator which initiates the next cycle. The current system does not provide for a priority relationship among commutator processors.

The main commutator processors are contained in the RTP1130 system and briefly described in the following sections.

TPIOX - SCA Input/Output Control Processor

Controls the transmission of data and/or control records between HASP and RTP1130 via the SCA. All adapter I/O is initiated using the SCA I/O Supervisor - BSXIOS.

TPGET - Processor For TP Buffers From HASP

TPGET processes data received from HASP in the form of TP buffers or control records preprocessed by TPIOX. Control record processing is in the form of request-to-start or permission-to-send functions.

Data buffers are deblocked, decompressed, converted to appropriate codes (1403 Printer, 1442 Punch, etc.), and queued for the specified commutator I/O processor.

Control information pertinent to the unique requirements of each data type is provided through the associated UFCB.

TPPUT - Processor For Data Destined For HASP

TPPUT acquires a TP buffer from the free chain and collects data from defined sources (card reader(s), console keyboard, etc.) to be processed (converted, truncated, compressed, etc.) and inserted into the buffer which is queued for TPIOX transmission to HASP.

RDTFO - 2501 Card Reader Processor

RDTFO is a conditionally assembled processor which supports the 2501 Card Reader as a job entry device. The functions of monitoring for a 2501 ready condition; reading cards; requesting permission to transmit to HASP; waiting for permission to send; queueing data for TPPUT; transmitting end-of-file conditions; and device error recovery are contained in this processor.

RPFFT - 1442 Reader And/Or Punch Processor

RPFFT is a conditionally assembled processor which supports the 1442 - 5, 6 or 7 - as a card reader, card reader/punch, or a card punch only. The functions to be performed are controlled by the assembly variables chosen and the use of local operator commands, when applicable. The reader section of code monitors for a "ready" condition; reads cards for transmission to HASP via TPPUT; processes end-of-file communications and provides error recovery. The punch section of code waits for data to be punched through interrogation of a queue developed by the TPGET processor and provides error recovery and punch termination procedures.

PRFOT - 1403 Printer Processor

PRFOT is a conditionally assembled processor which supports the 1403 Printer as a terminal output device. The functions of monitoring for input to be printed; simulating carriage control operations; processsing end-of-file conditions; setting UFCB status information and error recovery are included in this processor.

PRETT - 1132 Printer Processor

PRETT is a conditionally assembled processor which supports the 1132 Printer as a terminal output device. The functions of monitoring for input to be printed; initialization of interrupt processing routines for the 1132 print scan operations; simulation of carriage control operations; processing end-of-file conditions; setting UFCB status information and error recovery are contained in this processor.

CONSL - Console Keyboard/Printer Processor

CONSL processes console keyboard input and prints on the typewriter messages originating from HASP or internal sources.

Keyboard input is initiated by activation of the INT REQ key and by the interrupt routine which sets a flag and opens the console routine gate. Note that the position of the KEYBOARD/CONSOLE switch is not interrogated, and input is assumed to be from the keyboard. The value of the console entry keys is read every commutator cycle and, if key 0 is on, is stored in location $ENTKEYS. All noncontrol character input is printed, and the card code value is stored for investigation at EOF time. If the first character of input is "." (period), the data is assumed to be a local command. All other data is transmitted to HASP for action as a HASP operator command.

Print input is obtained from a queue which originates locally and/or from HASP. Data to be printed may be EBCDIC or tilt-rotate code and black or red ribbon.

RTPET - Initialization Processor

This special commutator processor is responsible for the initialization functions necessary for the commencement of the 1130 terminal operation in conjunction with HASP. The major functions performed are:

1. Sets the interrupt transfer vectors for RTP1130 operation
2. Dynamically builds the TP buffer pool using the defined extent of 1130 storage, the end of the 1130 program, and the defined TP buffer size
3. Builds a TP buffer containing the sign-on information processed by RTPLOAD for transmission to HASP
4. Establishes SCA communications with HASP and prepares TPIOX for sign-on
5. Opens the commutator gates for all SCA and input processors
6. Disconnects initialization from the commutator
7. Branches to commutator, which initiates MULTI-LEAVING operation.

SYSTEM SUBROUTINES

The following are brief descriptions of the major subroutines contained in the RTP1130 program. These subroutines are available for use by any system commutator processor with the restriction that they may not be used at interrupt time. Detailed information concerning the calling sequences, input values, etc. may be found in the listing of the RTP1130 program.

SGETQEL - Dequeue An Element From a Chained List

Given the address of a chained list, SGETQEL returns the address of the first element available in the list and removes the element and rechains the list. The chain field of the dequeued element is set to zero before returning. If the chain is null, an indication is returned to the user.

SPUTFQL - Enqueue An Element In A Free Element Chain

Given the address of a free element chain pointer and the address of an element to be returned to the free chain, the element is returned to the free chain. The construction of the free chain is in random order depending on system processor utilization of the free element chain.

SPUTAQL - Enqueue An Element In An Active Chained List

The address of an element supplied by the caller is used to build a chained list in first-in, first-out order.

STPOPEN - Initiate Control Record Transmission

Control record communications with HASP in the form of request-to-start and permission-to-send sequences is the function of this routine. Input includes an indication of the control record type and a pointer to the UFCB for the device being processed.

SSRCHB - Search UFCB Chain For Matching RCB

The RCB code supplied by the user is used to search the UFCB chain for a UFCB with a matching RCB code. An indication of the status of the search is returned to the caller.

SWTOPR - Type Message On Console Typewriter

The caller supplies the address of a message in EBCDIC with control information indicating red or black ribbon and the number of characters to be typed. The address of a routine to be given control in the event that the message cannot be processed immediately must also be supplied.

The message is queued for processing by the console typewriter commutator routine.

SLOGSCA - Log SCA Error Messages On Console Typewriter

Error conditions associated with the SCA operation are logged on the console typewriter for information and possible remedial purposes. The format of the message logged is:

SCA LOG XXXXXXXX

Where the value of "XXXXXXXX" is determined by the caller and is in fact the contents of the ACC and EXT on entry to the routine.

An indication of the status of the request to log is returned to the caller.

SMOVE - Move A Variable Number Of Words

This routine provides for the moving of a specified number of words from a source block to a target block.

SXPRESS - Convert Card Code To EBCDIC

The card code (12-bit) input is converted to EBCDIC using a high-speed conversion algorithm in conjunction with a minimal conversion table. Special consideration is given to "blank" conversion under the assumption that most cards are dense with "blank" data.

SXCPRNT - EBCDIC To Console Printer Code Conversion

SXCPRNT converts a single EBCDIC character to the equivalent console printer tilt-rotate code, using a table lookup method.

SXPPRNT - EBCDIC To 1403 Printer Code Conversion

SXPRNT converts a single EBCDIC character to the equivalent 1403 Printer 6-bit-with-parity code, using a table lookup method.

SXCPNCH - EBCDIC To Card Code Conversion

SXCPNCH converts a single EBCDIC character to the equivalent 12-bit card code, using a table lookup method and conversion algorithm.

STRACE - Trace Machine Registers

STRACE stores the information shown below in a table of variable length. Each entry is the result of the execution of the linkage created by the STRACE macro. The trace table created at assembly time is circular.

Trace table entry:

Word	Description
1	Count of the number of entries for this $TRACE
2	Location +1 of caller to $TRACE
3	Contents of ACC
4	Contents of EXT
5	Contents of XR1
6	Contents of XR2
7	Contents of XR3

The count of the number of entries is also stored in the STRACE macro linkage.

The assembly of STRACE is a function of the variable $TRACE.

SSDUMP - System Core Dump

SSDUMP is a conditionally-assembled subroutine which allows post-mortem or dynamic dumps on either the 1132 or 1403 Printer. SSDUMP is assembled if &DEBUG SETA 1 is included in the RTP1130 source deck. Linkage to SSDUMP via location 0 is also established so that a post-mortem dump may be taken by pressing SYSTEM RESET and START.

The linkage to use this subroutine dynamically is contained in the system listing. Note that the logic of the subroutine does not allow concurrent operation of the selected printer and other devices.

PROCESSOR SUBROUTINES

The following are brief descriptions of the major subroutines which may be used by commutator processors subject to the restrictions that these routines are processor-dependent in their operation. For example, the SCA I/O Supervisor (BSXIOS) is used at initialization time and by the TP buffer manager but cannot be simultaneously used by these commutator processors.

BSXIOS - Low Speed BSCA Input/Output Supervisor

BSXIOS processes requests for transmit, receive, or program timer functions on the low-speed Binary Synchronous Communications Adapter. BSXIOS initiates the requested function and prepares the interrupt programs for the associated interrupt processing of the desired functions.

The status of the function performed by BSXIOS is contained in a communication cell which is addressed by a variable pointer word. A communication cell is defined for both read (receive) and write (transmit) operations. Various completion codes stored in the cells provide the status of the function with respect to normal or abnormal termination.

BSXIOS expects the caller to provide the address of an appendage routine to be entered at the termination (interrupt time) of every write operation. The purpose of the write end-of-operation appendage is to allow reinstruct (read operation) of the communications adapter as soon as possible after the write completion.

DBLOCK - Deblock, Decompress, Convert, And Store Data From HASP

DBLOCK locates a record (defined by RCB) in a TP buffer as specified by a given UFCB, decompresses, edits, and moves data to a selected target area. The target area must have the format described under "Output Element (Tank) Description."

The operation of DBLOCK includes the printing of the output tank with an initialization value supplied by the user (usually the value of a blank for the associated device); the updating of control information in the UFCB; the setting of control information in appropriate fields of the output tank; the automatic entry to conversion and store routines unique to the device associated with the UFCB supplied, and the communication of the status of the buffer being processed (end-of-file, end-of-block conditions).

TPCOMPR - Construct Records For Insertion In TP Buffers

TPCOMPR constructs a logical record consisting of a physical input record from attached 1130 devices (card reader(s), console, etc.). The logical record constructed consists of the original input after code translation, data truncation, and/or compression (optionally) and attachment of the control bytes necessary for HASP processing. The control bytes are per the standard HASP MULTI-LEAVING conventions.

The options listed below are set at assembly time to generate the supporting code.

1. No compression or truncation
2. Trailing blank elimination only (truncation)
3. Blank and duplicate compression and blank truncation

The current version of TPCOMPR assumes card code input.

DBUGSCAL - Trace Routine For Low Speed SCA

This routine is conditionally assembled as a function of "&DEBUG" and provides a trace of all SCA interrupts in the form shown below. Entry is from BSXIOS interrupt processing routines. External disabling of the SCA trace function is provided through the entry keys. The trace table limits are preset to use the upper 8K of a 16K 1130 and may be changed either by assembly or by the appropriate REP. See the program listing and refer to locations DBUGSTRT and DBUGSTND.

The trace table format is:

Word	Description
1	Operation type (BSXIOPT)
2	DSW at interrupt time
3	BSXIOS Completion Code (BSXOPF)
4	Location of interrupt
5	Data received/transmitted
6	Data transfer count
7	Read or write sequence index
8	Spare word

TPBUILD - Construct TP Buffers

TPBUILD constructs TP buffers for TPIOX transmission to HASP. Data to be inserted and length of insert are provided by user. TPPUT initializes this routine by providing the buffer to be used and setting pointers and variables.

The data to be inserted is usually in the form of a logical record as constructed by TPCOMPR.

RTP1130 CONTROL BLOCK AND DATA FORMATS

Chained List General Format

All queues maintained within RTP1130 are of the chained list form and consist of free queues and free queue pointers and active queues and active queue pointers. Free queues are chained in a random fashion

while active queues are maintained in a first-in, first-out order. The general form of a queue is:

Address of next element chain word. Set to zero if no element.

Variable length element.

Variable length element.

Last variable length element (chain word set to zero).

Examples of chained lists are: TP buffers, console message tanks, and printer data tanks, punch data tanks. The size and number of elements in the queue is variable according to the nature of the queue.

UFCB - Unit Function Control Block Description

Each device which transmits data to or from HASP via the communications adapter processor must be represented by a unit-function control block. The general format of a UFCB is:

Reference	Word	Description
UFCBCNW	0	Chain word to next UFCB
UFCBNFO	1	Information word... Input Byte 0 = Reserved Byte 1 = Input Code = 0 for IBM Card = 1 for PTTC/8 = 2 for EBCDIC
UFCBSAR	2	Status and RCB Code... Byte 0 = Status of unit-function = X'90' if request to start sent from input unit-function or if request to start received for output unit-function = X'A0' if permission to start received for input unit-function or if permission to start sent for output unit-function. Byte 1 = RCB code associated with this UFCB
UFCBFCS	3	Function control sequence bit associated with this UFCB (and RCB)

UFCBCOM	4	Address of commutator processor gate address for processor associated with this UFCB
UFCBFQP	5	Tank free queue pointer for output devices or address of input element for input devices
UFCBBFP	6	Queue pointer for active TP buffers for output devices or end-of-file flag for input devices
UFCBBFC	7	Count of active TP buffers for associated device
UFCBBFL	8	Limit of active TP buffers for associated device
UFCBPBP	9	Buffer address of current buffer being processed by TPGET processor
UFCBPBA	10	Address of next RCB in buffer being processed
UFCBPBS	11	Position indicator for next RCB in buffer being processed. Set to 0 if RCB right-justified. Set to 1 if RCB left-justified.
UFCBPWD	12	Output device width = 2*W/P, where W = actual width in characters and P = 2 for packed output tanks or P = 1 for unpacked output tanks
UFCBPRO	13	Address of data processing routine (usually a conversion program) for each character processed by $DEBLOCK
UFCBSTO	14	Address of routine to store data processed by "UFCBPRO" program

TPBUF - TP Buffer Element Description

All data transmitted to or from HASP is contained in variable length buffers (variable at generation time) with the following general format:

Reference	Word	Description
TPBUFCW	0	Chain word to next TP buffer
TPBUFST	1	Reserved
TPBUFCB	2	Buffer control word
		Byte 0 = 0 (Reserved)
		Transmit function...
		Byte 1 = Number of bytes to be transmitted minus 2 for end sequence which is inserted by BSXIOS

		Receive function...
		Byte 1 = Number of bytes received
		Timer function...
		Byte 1 = Number of program time interrupts processed before ending timer operation
TPBUFDT	3	Start of data area of length defined by "&MLBFSIZ" which includes...
TPBUFHD	3	BSC header value indicating the function (Read, write, timer) to be performed as defined by SCA function indicators
TPBUFBF	4	Control sequence...
		Byte 0 = BCB
		Byte 1 = first byte of FCS
TPBUFFR	5	Control sequence...
		Byte 0 = Second byte of FCS
		Byte 1 = RCB
TPBUFSR	6	Control sequence...
		Byte 0 = SRCB
		Byte 1 = SCB

OUTPUT ELEMENT (TANK) DESCRIPTION

Local terminal output devices (printers, punch, etc.) receive data via elements or tanks which are built by the commutator routine responsible for processing TP buffers transmitted by HASP. The general format of these tanks is described below.

Reference	Word	Description
TANKWRDA	0	Chain word to next tank
TANKWRDB	1	Reserved
TANKWRDC	2	Control word...
		Byte 0 = Reserved for device use
		Byte 1 = SRCB from record received

TANKWRDD	3	Control word...
		Byte 0 = Reserved for device use
		Byte 1 = Actual tank data count
TANKWRDE	4	Start of variable length data area determined at generation time

Note: The element chain word and the data area must start on even 1130 word boundaries.

OBJECT DECK FORMAT

The following is the format of the object decks (RPT1130, RTPLOAD) produced from OS/360 assembler output by LETRRIP.

TEXT Card

Column(s)	Description
1	"T" for TEXT card identification
2-3	Absolute 1130 load address
4	Word count of data field
5-72	Data field (maximum of 34 words)
73-74	Check sum of columns 1-72
75-76	Identification
77-80	Sequence number

END Card

Column(s)	Description
1	"E" for END card identification
2-3	Entry point to program loaded
4-72	Reserved
73-74	Check sum of columns 1-72
75-76	Identification
77-80	Sequence number

REP Card

Column(s)	Description
1	Any legal EBCDIC punch
2-4	"REP"
5	Blank
6	Load address format field: "L" for listing option where the specified load address corresponds to the OS/360 assembler listing. "X" for absolute 1130 core address
7	Currently unused but usually punched "0" for continuity
8-11	Load address for first data word and is incremented by 1 for each additional data word. REP cards may be continued by leaving this field blank
12	Blank
13	Format field for data following. Subject to same definition as column 6.
14-17	Data field to be loaded in the location computed as a function of columns 8-11
18	","

Columns 19 through 78 are in the same format as columns 13-18 with the exception of column 78, which must be blank. A blank in columns 18, 24, ...72 terminates the scan of the card.

Note: The "L" option causes the specified data to be divided by 2 for conversion from 360 byte data to 1130 word data.

EXAMPLES OF REP CARDS

1. The following cards:

```
Col

0   00      11
1   56      23

RREP L02208 X4C00,L004E,X4400,X000F

RREP        X74FF,X0000,X7101
```

would result in the code represented below starting in 1130 core location 1104 (Hex):

```
1104    $B      39,,L
1106    $TSL    15
1108    $MDM    0,-1
110A    $MDX    1,1
```

2. The following card:

```
Col

0   00      11
1   56      23

RREP L01772 X4C18,X1FF8
```

would be ignored because columns 2-4 unequal to "REP".

REMOTE TERMINAL MAIN LOADER (RTPLOAD)

RTPLOAD is an EBCDIC format loader which is loaded by RTPBOOT into the upper part of defined 1130 core. The 1130 core definition (which is a RMTGEN variable) is used to specify the origin of RTPLOAD. The format of RTPLOAD (and RTP1130) is given under Control Blocks and Data Formats.

RTPLOAD also reads and processes "REP" cards as well as the optional /*SIGNON control card.

The major functions of RTPLOAD are:

1. Clears storage from location 0 to "&RTPLORG-1"
2. Tests for a 2501 or 1442 Card Reader and initializes the card read routine for the appropriate device
3. Reads RTP1130 program cards, performing the conversion from card code to EBCDIC and loading the data into the specified locations
4. Sets up the entry to RTP1130 when the END card is processed
5. Reads, converts, and stores /*SIGNON and sets indicator for RTP1130 signaling existence if /*SIGNON encountered
6. Transfers control to RTP1130.

REMOTE TERMINAL BOOTSTRAP (RTP1130)

The bootstrap loader distributed in object form as shown in the subsequent pages is specifically constructed to "bootstrap" the EBCDIC main loader (RTPLOAD) into the storage locations defined by "&RTPLORG"

at RMTGEN time. RTPBOOT loads into lower 1130 storage via the load-mode format first card and following binary program cards and EBCDIC conversion table cards. RTPBOOT will load from a 2501 or 1442 Card Reader which is wired for the load-mode sequence initiated by the console LOAD button.

REMOTE TERMINAL PROGRAM 360 PROCESSING (LETRRIP)

LETRRIP (Loader for 1130 Relocatable Remote Interleaving Processor) is an OS program executed under OS as part of the RMTGEN procedure. The purpose of this program is to condense the object deck produced by the OS assembler; relocate address constants according to the requirements of the 1130 and produce a new object deck.

1130 INSTRUCTION MACROS

The OS Assembler Macro instruction listed on the following pages are used to assemble the RTP1130 and RTPLOAD programs as a part of the RMTGEN process necessary to create the 1130 work station program.

The general format of the instructions to be assembled with the macros is:

```
LABEL $OP ADDR,TAG,FMT,MOD
```

where:

LABEL is the statement label subject to the OS assembler rules and restrictions.

$OP is a macro from the set listed at the end of this section.

ADDR is the address field of the 1130 instruction.

TAG is the index register (TAG) field of the 1130 instruction.

FMT is the format indicator for the 1130 instruction:

```
FMT=L for long form
FMT=I for long form indirect address
FMT=X for short form absolute address
FMT='blank' for short form relative address
```

MOD is the modifier bits field required for some 1130 instructions.

Listed below are some of the conventions which must be followed to successfully use the macro package in producing a program for operation on an 1130.

1. All symbols starting with the character "$" are deemed to be absolute in value.

2. The symbols WA, WB, and WC are assumed to define absolute values. Note: WA, WB, and WC cannot be used as the first two characters of any relocatable symbols.

3. All other symbols are assumed to be relocatable as defined by the OS assembler SRL.

4. Parenthetical expressions are considered to be relocatable if contained in an instruction, e.g.,

   $AXT (*-*),WA,L is considered relocatable, whereas

   $AXT *-*,WA,L

   is considered absolute.

1130 instruction macros are:

Macro Form		Description And Notes
$LD	ADD,TAG,FMT	Load ACC
$LDD	ADD,TAG,FMT	Load double (ACC,EXT)
$STO	ADD,TAG,FMT	Store ACC
$STD	ADD,TAG,FMT	Store double (ACC,EXT)
$LDX	ADD,TAG,FMT	Load index
$LXA	ADD,TAG	Load index from address. A variation of $LDX with F=1 and IA = 1.
$AXT	ADD,TAG,FMT	Address to index true. Identical to $LDX.
$STX	ADD,TAG,FMT	Store index
$STS	ADD,TAG,FMT	Store status
$LDS	ADD,TAG	Load status
$A	ADD,TAG,FMT	Add
$AD	ADD,TAG,FMT	Add double
$S	ADD,TAG,FMT	Subtract
$SD	ADD,TAG,FMT	Subtract double
$M	ADD,TAG,FMT	Multiply
$D	ADD,TAG,FMT	Divide
$AND	ADD,TAG,FMT	Logical AND

$OR	ADD,TAG,FMT	Logical OR
$EOR	ADD,TAG,FMT	Logical exclusive OR
$SLA	ADD,TAG	Shift left ACC
$SLCA	ADD,TAG	Shift left and count ACC
$SLC	ADD,TAG	Shift left and count ACC and EXT
$SRA	ADD,TAG	Shift right ACC
$SRT	ADD,TAG	Shift right ACC and EXT
$RTE	ADD,TAG	Rotate right ACC and EXT
$BSC	ADD,TAG,FMT,MOD	Branch/skip on condition
$BOSC	ADD,TAG,FMT,MOD	Branch/skip and reset interrupt
$BP	ADD,TAG,FMT	Branch ACC positive (long)
$BNP	ADD,TAG,FMT	Branch ACC not positive (long)
$BN	ADD,TAG,FMT	Branch ACC negative (long)
$BNN	ADD,TAG,FMT	Branch ACC not negative (long)
$BZ	ADD,TAG,FMT	Branch ACC zero (long)
$BNZ	ADD,TAG,FMT	Branch ACC not zero (long)
$BC	ADD,TAG,FMT	Branch on carry (long)
$BO	ADD,TAG,FMT	Branch on overflow (long)
$BOD	ADD,TAG,FMT	Branch ACC odd (long)
$SKPP		Skip ACC positive (short)
$SKPN		Skip ACC nonzero (short)
$SKPZ		Skip ACC zero (short)
$SKPO		Skip overflow off (short)
$SKPC		Skip carry off (short)
$SKPX		Skip ACC not equal zero and carry off (short)
$B	ADD,TAG,FMT	Branch unconditionally. FMT = L or I generates long form $BSC with MOD = 0.
$BSI	ADD,TAG,FMT	Branch conditionally and store IAR

$TSL	ADD,TAG,FMT	Transfer and store location counter. Assembled as a $BSI with FMT = L, MOD = 0 (long form unconditional branch and store IAR).
$MDX	ADD,TAG,FMT	Modify index and skip
$STL	ADD,FMT	Store location counter. Assembles as $STX ADD,0,FMT.
$MDM	ADD,VALUE	Modify memory
$WAIT		Wait for interrupt
$XIO	ADD,TAG,FMT	Execute I/O
$BSS	N,X	Block started by symbol N = number of words X = E for even storage
$BES	N,X	Block ended by symbol N = number of words X = E for even storage
$NULL		Null operation for symbol definition
$ADCON	ADDR	Address constant. Assembles as an absolute 1130 address. "ADDR" must be a relocatable symbol by the OS assembler definition.
$NOP		No operation. Assembles as $SLA 0.
$ZAC		Clear ACC. Assembles as $SRA 16.

GENERAL INFORMATION

OS Assembly Output

If the value of &FULLIST is set to 1 at the time of generation of RTP1130 or RTPLOAD, then the listing produced by the OS Assembler will contain the following information:

1. The location counter value for each 1130 instruction or storage location in terms of bytes. The actual 1130 location in terms of words can be determined by dividing the displayed value by 2. The REP facility allows a specification of either byte or word form.

2. The 1130 instruction is printed in 1130 format. The long form address is in terms of 1130 words and the short form is true relative format.

Variable Internal Parameters

The generation of the RTP1130 program using RMTGEN provides the user with a simple and flexible means of changing common parameters germane to the configuration of the 1130. Additional internal parameters may be varied by using the source file update feature of the RMTGEN program.

Listed below are the major parameters, with a brief description of each, which the user might consider altering as a function of hardware and software performance considerations.

Variable	Description
&DEBUG	Conditionally assembles the RTP1130 internal core dump program ($SDUMP) and the BSC adapter trace routine (DBUGSCAL). Default value inhibits the assembly of these debugging programs.
&CNPSIZE	Maximum console printer message size. Default value is 120 bytes per message.
&CONINSZ	Maximum console keyboard input buffer size. Default value is 120 characters per command.
&PRFOTKL	Number of 1403 Printer buffers (tanks) provided at assembly time. Default value is 2. The TPGET Processor will build up to the value of &PRFOTKL and then suspend operation for the 1403 until the count of buffers falls below &PRFOTKL.
&PRETTKL	Number of 1132 Printer buffers (tanks) provided at assembly time. Default value is 2. See &PRFOTKL for TPGET action.
&PUNFTKL	Number of 1442 Punch buffers (tanks) provided at assembly time. Default value is 2. See &PRFOTKL for TPGET action.
&CONSTKL	Number of console printer buffers (tanks) provided at assembly time. Default value is 5. See &PRFOTKL for TPGET action.
&PRFOBFL	Maximum number of TP buffers containing data destined for the 1403 Printer which will be accepted by TPIOX before setting the transmission suspension bit defined in the FCS for the 1403. HASP will suspend transmission of 1403 print data until the FCS bit is reset when the number of 1403 TP buffers becomes less than the value of &PRFOBFL. Default value is 2.
&PRETBFL	Same definition as &PRFOBFL except it applies to the 1132 Printer. Default value is 2.

&PUNFBFL Same definition as &PRFOBFL except it applies to the 1442 Punch. Default value is 2.

&CNSPBFL Same definition as &PRFOBFL except it applies to the console printer. Default value is 1.

&NPTFBFL Maximum number of TP buffers allotted to input devices collecting data to be sent to HASP. Default value is one greater than the number of card readers defined for RTP1130.

The following sections outline the basic logic flow of the MULTI-LEAVING Remote Terminal Processor program for System/360 (including Model 20) work stations utilizing Binary Synchronous Communications (BSC) devices. The same work station program is utilized for both the Model 20 and System/360 work stations, depending on the RMTGEN parameter &MACHINE.

The MULTI-LEAVING Remote Terminal Processor program is created by the RMTGEN to operate as an extension of HASP on any model of System/360 used as a remote work station for HASP. This terminal program maintains constant communication with HASP at the central site via several classes of telephone lines to (1) encode and transmit jobs submitted at the remote site for processing on the central computer and (2) print and/or punch the output from jobs thus submitted as the output becomes available. Optionally, if an operator console is attached to the remote system, informational and control facilities are provided. All of the above functions may occur simultaneously. Various techniques are utilized by HASP and the work station program to obtain maximum performance of the remote devices and the communications line.

The MULTI-LEAVING Remote Terminal Processor consists of an initialization section, four principal processors, three communications interface processors, and a Communications Input/Output Supervisor. Allocation of CPU time to the various processors is accomplished through a basic program commutator. A processor is entered into contention for CPU time by changing its commutator entry from a NOP to a branch instruction. A single control block, the Total Control Table (TCT) is utilized by all processors to provide for synchronization of concurrent operations, processor status information, reenterability and both interprocessor and intraprocessor communication.

The following sections discuss the basic logic flow of the various components of the program.

COMMUNICATIONS INTERFACE PROCESSOR - OUTPUT ($TPPUT)

This processor serves as the interface between the various input processors and the Communications Input/Output Supervisor. Its function is to compress and encode records for subsequent transmission to HASP at the central site. $TPPUT is utilized as a subroutine by the various input processors and relieves the input routines of the responsibility of data compression and transmission buffer management. As records are submitted for transmission, $TPPUT compresses the records according to a compression type generation parameter (&CMPTYPE) and adds the encoded record to its current output buffer. When the current buffer is filled or truncated, it is chained in an ordered queue for transmission to HASP by the Communications Input/Output Supervisor and a new buffer obtained.

Details of the compression and encoding technique utilized by $TPPUT are included as an appendix to this manual.

COMMUNICATIONS INTERFACE PROCESSOR - INPUT ($TPGET)

This processor serves as the interface between the various output processors (Print, Punch, Console, etc.) and the Communications Input/Output Processor. Its function is to decode and uncompress transmission buffers received from HASP and to queue the decompressed records to the processor for processing. $TPGET is entered from the commutator and processes buffers from a ordered queue of received buffers established by the Communications Input/Output Supervisor. Records received are deblocked into "decompression tanks" and passed to the appropriate processor. Synchronization and passage of the tanks to the processors is accomplished through the Total Control Table (TCT) for each processor. $TPGET additionally is partially responsible for metering the flow of each type of record from HASP. This is accomplished by utilizing the various buffer and tank limits indicated in the TCT for each processor.

CONTROL RECORD PROCESSOR ($CONTROL)

This processor provides synchronization between the various processing functions at the work station and the HASP System at the central site. Control records from HASP (i.e. request to start a function, etc.) are queued on this processor by the $TPGET processor. $CONTROL then processes the control record, transmits a response, if required, through $TPPUT and initializes the required functional processor.

COMMUNICATIONS INPUT/OUTPUT SUPERVISOR (COMSUP)

COMSUP maintains communications with HASP in the central CPU at all times and is responsible for the transmission of all data to and from the remote site. The data processed by COMSUP is always in compressed buffer form and passes to and from COMSUP via ordered queues established by $TPPUT and for $TPGET.

The communications I/O is primarily interrupt driven and is completely maintained by COMSUP (i.e., COMSUP is both the initiator and executor of communications I/O). During periods requiring no data transmission, COMSUP maintains a "handshaking" cycle with HASP at approximately 2-second intervals to ensure full bi-directional capabilities and to avoid unprogrammed timeouts of the adapter.

In addition, COMSUP maintains, verifies, and corrects (if necessary) the MULTI-LEAVING block sequence check and detects, logs, and retries all communications errors. COMSUP queues all input data buffers to the appropriate processor TCT queues and provides metering of the various streams to the extent of preventing HASP output stream backup when responding to HASP.

INITIALIZATION PROCESSOR

The Initialization Processor receives control from the loader and initializes the remote terminal program as follows:

1. If the CPU is not Model 20, general registers 1, 2, and 3 are loaded to establish 16 K addressability.

2. Replacement (REP) cards are read from READER 1 for possible modifications to the program. The format of the REP card is as follows:

Col.	2-4	REP
Col.	9-12	Replacement address - hexadecimal address of the first half word of storage to replace (if blank, the previous REP card is continued)
Col.	17-n	xxxx,xxxx,...xxxx replacement data - one or more half-word groups of hexadecimal data separated by commas
Col.	n+1	blank - terminator for the replacement data
Col.	n+2-80	comments - any text

   Each REP card is printed on PRINTER 1 when read as a record of program modification. REP reading is terminated when either a blank card (blank in Col. 1-5) or a /*SIGNON card is encountered.

3. The HASP ENVIRONMENT RECORDING ERROR PRINTOUT (HEREP) is printed if the recording table is intact from the last execution of the program; otherwise, a new table is created for future recording and print out.

4. Interrupt PSW's are set for non-model 20 CPUs.

5. The communication adapter is enabled, and communications are established with HASP as follows:

   a. Write SOH-ENQ to HASP

   b. Read for DLE-ACK0 from HASP

   If I/O errors occur or HASP responses do not match the expected sequence, the sequence is repeated.

6. The processor constructs a buffer pool over itself and queues the sign-on record for transmission to HASP.

7. I/O PSWs are set (I/O old points to commutator) and control is passed to the Communication Adapter Interrupt routine.

PRINT SERVICE PROCESSOR - $PRTN1

The Print Service Processor's major functions are dequeueing decompression tanks containing print information from the printer Total Control Table, examining the subrecord control byte for carriage control

information, performing required carriage control, printing the information on the designated printer, and releasing the used decompression tank to the pool. The processor also provides event control upon dequeueing and releasing the tanks. If no console typewriter is attached to the system and the value of the user option &PRTCONS is not zero, the processor will set status information at the end of each print data set which allows the console processor to queue operator messages for printing.

INPUT SERVICE PROCESSOR - $RRTN1

The Input Service Processor supports various card readers used for the purpose of submitting job streams to HASP and in the case of Model 20 DUAL 2560 MFCM serves the functions of Punch Service Processor. The processor provides error analysis and recovery for supported devices. Execution begins with the initial read routine which continuously attempts to read cards from the designated card reader. In the case of a DUAL 2560, control is passed to the punch routine if the primary feed is empty. If the reader is a DUAL 2520 or 1442, the routine will check the first card for blank and if so pass control to the punch preparation routine; otherwise, subroutine $TPOPEN is called to send a request to send a job stream to HASP. When permission is received the job stream submission routine is entered to read cards into one of two decompression tanks, calling the $TPPUT processor which compresses the data and schedules transmission to HASP. At end-of-file, $TPPUT is used to signal HASP, and control is passed to the initial read routine.

The DUAL 2560 Punch routine attempts to dequeue a decompression tank from the Total Control Table. If successful, the card image is punched and the used "tank" is released to the pool. The routine continues to dequeue and punch for a maximum of 100 cards; at this time, tests are made to determine the existence of cards in the primary feed. The tests are also made in the event of no tanks available for dequeueing. If the tests are negative, the processor continues to punch cards; otherwise, control is passed to the read routine following the initial read. The processor provides event control upon dequeueing and releasing decompression tanks.

Dual 2520/1442 Punch preparation routine tests for:

1. Operator signal - changing of the data dials, .SR1 command, or unsolicited device end (depends on configuration).

2. Presence of decompression tanks for punching.

If the operator signals, the routine passes control to the initial read routine. If a "tank" is queued to the device Total Control Table, control is passed to the Punch Service Processor ($URTN1).

PUNCH SERVICE PROCESSOR - $URTN1

The Punch Service Processor's major functions are dequeueing decompression tanks containing punch information from the punch Total Control Table, punching the information into cards on the designated punch, and releasing the used tanks to the pool. The processor also provide event control upon dequeueing and releasing the tanks, in addition to error recovery upon erroneous punching of data. If the device is a DUAL 2520 or 1442, control is passed to the Input Service Processor ($RRTN1) after servicing each output record.

CONSOLE SERVICE PROCESSOR - $WRTN1

If the remote terminal has an attached operator printer keyboard, the console processor performs the following functions:

1. Reads operator commands from the console keyboard.

2. Examines the input for local commands (Model 20 only), passing local commands to the command processor and passing all other commands to HASP.

3. Type operator messages contained in decompression tanks queued to the console Total Control Table.

4. Convert codes in the Error Message Log Table to readable form and type the resulting messages.

Execution begins with the processor testing for an operator command in the console input tank waiting to be transmitted to HASP. If so, the console read-in function is skipped, and an attempt is made to send the command to HASP. Control is passed to the console output routine, which tests for output messages. If so, the processor dequeues the output tank, types the message, and releases the tank. Control is then passed to the beginning of the processor. If no output messages are pending, the Console Logging routine is entered which converts the message to readable form, types the message, and passes control to the beginning of the processor. The console read routine tests for operator requests and, if a request is pending, reads the command from the keyboard, calls the $TPPUT processor to compress the data and transmit the command to HASP, and passes control to the console output routine. If the remote terminal is a Model 20, the read routine tests for local commands and calls the Command Processor which, in case of ".S" command, and posts the appropriate Service Processor. Local commands are not transmitted to HASP.

The Console Service Processor without a console keyboard exists only when the value of the user option &PRTCONS is not zero. Execution begins with a test for printer availability. If available, any console messages are removed from the console output queue by the dequeue routine and are attached to the printer queue, allowing the Print Service Processor to print the message. If no console messages are queued the processor will convert any log messages into readable form, move the resulting message into a tank obtained from the pool, queue it to the console output queue and pass control to the console dequeue

routine. If the value of &PRTCONS is one and the printer is not available, console messages are allowed to accumulate to a maximum queue limit. If the limit is reached prior to the printer becoming otherwise available, the printer is forced available and the messages are queued to the printer with the subrecord control byte of the first message set to skip to channel 1 before print. If the value of &PRTCONS is two and the printer is not available to the console, the processor will dequeue console tanks and release them to the pool.

TOTAL CONTROL TABLE (TCT)

The Total Control Table is the major working storage area for the unit record processors and is customized for each configuration and device supported by the remote terminal program. Each basic TCT field may be referred to by using symbols defined in the DSECT named TCTDSECT; however, each processor has the option of uniquely referring to the fields directly by using the alternate 3-character prefix to each field name as follows:

TCT = General TCT prefix

CCT = Control record TCT

PCT = Printer TCT

RCT = Reader TCT

UCT = Punch TCT

WCT = Console TCT

Appropriate DSECTs are provided by generation macros in the event that more than one TCT of a given type is supported by the system. Basic control fields appearing only in systems with model numbers above the Model 20 are as follows:

Name	Description
$pCTCOMn	TCT addressability field - the commutator branches to this field to give control to the appropriate processor - the field contains a BALR R7,0 instruction which sets up TCT addressability for the processor - symbol characters "p" and "n" uniquely identify the TCT for the commutator
TCTSTRT	First two characters of unconditional branch instruction
TCTENTY	S-type address constant pointing to the appropriate processor - the field completes the branch instruction, which passes control to the processor at the desired entry point

TCTRTN — Return to next entry in commutator - each processor waits by branching to this field of the TCT, which in turn branches to the commutator

TCTCCW — Actual CCW opcode used in last I/O on the device - set by the processor and unit record IOS

TCTDATA — Address of data area used for last I/O transfer or address of input tank currently being compressed for transmission to HASP

TCTFLAG — CCW flags

TCTOPCOD — Opcode, which will be inserted into the TCTCCW field upon normal entry to unit record IOS

TCTCCWCT — CCW count field - length of data last transferred or to be transferred

TCTSENSE — Sense information - set by unit record IOS for error diagnostic purposes

TCTUCB — Device address - contains hexadecimal device address for SIO and interrupt recognition purposes - the high-order bit of the field is set on by the processor when waiting for HASP to authorize job submission.

TCTECB — Event Control Block - contains all bits stored in CSW byte 4 since the last SIO instruction for the device - busy bit is set at SIO and when the processor desires to wait for unsolicited device end - busy bit is reset at device end

TCTALTOP — Alternate opcode for DUAL reader/punch devices - processors requiring alternate opcodes have the option of setting the TCTTCCW field with the contents of this field prior to entry to unit record IOS

TCTSAV1 — Save area for the processor subroutine LINK register

Basic fields which may appear in remote terminal programs for all 360 models are as follows:

TCTNEXT — Next TCT in the chain of TCTs

TCTFCS — Function Control Sequence Mask - used by $TPGET Processor to setup the FCS transmitted to HASP for backlog control

TCTRCB — Record Control Byte - records from HASP which have RCB byte identical to this field will be queued for output on the corresponding device

TCTSTAT — Status Flag - each bit has one or more meanings which are dependent upon the processor involved:

bit 0 = TCTOPEN - always off indicating device is in use by HASP output (as appropriate)

bit 1 = TCTACT - used by $TPGET to determine which output devices need more data - processors set bit 1 when dequeueing output tanks

bit 2 = TCTSTOP - device has been stopped and is awaiting a start command

bit 3 = TCT1052, TCT2152 - console device identifier

bit 4 = PCT only = TCT1403, TCT1443, TCT2203, TCTPRTSW - indicates the status of the corresponding printer - if set the printer is available for printing operator messages

bit 4 = WCT only = TCTREQ - console request - operator desires to enter a command

bit 4 = UCT only = TCT1442 - the device is a 1442 with single stacker pocket

bit 5 = RCT or UCT = TCT2540 - TCT is for a 2540

bit 5 = WCT only = TCTREL - release requested - an unsuccessful attempt has been made to obtain a buffer for command transmission to HASP - the command is in compressed form in the console's tank waiting for a free buffer

bit 6 = RCT/UCT = TCT14420, TCT25600 - TCT is for a DUAL 1442 Reader/Punch or DUAL 2560 MFCM

bit 7 = RCT/UCT = TCT25200 - TCT is for a DUAL 2520 Reader/Punch device

TCTCOM Pointer to corresponding commutator entry

TCTID Optional field - 2-character identification for local command processors

TCTINRCB Optional field - exists when DUAL devices are attached to the system - identifies the Input Service Processor function as opposed to the Punch Service Processor function identified by TCTRCB - TCTINRCB is equated to TCTRCB if no DUAL devices are attached

The following fields are normal device extensions and do not exist for card reader devices when DUAL devices are not attached to the remote terminal:

TCTTANK Beginning of output tank queue - output records appear in unit record image form

TCTBUFER — Beginning of output buffer queue - contains records in compressed form waiting for decompression into tanks

TCTINKLM — Tank limit - maximum number of tanks which may be placed in the TCTTANK queue

TCTTNKCT — Tank count - actual number of tanks queued to the TCT

TCTBUFLM — Buffer limit - maximum number of output buffers which may be placed in the TCTBUFFER queue before signaling HASP to suspend sending the streams - limit is ignored for WCT

TCTBUFCT — Buffer count - actual number of buffers queued to the TCT

Reader and console TCTs have extensions which are used as tanks for records transmitted to HASP. These tanks belong to the device (2 for readers and 1 for the console) and are not released to the tank pool. The following field symbols are only defined for the TCTs with prefix designators RCT, WCT, and for DUAL devices UCT:

RCTTANK1, RCTTANK2 — Tank origin and working storage

RCTTRCB1, RCTTRCB2 — Input RCB for HASP identification

RCTTSRC1, RCTTSRC2 — Subrecord control byte = X'80'

RCTTCT1, RCTTCT2 — Count field - length of data portion

RCTTDTA1, RCTTDA2 — Data area - input card or operator command - will be blank for the DUAL 2520 and 1442 while in output status.

SECTION 4

HASP

DIRECTORIES

SYMBOL	<======= REFERENCED IN HASP ASSEMBLY MODULE =======>											
$ACTION	....	....	CON	....	....	...	PRPU	RDR	....	...	...	XEQ
$ACTIVE	....	COMM	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$ALL	....	....	...	....	....	NUC	....	...	....	...	...	...
$ALMSGSW	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
$ALWAYS	....	COMM	...	....	MISC	NUC	....	...	....	...	...	XEQ
$ASYNCQ	....	....	...	....	....	NUC	....	...	....	SVC	...	...
$BUFPOOL	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$BUSYQUE	....	....	CON	....	....	NUC	....	...	RTAM	...	...	...
$CKPTACT	....	COMM	CON	....	MISC	NUC	PRPU	...	....	...	...	...
$CKPTIME	....	....	...	....	MISC	...	....	...	....	...	...	...
$CKPTRAK	....	....	...	INIT	MISC	...	....	...	....	...	...	...
$COMMCT	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	...
$COMMQUE	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	...
$CURPCE	....	....	...	....	....	NUC	....	...	....	...	...	XEQ
$CVTPTR	....	COMM	CON	INIT	....	...	....	RDR	....	SVC	WTR	XEQ
$CYLMAP	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$DACKPT	....	....	...	INIT	....	...	....	...	....	...	...	...
$DATAKEY	....	....	...	....	....	...	....	RDR	....	...	...	...
$DCBLIST	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
$DCTPOOL	....	COMM	...	INIT	....	NUC	....	RDR	....	...	...	...
$DISALL	ACCT	COMM	CON	INIT	....	NUC	....	RDR	RTAM	...	...	XEQ
$DISTERR	....	....	...	....	MISC	NUC	PRPU	...	....	...	...	XEQ
$DOM	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
$DOMACT	....	....	CON	....	....	NUC	PRPU	...	....	...	...	...
$DOMQUE	....	COMM	CON	....	....	NUC	....	...	....	...	...	...
$DRAINED	....	COMM	...	....	....	NUC	PRPU	...	....	...	...	...
$ENBALL	ACCT	COMM	CON	INIT	....	NUC	....	RDR	RTAM	...	...	XEQ
$ERR	....	....	...	....	MISC	NUC	....	...	....	...	...	...
$ERROR	....	....	...	....	....	NUC	....	...	RTAM	...	...	XEQ
$EWBBUF	....	....	...	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$EWBCKPT	....	COMM	CON	....	....	NUC	PRPU	...	....	...	...	XEQ
$EWBCMB	....	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$EWBDDB	....	....	...	....	....	...	....	...	....	...	...	XEQ
$EWBHOLD	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
$EWBIO	....	COMM	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$EWBJOB	....	....	...	....	MISC	...	PRPU	RDR	....	...	...	XEQ
$EWBJOT	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
$EWBOLAY	....	....	...	....	....	NUC	....	...	....	...	...	...
$EWBOPER	....	....	...	....	MISC	NUC	....	...	....	...	...	...
$EWBOROL	....	....	...	....	....	NUC	....	...	....	...	...	...
$EWBSMF	....	....	...	....	....	NUC	....	...	....	...	...	...
$EWBTRAK	....	....	...	....	....	NUC	....	...	....	...	...	...
$EWBUNIT	....	....	...	....	MISC	...	PRPU	RDR	....	...	...	XEQ
$EWBWORK	....	COMM	CON	....	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
$EWBXFER	....	COMM	...	....	....	NUC	....	...	....	...	...	...
$EWFBUF	....	....	...	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$EWFCKPT	....	COMM	CON	....	MISC	NUC	PRPU	...	....	...	...	XEQ
$EWFCMB	....	COMM	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ

```
SYMBOL      <======== REFERENCED IN HASP ASSEMBLY MODULE ========>

$EWFDDB     .... .... ... .... .... ... .... ... .... ... ... XEQ
$EWFHOLD    .... COMM ... INIT .... ... .... ... .... ... ... XEQ
$EWFIO      .... COMM ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$EWFJOB     .... COMM ... .... MISC NUC PRPU RDR .... ... ... XEQ
$EWFJOT     .... COMM ... .... .... NUC PRPU ... RTAM ... ... ...
$EWFOLAY    .... .... ... .... .... NUC .... ... .... ... ... ...
$EWFOPER    .... .... ... .... MISC NUC .... ... .... ... ... ...
$EWFOROL    .... .... ... .... .... NUC .... ... .... ... ... ...
$EWFPOST    .... COMM CON INIT .... NUC PRPU RDR RTAM SVC ... XEQ
$EWFSMF     ACCT .... ... .... .... NUC .... ... .... ... ... ...
$EWFTRAK    .... .... ... .... .... NUC .... ... .... ... ... ...
$EWFUNIT    .... COMM ... .... MISC NUC PRPU RDR RTAM ... ... XEQ
$EWFWORK    .... COMM CON .... MISC NUC PRPU RDR RTAM SVC ... XEQ
$EWFXFER    .... COMM ... .... .... NUC .... ... .... ... ... ...
$EXCP       .... .... ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$EXCPCT     .... .... ... .... .... NUC .... ... .... ... ... ...
$EXITNOP    ACCT COMM CON INIT MISC NUC PRPU RDR RTAM SVC ... XEQ
$EXTPCLO    .... .... ... .... .... NUC PRPU RDR RTAM ... ... ...
$EXTPGET    .... .... ... .... .... NUC .... RDR RTAM ... ... ...
$EXTPOPE    .... .... ... .... .... NUC PRPU RDR RTAM ... ... ...
$EXTPPUT    .... .... ... .... .... NUC PRPU ... RTAM ... ... ...
$FREEBUF    .... .... ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$FREEMSG    .... COMM CON .... .... NUC .... ... RTAM ... ... XEQ
$FREEQUE    .... .... CON INIT .... NUC .... RDR RTAM ... ... ...
$FREUNIT    .... .... ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$GETBUF     .... .... ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$GETPBUF    .... .... ... .... .... NUC .... ... RTAM ... ... ...
$GETSMFB    .... COMM ... INIT MISC NUC PRPU ... RTAM ... ... ...
$GETUNIT    .... .... ... INIT MISC NUC PRPU RDR RTAM ... ... XEQ
$HARDCPY    .... COMM ... .... .... ... PRPU ... .... ... ... ...
$HASPECB    .... COMM CON INIT .... NUC .... ... .... SVC ... XEQ
$HASPECF    ACCT COMM CON INIT MISC NUC PRPU RDR RTAM SVC ... XEQ
$HASPTCB    .... .... CON INIT .... ... .... ... .... SVC ... XEQ
$HI         .... COMM ... .... MISC NUC .... RDR RTAM ... ... ...
$HVT        .... .... ... INIT .... ... .... ... .... SVC ... ...
$HVTEXCP    .... .... ... .... .... ... .... ... .... SVC ... ...
$INPUT      .... .... CON INIT .... ... .... RDR .... ... ... ...
$IOERROR    .... .... ... .... MISC NUC PRPU ... RTAM ... ... XEQ
$IOTEST     .... .... ... INIT MISC ... PRPU RDR .... ... ... XEQ
$JCTTEST    ACCT COMM CON INIT MISC NUC PRPU RDR .... ... ... XEQ
$JITABLE    .... .... ... INIT .... NUC .... ... .... ... ... ...
$JITCKPT    .... .... ... .... MISC ... .... RDR .... ... ... ...
$JOBNO      .... COMM ... INIT .... ... .... RDR .... ... ... ...
$JOBQPTR    .... COMM ... INIT MISC NUC PRPU RDR .... ... ... XEQ
$JOEBUSY    .... COMM ... INIT .... ... PRPU ... .... ... ... ...
$JOECKV     .... .... ... INIT .... ... PRPU ... .... ... ... ...
$JOEIRTE    .... COMM ... .... .... ... PRPU ... .... ... ... ...
$JOT        .... .... ... INIT MISC ... PRPU ... .... ... ... ...
```

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
$JOTABLE	....	COMM	...	INIT	....	...	....	...	....	...	...	...
$JOTCKPT	....	COMM	...	....	MISC	...	PRPU	...	....	...	...	...
$JQENT	....	COMM	CON	INIT	MISC	NUC	....	...	....	...	...	XEQ
$JQFREE	....	....	...	INIT	....	NUC	....	RDR	....	...	...	...
$L	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$LOG	....	COMM	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$LOGQUE	....	....	CON	....	....	NUC	....	...	....	...	...	XEQ
$L1	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$L2	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$L3	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$L4	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$L5	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$L6	ACCT	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$MAIN	....	....	...	....	....	NUC	....	RDR	....	...	...	XEQ
$MSGRPNO	....	....	...	INIT	....	...	....	...	....	...	...	...
$NORMAL	....	....	...	....	....	...	PRPU	RDR	RTAM	...	...	XEQ
$NUCLEN	....	....	...	INIT	....	...	....	...	....	...	...	...
$NUCTABL	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
$ODEL	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
$OLINK	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$OLOAD	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
$OPTCOLD	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTFMT	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTLIST	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTNFMT	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTNLST	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTNREP	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTNREQ	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTNTRC	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTRACE	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$OPTREP	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTREQ	....	....	...	INIT	....	...	....	...	....	...	...	...
$OPTSTAT	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$OPTWARM	....	....	...	INIT	....	...	....	...	....	...	...	...
$ORET	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
$OUTPUT	....	COMM	CON	INIT	....	...	PRPU	RDR	....	...	...	XEQ
$OXCTL	....	COMM	...	INIT	....	NUC	....	RDR	....	...	...	XEQ
$PCEORG	....	COMM	...	INIT	....	NUC	....	...	....	SVC	...	...
$PRCHKPT	....	....	...	INIT	MISC	...	PRPU	...	....	...	...	...
$PURGE	....	....	...	INIT	MISC	...	PRPU	...	....	...	...	XEQ
$PURGER	....	....	...	....	MISC	NUC	....	...	....	...	...	XEQ
$QADD	....	COMM	...	....	....	NUC	....	RDR	....	...	...	XEQ
$QGET	....	....	...	....	MISC	NUC	PRPU	...	....	...	...	XEQ
$QJITLOC	....	COMM	CON	....	....	NUC	....	...	....	...	...	...
$QLOC	....	COMM	...	....	....	NUC	PRPU	RDR	....	...	...	...
$QPUT	....	COMM	CON	....	....	NUC	PRPU	RDR	....	...	...	XEQ
$QREM	....	COMM	...	INIT	MISC	NUC	....	RDR	....	...	...	XEQ
$QSIZ	....	....	...	....	....	NUC	....	...	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>										
$QUESMFB	....	COMM	...	INIT	MISC	NUC	PRPU	...	RTAM	...	...	...
$RDRPEND	....	COMM	...	INIT	....	NUC	....	...	....	...	...	XEQ
$REMOTE	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
$RESTORE	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
$RJECHEQ	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
$SAVEBEG	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
$SAVEEND	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
$SAVELEN	....	....	...	INIT	MISC	...	....	...	....	...	...	...
$SMFBUSY	ACCT	....	...	....	....	NUC	....	...	....	...	...	...
$SMFFREE	ACCT	....	...	INIT	....	NUC	....	...	....	...	...	...
$ST	....	....	CON	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$STATENT	....	....	...	....	....	...	....	...	....	...	...	XEQ
$STATUS	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
$STIMER	....	....	...	....	MISC	NUC	....	...	RTAM	...	...	...
$SVCRSET	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$SYSEXIT	ACCT	COMM	CON	....	....	NUC	....	...	....	...	...	XEQ
$TAPE	....	....	...	....	....	...	....	RDR	....	...	...	...
$TEDADDR	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$TIMEARG	....	....	CON	....	....	...	....	...	....	...	...	...
$TIMENT	....	....	CON	....	....	...	....	...	....	...	...	...
$TP	....	COMM	...	....	....	NUC	....	RDR	RTAM	...	...	...
$TPBPOOL	....	....	...	INIT	....	NUC	....	...	....	...	...	...
$TRACK	....	....	...	....	....	NUC	....	RDR	....	...	...	XEQ
$TRIVIA	....	....	...	....	MISC	NUC	PRPU	RDR	....	...	...	XEQ
$TTIMER	....	....	...	....	MISC	NUC	....	...	....	...	...	...
$UR	....	....	...	....	MISC	NUC	PRPU	RDR	....	...	...	...
$VERSION	....	....	...	INIT	....	...	....	...	....	...	...	...
$WAIT	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$WTO	....	COMM	...	....	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
$XEQ	....	COMM	CON	INIT	....	...	....	RDR	....	...	...	...
$XEQACT	....	....	...	....	....	...	....	...	....	...	...	XEQ
$XFRHASP	....	....	...	....	....	...	PRPU	...	....	...	...	...
$XFRSTPT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
$XSMFENT	....	....	...	....	....	...	....	...	....	...	...	XEQ
BASE1	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
BASE2	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
BASE3	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
BIT0	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT1	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT2	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT3	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT4	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT5	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT6	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BIT7	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
BUFCHAIN	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	...
BUFDCT	....	....	...	....	....	NUC	....	...	RTAM	...	...	XEQ
BUFDDB	....	....	...	....	....	...	....	...	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
BUFDSECT	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
BUFECBCC	....	....	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
BUFEWF	....	....	...	....	....	NUC	....	...	....	...	...	...
BUFSTART	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
BUFTYPE	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	...
CMBCHAIN	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	XEQ
CMBCLASS	....	....	CON	....	....	NUC	....	...	RTAM	...	...	...
CMBCONS	....	....	CON	....	....	...	....	RDR	RTAM	...	...	XEQ
CMBDOMID	....	COMM	CON	....	....	NUC	....	...	....	...	...	...
CMBDSECT	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	XEQ
CMBEND	....	....	CON	....	....	...	....	...	....	...	...	...
CMBFLAGS	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	XEQ
CMBFLD	....	....	CON	....	....	...	....	...	....	...	...	...
CMBJOBNO	....	COMM	CON	....	....	...	....	...	....	...	...	XEQ
CMBMARK	....	....	CON	....	....	...	....	...	....	...	...	...
CMBMSG	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	XEQ
CMBMSGL	....	COMM	CON	....	....	NUC	....	RDR	RTAM	...	...	XEQ
CMBTEXT	....	....	CON	....	....	...	....	...	....	...	...	XEQ
CVTCUCB	....	....	CON	....	....	...	....	...	....	...	...	...
CVTDSECT	....	COMM	CON	INIT	....	NUC	....	RDR	....	SVC	WTR	XEQ
CVTHEAD	....	....	CON	....	....	NUC	....	...	....	...	...	...
CVTHJES	....	....	...	....	....	...	....	...	....	SVC	...	...
CVTILK2	....	COMM	...	INIT	....	...	....	...	....	...	...	...
CVTJOB	....	....	...	....	....	...	....	...	....	...	WTR	...
CVTPCNVT	....	....	...	....	....	...	....	...	....	...	WTR	...
CVTPTP	....	COMM	CON	INIT	....	NUC	....	...	....	...	...	XEQ
CVTSEGA	....	....	...	....	....	...	....	...	....	...	...	XEQ
CVTSMCA	....	....	...	....	....	...	....	RDR	....	...	...	...
CVTTCBP	....	....	CON	INIT	....	...	....	...	....	...	WTR	XEQ
CVTTSCVT	....	....	CON	....	....	...	....	...	....	...	...	...
CVTTSRDY	....	....	CON	....	....	...	....	...	....	...	...	...
CVTXAPG	....	....	...	INIT	....	...	....	...	....	...	...	...
CVTXINT1	....	COMM	CON	INIT	....	NUC	....	RDR	....	SVC	WTR	XEQ
CVTXINT2	....	COMM	CON	INIT	....	NUC	....	RDR	....	SVC	WTR	XEQ
CVTZDIAG	....	....	...	INIT	....	...	....	...	....	...	...	...
CVTOPTO	....	....	CON	....	....	NUC	....	...	....	SVC	...	XEQ
CVTOSCR1	....	....	...	INIT	....	...	....	...	....	...	WTR	...
CVT4MS1	....	COMM	CON	INIT	....	NUC	....	RDR	....	SVC	WTR	XEQ
CVT6DAT	....	COMM	CON	INIT	....	NUC	....	RDR	....	SVC	WTR	XEQ
DCBDEBAD	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	...
DCBDSECT	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
DCBFDAD	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
DCBIFLGS	....	....	...	INIT	....	NUC	....	...	....	...	...	XEQ
DCBMACRF	....	....	...	INIT	....	...	....	...	....	...	...	...
DCBMACR1	....	....	...	....	....	...	....	...	....	...	...	XEQ
DCBMRAPG	....	....	...	INIT	....	...	....	...	....	...	...	...
DCBMRECP	....	....	...	INIT	....	...	....	...	....	...	...	XEQ
DCBMRFE	....	....	...	....	....	...	....	...	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
DCBOPTCD	....	....	...	....	....	...	....	...	....	...	...	XEQ
DCBSSID	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
DCBTIOT	....	....	...	INIT	....	...	....	...	....	...	...	XEQ
DCBWTOID	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
DCTABORT	....	....	...	....	....	NUC	PRPU	...	RTAM	...	...	...
DCTBKSP	....	COMM	...	....	....	...	PRPU	...	RTAM	...	...	...
DCTBUFAD	....	....	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
DCTBUFCT	....	....	...	....	....	NUC	PRPU	RDR	RTAM	...	...	...
DCTCHAIN	....	COMM	...	INIT	....	NUC	....	RDR	RTAM	...	...	XEQ
DCTCLASS	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTDA	....	COMM	...	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
DCTDCB	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	...
DCTDCTE	....	COMM	...	INIT	....	NUC	PRPU	...	....	...	...	...
DCTDELET	....	COMM	...	....	....	...	PRPU	RDR	RTAM	...	...	...
DCTDEVN	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	...
DCTDEVTP	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	...
DCTDRAIN	....	COMM	...	INIT	....	NUC	....	...	RTAM	...	...	XEQ
DCTDSECT	....	COMM	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
DCTECHAR	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTECM	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTECPCD	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTECPYG	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTECPYM	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTECPYN	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTEDCB	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTEDCT	....	COMM	...	INIT	....	NUC	PRPU	...	....	...	...	...
DCTEDSEC	....	COMM	...	INIT	....	NUC	PRPU	...	....	...	...	...
DCTEEND	....	COMM	...	INIT	....	NUC	PRPU	...	....	...	...	...
DCTEFCB	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
DCTEFLAG	....	COMM	...	INIT	....	NUC	PRPU	...	....	...	...	...
DCTEFLCT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTEFLOP	....	COMM	...	....	....	...	....	...	....	...	...	...
DCTEFRMN	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTEJAM	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
DCTEJAMC	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
DCTEJECT	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTEMARK	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTEND	....	....	...	....	....	...	....	...	....	...	...	XEQ
DCTEOF	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
DCTERST	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTESIZ	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTESPL1	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
DCTESPL2	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
DCTESTCN	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTETRC	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTETX	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTEUCBX	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTEWF	....	....	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>										
DCTFCB	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTFLAGS	....	COMM	...	....	....	...	PRPU	RDR	RTAM	...	...	...
DCTFORMS	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTHOLD	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
DCTHOLDJ	....	COMM	...	....	....	...	....	RDR	....	...	...	...
DCTINR	....	COMM	...	INIT	....	...	....	RDR	....	...	...	...
DCTINUSE	....	COMM	...	INIT	....	NUC	....	...	RTAM	...	...	XEQ
DCTIOTYP	....	COMM	...	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
DCTLEASE	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTLNE	....	COMM	...	INIT	....	NUC	PRPU	...	RTAM	...	...	...
DCTLOGAL	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTNO	....	COMM	...	INIT	....	...	PRPU	RDR	RTAM	...	...	...
DCTOLAY	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
DCTOTC	....	....	...	INIT	....	NUC	....	...	....	...	...	...
DCTOTT	....	....	...	INIT	....	NUC	....	...	....	...	...	...
DCTPASCI	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPBLK	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPBSC	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPBUF	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTPCE	....	COMM	...	INIT	....	NUC	....	...	RTAM	...	...	XEQ
DCTPCODE	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPCON	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTPFULL	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPHALF	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPHARD	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPHASP	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTPLINE	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPMRF	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPOST	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
DCTPPRES	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPPSW	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPPSWC	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPPSWF	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPPSWI	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTPPSWO	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPPSWQ	....	COMM	...	....	....	...	....	...	....	...	...	...
DCTPPSWS	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTPPSWT	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPPSWU	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTPRINC	....	....	...	....	....	...	....	RDR	....	...	...	...
DCTPRINT	....	....	...	INIT	....	...	....	RDR	....	...	...	...
DCTPRLEN	....	....	...	INIT	....	...	....	RDR	RTAM	...	...	...
DCTPRLIM	....	....	...	....	....	...	....	RDR	....	...	...	...
DCTPROG	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPRT	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPSTAT	....	COMM	...	INIT	....	NUC	PRPU	...	RTAM	...	...	...
DCTPSYS3	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPTAB	....	....	...	INIT	....	...	....	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
DCTPTRSP	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTPUN	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
DCTPUNCH	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPVAR	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTPWIDE	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTP1130	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTP20	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTP20S2	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
DCTP2770	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTP360	....	....	...	INIT	....	...	....	...	....	...	...	...
DCTRCON	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTRDR	....	COMM	...	INIT	....	...	....	RDR	....	...	...	...
DCTREAD	....	....	...	INIT	MISC	NUC	PRPU	...	....	...	...	XEQ
DCTREJDV	....	COMM	CON	....	....	...	....	...	....	...	...	...
DCTREJJB	....	COMM	CON	....	....	...	....	...	....	...	...	...
DCTREJSY	....	COMM	CON	....	....	...	....	...	....	...	...	...
DCTRJR	....	COMM	...	INIT	....	...	....	RDR	RTAM	...	...	...
DCTRPR	....	....	...	INIT	....	...	PRPU	...	RTAM	...	...	...
DCTRPT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTRPU	....	....	...	INIT	....	...	PRPU	...	RTAM	...	...	...
DCTRSTRT	....	COMM	...	....	....	...	PRPU	RDR	RTAM	...	...	...
DCTSEEK	....	....	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
DCTSINON	....	COMM	...	INIT	....	...	....	...	RTAM	...	...	...
DCTSOFF	....	....	...	....	....	...	....	...	RTAM	...	...	...
DCTSPACE	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTSTAT	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
DCTSTOP	....	COMM	...	....	....	...	PRPU	RDR	RTAM	...	...	...
DCTUCS	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DCTWORK	....	....	...	....	....	...	PRPU	...	....	...	...	...
DCTWRITE	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
DC1FCB	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC1FLAG1	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC1FRMNR	....	COMM	...	....	....	...	....	...	....	...	...	...
DC1IMAGE	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC1MODPT	....	COMM	...	....	....	...	....	...	....	...	...	...
DC1TRC	....	COMM	...	....	....	...	....	...	....	...	...	...
DC1XLAT1	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2CPYNR	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2DCBA	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2FCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2FLAG1	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2FLAG2	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2FRMNR	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2IMAGE	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2MODPT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2STCNR	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2TRC	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
DC2XLAT1	....	....	...	....	....	...	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
DC2XLAT2	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2XLAT3	....	....	...	....	....	...	PRPU	...	....	...	...	...
DC2XLAT4	....	....	...	....	....	...	PRPU	...	....	...	...	...
DDBCHAIN	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBCOUNT	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBDDNR	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBDSECT	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBEND	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
DDBLNG	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBPBUF	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBPCE	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBSBUF	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBSTAT1	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBSTAT2	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBTTR	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBTYPE	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBUFPTR	....	....	...	....	....	...	....	...	....	...	...	XEQ
DDBUNIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
DEBAPPAD	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBDCBAD	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
DEBDSECT	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	...
DEBENDCC	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBENDHH	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBEXSCL	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBNMEXT	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBNUMTR	....	....	...	INIT	....	...	....	...	....	...	...	...
DEBTCBAD	....	COMM	...	INIT	....	...	....	...	....	...	...	...
DEBUCBAD	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	...
E	....	COMM	CON	INIT	....	NUC	....	RDR	RTAM	...	WTR	XEQ
H	....	COMM	CON	....	....	NUC	PRPU	RDR	....	...	...	...
HCTDSECT	ACCT	COMM	CON	INIT	MISC	...	PRPU	RDR	RTAM	SVC	...	XEQ
HDBDSKEY	....	....	...	....	....	...	PRPU	RDR	RTAM	...	...	XEQ
HDBNXTRK	....	....	...	....	....	...	PRPU	RDR	RTAM	...	...	XEQ
HDBSTART	....	....	...	....	....	...	PRPU	RDR	RTAM	...	...	XEQ
ICHAIN	....	....	...	....	....	NUC	....	...	....	...	...	...
IHADCB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
IOBCCW1	....	....	...	INIT	....	NUC	PRPU	...	RTAM	...	...	...
IOBCCW2	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
IOBCCW3	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
IOBCCW4	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	...
IOBCCW5	....	....	...	....	....	...	....	...	RTAM	...	...	...
IOBCCW6	....	....	...	....	....	...	....	...	RTAM	...	...	...
IOBCCW7	....	....	...	....	....	...	....	...	RTAM	...	...	...
IOBCCW8	....	....	...	....	....	...	....	...	RTAM	...	...	...
IOBCSW	....	....	...	....	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
IOBDCBPT	....	....	...	INIT	....	NUC	....	...	....	...	WTR	XEQ
IOBECBCC	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	XEQ
IOBECBPT	....	....	...	INIT	....	...	....	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
IOBERRCT	....	....	...	....	....	NUC	....	...	....	...	...	...
IOBFLAG1	....	....	...	INIT	....	NUC	....	...	....	...	...	XEQ
IOBFLAG2	....	....	...	....	....	NUC	....	...	....	...	...	...
IOBRESTR	....	....	...	....	....	...	....	...	RTAM	...	...	XEQ
IOBSEEK	....	....	...	INIT	....	NUC	....	...	....	...	WTR	...
IOBSENS0	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
IOBSENS1	....	....	...	....	....	...	....	...	RTAM	...	...	...
IOBSIOCC	....	....	...	....	....	NUC	....	RDR	RTAM	...	...	...
IOBSTART	....	....	...	INIT	....	NUC	PRPU	RDR	RTAM	...	WTR	XEQ
IOBXTENT	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	...
IOTCYMAP	....	....	...	INIT	MISC	...	....	RDR	....	...	...	XEQ
IOTCYMXM	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
IOTDSECT	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
IOTFLAGS	....	....	...	....	....	...	....	...	....	...	...	XEQ
IOTIOT	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
IOTIOTTR	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
IOTJCTTR	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
IOTPDDB	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
IOTPDDBP	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
IOTTRACK	....	....	...	INIT	....	...	....	RDR	....	...	...	XEQ
IOTWRITE	....	....	...	....	....	...	....	...	....	...	...	XEQ
IPOST	....	....	...	....	MISC	NUC	....	...	RTAM	...	...	...
ITIME	....	....	...	....	MISC	NUC	....	...	RTAM	...	...	...
JCT	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
JCTACCTN	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTCARDS	....	....	...	....	MISC	...	PRPU	RDR	....	...	...	...
JCTCPUID	....	....	...	....	MISC	...	....	RDR	....	...	...	XEQ
JCTCPYCT	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
JCTCYSAV	....	....	...	INIT	MISC	...	....	RDR	....	...	...	XEQ
JCTDSECT	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
JCTDSKEY	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
JCTEND	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
JCTESOUT	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTESTLN	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTESTPU	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTETIME	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTFORMS	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
JCTINDC	....	....	...	....	MISC	...	....	RDR	....	...	...	...
JCTINDEV	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTINJCT	....	....	...	INIT	....	...	....	RDR	....	...	...	XEQ
JCTIOTTR	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
JCTJBOPT	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
JCTJCLAS	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTJMR	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
JCTJMRJN	....	....	...	....	MISC	...	PRPU	RDR	....	...	...	...
JCTJMRUX	....	....	...	....	MISC	...	....	...	....	...	...	...
JCTJNAME	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
JCTJOBEB	....	COMM	CON	....	....	NUC	PRPU	RDR	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
JCTJOBIN	....	....	...	....	MISC	...	....	RDR	....	...	...	...
JCTJQE	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
JCTLINCT	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
JCTLINES	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
JCTMCLAS	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
JCTNOLOG	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTODTOF	....	....	...	....	MISC	...	....	...	....	...	...	...
JCTOPRIO	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTOUTOF	....	....	...	....	....	...	PRPU	...	....	...	...	...
JCTOUTON	....	....	...	....	....	...	PRPU	...	....	...	...	...
JCTPAGCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
JCTPNAME	....	COMM	...	....	MISC	NUC	PRPU	RDR	....	...	...	...
JCTPRICD	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTPRIO	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTPROUT	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTPRTCT	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
JCTPUNCH	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
JCTPUNCT	....	....	...	....	MISC	...	PRPU	...	....	...	...	...
JCTPUOUT	....	....	...	....	MISC	...	....	RDR	....	...	...	...
JCTPURGE	....	....	...	....	MISC	...	....	...	....	...	...	...
JCTRDR	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTRDROF	....	....	...	....	MISC	...	....	RDR	....	...	...	XEQ
JCTRDRON	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
JCTROOMN	....	COMM	...	....	....	NUC	PRPU	RDR	....	...	...	...
JCTROUTE	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTSETUP	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTTHOLD	....	....	...	....	....	...	....	RDR	....	...	...	...
JCTUJVP	....	....	...	....	....	...	....	...	....	...	...	XEQ
JCTUSEID	....	....	...	....	MISC	...	PRPU	RDR	....	...	...	...
JCTWORK	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
JCTXBACH	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
JCTXDTOF	....	....	...	....	....	...	....	...	....	...	...	XEQ
JCTXDTON	....	....	...	....	....	...	....	...	....	...	...	XEQ
JCTXEQOF	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
JCTXEQON	....	....	...	....	MISC	...	PRPU	...	....	...	...	XEQ
JCTXOUT	....	....	...	....	....	...	....	...	....	...	...	XEQ
JITJNAME	....	COMM	CON	....	....	...	....	RDR	....	...	...	...
JMPCHAIN	ACCT	....	...	....	MISC	...	....	...	....	...	...	...
JOEACTPR	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
JOEACTPU	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEBURST	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOECHAR	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOECKARG	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOECKFLG	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOECKPT	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
JOECPU	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEDEST	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEDSECT	....	COMM	...	INIT	MISC	NUC	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
JOEFCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEFLAG	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
JOEFLASH	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOEFORM	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOEJQE	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
JOENEXT	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
JOERECCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEROUT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOESEC	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOESIZE	....	COMM	...	INIT	MISC	NUC	PRPU	...	....	...	...	...
JOEUCS	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEUSE	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOEWTRID	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOE38FLG	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
JOTCHRQ	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
JOTCKPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTCLSQ	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
JOTDSECT	....	COMM	...	INIT	MISC	...	....	...	....	...	...	...
JOTFREC	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTFREL	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTFREQ	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTJOBNO	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTJOES	....	....	...	....	....	...	PRPU	...	....	...	...	...
JOTSIZE	....	....	...	INIT	MISC	...	....	...	....	...	...	...
JSCBTJID	....	....	CON	....	....	...	....	...	....	...	...	...
JSCDSECT	....	....	CON	....	....	...	....	...	....	...	...	XEQ
JSCHPCE	....	....	...	....	....	...	....	...	....	...	...	XEQ
L	....	COMM	...	....	....	...	PRPU	...	RTAM	...	...	...
LCBACK	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
LCBMCB	....	....	...	....	....	...	....	...	RTAM	...	...	...
LCBRCB	....	....	...	....	....	...	....	...	RTAM	...	...	...
LINK	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
M	....	COMM	...	....	....	...	....	...	....	...	...	...
MDCTATTN	....	....	...	....	....	...	....	...	RTAM	...	...	...
MDCTCODE	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
MDCTDCT	....	COMM	...	INIT	....	...	....	...	RTAM	...	...	...
MDCTERCT	....	....	...	....	....	...	....	...	RTAM	...	...	...
MDCTFCS	....	....	...	INIT	....	...	PRPU	...	RTAM	...	...	...
MDCTOBUF	....	....	...	....	....	...	....	...	RTAM	...	...	...
MDCTOPCT	....	....	...	....	....	...	....	...	RTAM	...	...	...
MDCTPSWD	....	COMM	...	....	....	...	....	...	RTAM	...	...	...
MDCTRCB	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
MDCTRSEQ	....	....	...	....	....	...	....	...	RTAM	...	...	...
MDCTTSEQ	....	....	...	....	....	...	....	...	RTAM	...	...	...
MHASPECF	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
MSABITS	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
MSADSECT	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
MSAMTTR	....	....	...	INIT	....	...	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
MSARPTRK	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
NE	....	COMM	CON	INIT	....	NUC	....	RDR	RTAM	SVC	...	...
NH	....	COMM	...	....	....	NUC	....	...	....	...	...	...
NL	....	COMM	...	INIT	....	...	....	...	....	...	...	...
NO	....	....	...	....	....	...	....	...	....	...	...	XEQ
NP	....	....	CON	INIT	....	...	....	...	....	...	...	XEQ
NZ	....	....	CON	....	....	NUC	PRPU	...	....	...	...	XEQ
O	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
OACEASMO	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OACECHN	....	....	...	....	....	NUC	....	...	....	...	...	...
OACENAME	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OACEOCON	....	....	...	....	....	NUC	....	...	....	...	...	...
OACEPCE	....	....	...	....	....	NUC	....	...	....	...	...	...
OACEPRIO	....	....	...	....	....	NUC	....	...	....	...	...	...
OACEPROG	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
OCRBURST	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRCHAR	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRCODE	....	....	...	....	....	...	....	...	....	...	...	XEQ
OCRCOPY	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRCOPYG	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRDEST1	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRDEST2	....	....	...	....	....	...	....	...	....	...	...	XEQ
OCRDSECT	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCREND	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRFCB	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRFLASH	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRFLCT	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRFORMS	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRINDEX	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRMOD	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRMODTR	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRNUMGR	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRRECNT	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OCRUCS	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
OLAYBUF	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OPCHAR	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPCKPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPCLASS	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPDADCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPDBEND	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPDDB	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPIOT	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPJCTBUF	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPJOBCPY	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPJOBFRM	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPJQE	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPMSGCLS	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPRECCT	....	....	...	....	....	...	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
OPTIMEON	....	....	...	....	....	...	PRPU	...	....	...	...	...
OPWORK	....	....	...	....	....	...	PRPU	...	....	...	...	...
OTBADDR	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OTBCALLS	....	....	...	....	....	NUC	....	...	....	...	...	...
OTBDSECT	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OTBLODS	....	....	...	....	....	NUC	....	...	....	...	...	...
OTBPRIO	....	....	...	....	....	NUC	....	...	....	...	...	...
OTBSIZE	....	....	...	INIT	....	NUC	....	...	....	...	...	...
OTBTRAK	....	....	...	....	....	NUC	....	...	....	...	...	...
OUTEMP	....	....	...	....	....	...	PRPU	...	....	...	...	...
OUTWKSIZ	....	....	...	....	....	NUC	....	...	....	...	...	...
P	....	....	...	INIT	....	...	....	...	....	...	...	...
PBSPGCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PBSPTBL	....	....	...	....	....	...	PRPU	...	....	...	...	...
PBUFOPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PBUFSAVE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCCWCHN	....	COMM	...	....	....	NUC	PRPU	...	....	...	...	...
PCCWEND	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCCWPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEASYID	....	....	...	INIT	....	NUC	....	...	....	...	...	...
PCEBASE2	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
PCEBASE3	....	....	...	....	....	NUC	....	...	RTAM	...	...	...
PCECKPID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCECONID	....	....	...	INIT	....	NUC	....	...	....	...	...	...
PCEDSECT	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
PCEEJRCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEEWP	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
PCEFCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEFORM	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEGPRID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEID	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
PCEINRID	....	....	...	....	....	NUC	....	RDR	....	...	...	...
PCEIOTTR	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEJCT	....	....	CON	INIT	....	...	....	...	....	...	...	XEQ
PCEJMTTR	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCELCLID	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
PCELINK	....	COMM	...	INIT	....	NUC	PRPU	...	RTAM	...	...	...
PCEMLMID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCENEXT	....	COMM	...	INIT	....	NUC	....	...	....	...	...	XEQ
PCEOCON	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEOPCE	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEOPRIO	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEORTRN	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEOUTID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEPRGID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEPRSID	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
PCEPRTID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCEPUNID	....	....	...	....	....	NUC	....	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
PCEPUSID	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
PCERDRID	....	COMM	...	....	....	NUC	....	...	....	...	...	...
PCERJEID	....	....	...	....	....	NUC	PRPU	RDR	....	...	...	...
PCER0	....	....	...	....	....	NUC	PRPU	...	RTAM	...	...	...
PCER1	....	....	...	....	....	NUC	PRPU	RDR	....	...	...	...
PCER15	....	....	...	INIT	....	NUC	PRPU	...	RTAM	...	...	...
PCESAVEA	....	....	...	....	....	NUC	PRPU	...	....	...	...	XEQ
PCETHWID	....	....	...	....	....	NUC	....	...	....	...	...	...
PCETIMID	....	....	...	INIT	....	NUC	....	...	....	...	...	...
PCEUCSB	....	....	...	....	....	...	PRPU	...	....	...	...	...
PCEWA	....	COMM	...	....	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
PCEWB	....	....	...	....	....	...	....	RDR	....	...	...	...
PCEWC	....	....	...	....	....	...	....	RDR	....	...	...	...
PCEWF	....	....	...	INIT	....	...	....	...	....	...	...	...
PCEWORK	....	COMM	CON	....	....	NUC	PRPU	RDR	....	...	...	XEQ
PCEXEQID	....	....	...	....	....	NUC	....	...	....	...	...	XEQ
PCEXFER	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
PCHJOE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDADCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDATATRC	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBBASLN	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
PDBBURST	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDBCH1	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCH2	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCH3	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCH4	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCLASS	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBCOPYS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCPU	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBCPYG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBDEST	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBDSECT	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBFCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBFLAG1	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBFLAG2	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBFLAG3	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBFLASH	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBFLCG	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFLCM	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFLIM	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDBFLX1	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFLX2	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFLX3	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFLX4	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBFORMS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBINDEX	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBLENG	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBMISC1	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
PDBMISC2	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBMOD	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBMTTR	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDBOPCDJ	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDBRECCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBSEC	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBSTNR	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDBUCS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDBWTRID	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDB1FLG2	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDB1LOG	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
PDB1MDES	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB1NULL	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDB2FCB	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB2FORM	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB2MSC1	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB2MSC2	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB2RECC	....	....	...	....	....	...	....	...	....	...	...	XEQ
PDB2UCS	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDB3800	....	....	...	....	....	...	PRPU	...	....	...	...	XEQ
PDB3800W	....	....	...	....	....	...	PRPU	RDR	....	...	...	XEQ
PDCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDCTFLAG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDDBDISP	....	....	...	....	....	...	PRPU	...	....	...	...	...
PDDBPGCT	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
PDDBSKIP	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
PDEVTYPE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PITBCLAS	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITBJST	....	....	...	....	....	...	....	...	....	...	...	XEQ
PITBUCBA	....	....	...	....	....	...	....	...	....	...	...	XEQ
PITBUNIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
PITBUSY	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITCLASS	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
PITHOLDA	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITHOLDT	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITICLAS	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
PITIDLE	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITLAST	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
PITLNGTH	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
PITPATID	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
PITPRIO	....	....	...	....	....	...	....	...	....	...	...	XEQ
PITSTAT	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
PJOB	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
PJUSTSEP	....	....	...	....	....	...	PRPU	...	....	...	...	...
PLSAVE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PLSAVE2	....	....	...	....	....	...	PRPU	...	....	...	...	...
PLSAVE3	....	....	...	....	....	...	PRPU	...	....	...	...	...
PLSAVE4	....	....	...	....	....	...	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
PMESSAGE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPBURST	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPCLRP	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPCPYG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPDSCPY	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPFLAG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPFLASH	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPFLCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJCARDS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJDSKEY	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJJNAME	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJJOBEB	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJLINES	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJNDS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJOBFRM	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJPNAME	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJPUNCH	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJROOMN	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJXEQOF	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPJXEQON	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPLNCDCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPMODPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPMSGCLS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPRCFLAG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPRCPYCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPRECCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPREXMIT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPSTCNR	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPTRC	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPXLAT1	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPXLAT2	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPXLAT3	....	....	...	....	....	...	PRPU	...	....	...	...	...
PPXLAT4	....	....	...	....	....	...	PRPU	...	....	...	...	...
PP38RS	....	....	...	....	....	...	PRPU	...	....	...	...	...
PP3800FL	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCCPYCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCFLAGS	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
PRCHKJOB	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
PRCHKPTE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCHKUSE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCJOBNO	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCKJOE	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
PRCLINCT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRCSIZE	....	....	...	INIT	MISC	...	PRPU	...	....	...	...	...
PRINDEX	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRLINECT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRPAGECT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PRRPCEWS	....	....	...	....	....	NUC	....	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
PRTPCEWS	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
PSEPPG	....	....	...	....	....	...	PRPU	...	....	...	...	...
PSMF6BID	....	....	...	....	....	...	PRPU	...	....	...	...	...
PSMF6JBN	....	....	...	....	....	...	PRPU	...	....	...	...	...
PSMF6NLR	....	....	...	....	....	...	PRPU	...	....	...	...	...
PSMF6PGE	....	....	...	....	....	...	PRPU	...	....	...	...	...
PSMF6UIF	....	....	...	....	....	...	PRPU	...	....	...	...	...
PTIMEON	....	....	...	....	....	...	PRPU	...	....	...	...	...
PUERRPT	....	....	...	....	....	...	PRPU	...	....	...	...	...
PUNPCEWS	....	....	...	....	....	NUC	PRPU	...	....	...	...	...
PURPCEWS	....	....	...	....	....	NUC	....	...	....	...	...	...
PWKJOE	....	....	...	....	....	...	PRPU	...	....	...	...	...
QENTBY	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
QUECHAIN	....	COMM	CON	INIT	MISC	NUC	....	...	....	...	...	XEQ
QUEFLAGS	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	....	...	...	XEQ
QUEHOLDA	....	COMM	CON	....	....	NUC	PRPU	...	....	...	...	...
QUEHOLD1	....	COMM	CON	....	....	NUC	PRPU	RDR	....	...	...	...
QUEHOLD2	....	COMM	CON	INIT	....	NUC	....	...	....	...	...	XEQ
QUEJCTSW	....	....	...	INIT	....	...	PRPU	...	....	...	...	...
QUEJOBNO	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	...	...	...
QUEJOECT	....	....	...	....	....	...	PRPU	...	....	...	...	...
QUEOPCAN	....	COMM	CON	....	MISC	...	....	...	....	...	...	...
QUEOPRTE	....	COMM	...	....	....	...	PRPU	...	....	...	...	...
QUEPRIO	....	COMM	...	....	MISC	NUC	PRPU	RDR	....	...	...	XEQ
QUEPRTRT	....	COMM	...	....	....	NUC	PRPU	RDR	....	...	...	...
QUEPUNRT	....	COMM	...	....	....	NUC	PRPU	RDR	....	...	...	...
QUEPURGE	....	COMM	CON	INIT	....	NUC	PRPU	...	....	...	...	XEQ
QUETRAK	....	....	...	INIT	MISC	...	PRPU	RDR	....	...	...	XEQ
QUETYPE	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	...	...	XEQ
RBDSECT	....	....	CON	....	....	NUC	....	...	....	...	...	...
RBIEND	....	....	...	....	....	...	....	RDR	....	...	...	...
RBINTCOD	....	....	CON	....	....	...	....	...	....	...	...	...
RBLINK	....	....	CON	....	....	NUC	....	...	....	...	...	...
RBOEND	....	....	...	....	....	...	....	RDR	....	...	...	...
RBONEXT	....	....	...	....	....	...	....	RDR	....	...	...	...
RBPREFIX	....	....	CON	....	....	...	....	...	....	...	...	...
RBSTAB	....	....	CON	....	....	...	....	...	....	...	...	...
RCARDID	....	....	...	....	....	...	....	RDR	....	...	...	...
RDADCT	....	....	...	....	....	...	....	RDR	....	...	...	...
RDRDCT	....	....	...	....	....	...	....	RDR	....	...	...	...
RDRDLM	....	....	...	....	....	...	....	RDR	....	...	...	...
RDRPCEWS	....	....	...	....	....	NUC	....	...	....	...	...	...
RDRSW	....	....	...	....	....	...	....	RDR	....	...	...	...
RIDBUSY	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
RIDDATA	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
RIDFLAGS	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
RIDPOST	....	....	...	....	....	...	....	RDR	....	...	...	XEQ
RIDTCB	....	....	...	....	....	...	....	RDR	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
RIDUCB	....	....	...	INIT	....	...	....	...	....	...	...	XEQ
RJCLTRAK	....	....	...	....	....	...	....	RDR	....	...	...	...
RJEPCEWS	....	....	...	....	....	NUC	....	...	....	...	...	...
RJOB	....	COMM	...	....	....	...	....	RDR	....	...	...	...
RLSAVE1	....	....	...	....	....	...	....	RDR	....	...	...	...
RLSAVE2	....	....	...	....	....	...	....	RDR	....	...	...	...
RLSAVE3	....	....	...	....	....	...	....	RDR	....	...	...	...
RMESSAGE	....	....	...	....	....	...	....	RDR	....	...	...	...
RMSGCLAS	....	....	...	....	....	...	....	RDR	....	...	...	...
RPRIORTY	....	....	...	....	....	...	....	RDR	....	...	...	...
RSAVE1	....	....	...	....	....	...	....	RDR	....	...	...	...
RSAVE2	....	....	...	....	....	...	....	RDR	....	...	...	...
RTPCARD	....	....	...	....	....	...	....	RDR	....	...	...	...
R0	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R1	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R10	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R11	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R12	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R13	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R14	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R15	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R2	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R3	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R4	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R5	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R6	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R7	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R8	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	WTR	XEQ
R9	....	....	CON	INIT	....	...	....	...	....	...	...	XEQ
SAVE	....	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	SVC	...	XEQ
SMFCBER	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMFCHAIN	ACCT	....	...	....	....	...	....	...	....	...	...	...
SMFDSECT	ACCT	COMM	...	INIT	MISC	...	PRPU	...	RTAM	...	...	...
SMFDSER	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMFHDRTY	....	COMM	...	INIT	MISC	...	PRPU	...	RTAM	...	...	...
SMFHSPID	....	COMM	...	INIT	MISC	...	....	...	RTAM	...	...	...
SMFJMR	ACCT	....	...	....	MISC	...	....	...	....	...	...	...
SMFJMRTP	ACCT	....	...	....	MISC	...	....	...	....	...	...	...
SMFLINEV	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMFOPCAN	....	....	...	....	MISC	...	....	...	....	...	...	...
SMFOUTTP	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMFPRGTP	....	....	...	....	MISC	...	....	...	....	...	...	...
SMFPSETP	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMFPSSTP	....	COMM	...	....	....	...	....	...	....	...	...	...
SMFRDW	ACCT	COMM	...	INIT	MISC	...	PRPU	...	RTAM	...	...	...
SMFRECTP	ACCT	....	...	....	....	...	....	...	....	...	...	...
SMFRMTEV	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMFRSTRT	ACCT	COMM	...	INIT	MISC	...	PRPU	...	RTAM	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
SMFSSETP	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMFSSID	....	COMM	...	INIT	....	...	....	...	RTAM	...	...	...
SMFSSLEN	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
SMFSSSTP	....	....	...	INIT	....	...	....	...	....	...	...	...
SMFSSTRT	ACCT	COMM	...	INIT	MISC	...	PRPU	...	RTAM	...	...	...
SMFTYPE	ACCT	....	...	....	MISC	...	....	...	....	...	...	...
SMF26END	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26ICD	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26IND	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26INF	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26JBN	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26LN1	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26LN2	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26LN3	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26OPD	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26PUR	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26RPT	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26RV1	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26SBS	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26UIF	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF26XST	....	....	...	....	MISC	...	....	...	....	...	...	...
SMF43END	....	....	...	INIT	....	...	....	...	....	...	...	...
SMF43OPT	....	....	...	INIT	....	...	....	...	....	...	...	...
SMF43RV1	....	....	...	INIT	....	...	....	...	....	...	...	...
SMF45END	....	COMM	...	....	....	...	....	...	....	...	...	...
SMF47END	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47EVT	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47LIN	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47LN1	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47LN2	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47MSG	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47PSW	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF47RMT	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF48EVT	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF48LIN	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF48PSW	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF48RMT	....	....	...	....	....	...	....	...	RTAM	...	...	...
SMF6BID	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6CHR	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6CPS	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6END	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6FCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6FLC	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6FLI	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6FMN	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6IOE	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6JBN	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6JNM	....	....	...	....	....	...	PRPU	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
SMF6LN2	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6MID	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6NDS	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6NLR	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6OUT	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6OWC	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6PGE	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6RTE	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6UCS	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6UIF	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF6WST	....	....	...	....	....	...	PRPU	...	....	...	...	...
SMF638E	....	....	...	....	....	...	PRPU	...	....	...	...	...
SPPADFCB	....	....	...	....	....	...	PRPU	...	....	...	...	...
SPPARM	....	....	...	INIT	....	...	....	...	....	...	...	...
SPPBFREQ	....	....	...	INIT	....	...	....	...	....	...	...	...
SPPBTREQ	....	....	...	INIT	....	...	....	...	....	...	...	...
SPPBURST	....	COMM	...	....	....	...	....	...	....	...	...	...
SPPFLAG1	....	....	...	INIT	....	...	....	...	....	...	...	...
SPPINIT	....	....	...	....	....	...	PRPU	...	....	...	...	...
SPPMODI	....	....	...	....	....	...	PRPU	...	....	...	...	...
SRTEALOC	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
SRTEASCI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
SRTEBPRV	....	....	...	INIT	....	...	....	...	....	...	...	...
SRTEBPUB	....	....	...	INIT	....	...	....	...	....	...	...	...
SRTEBSTR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
SRTECHGS	....	COMM	...	INIT	....	...	....	...	....	...	...	...
SRTEONLI	....	COMM	...	INIT	....	...	....	...	....	...	...	...
SRTEPRES	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
SRTERESV	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
SRTESTAB	....	....	...	INIT	....	...	....	...	....	...	...	...
SRTESTAT	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
SRTEUSER	....	COMM	...	INIT	....	...	....	...	....	...	...	...
SRTEVOLI	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
TCBDSECT	....	....	CON	INIT	....	NUC	....	...	....	...	...	XEQ
TCBFLGS	....	....	CON	....	....	...	....	...	....	...	...	XEQ
TCBGRS	....	....	...	....	....	...	....	...	....	...	...	XEQ
TCBHNDSP	....	....	...	....	....	...	....	...	....	...	...	XEQ
TCBJSCB	....	....	CON	....	....	...	....	...	....	...	...	XEQ
TCBJSICB	....	....	CON	....	....	...	....	...	....	...	...	XEQ
TCBLMP	....	....	...	INIT	....	...	....	...	....	...	...	...
TCBLTC	....	....	...	....	....	...	....	...	....	...	...	XEQ
TCBNTC	....	....	...	....	....	...	....	...	....	...	...	XEQ
TCBOTC	....	....	...	....	....	...	....	...	....	...	...	XEQ
TCBRBP	....	....	CON	....	....	NUC	....	...	....	...	...	...
TCBTCB	....	....	CON	....	....	NUC	....	...	....	...	...	...
TCBTIO	....	....	CON	....	....	...	....	...	....	...	...	XEQ
TEDDSECT	....	....	...	INIT	MISC	NUC	....	...	....	...	...	...
TEDSIZ	....	....	...	INIT	....	NUC	....	...	....	...	...	...

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
INCH	....	....	...	....	....	NUC	....	...	....	...	...	...
INMB	....	....	...	INIT	....	NUC	....	...	....	...	...	...
INMD	....	....	...	INIT	....	NUC	....	...	....	...	...	...
INMO	....	....	...	INIT	....	NUC	....	...	....	...	...	...
INRT	....	....	...	INIT	....	...	....	...	....	...	...	...
INTC	....	....	...	INIT	....	NUC	....	...	....	...	...	...
INTG	....	....	...	INIT	....	NUC	....	...	....	...	...	...
IPBFDATA	....	....	...	....	....	...	....	...	RTAM	...	...	...
IPBLCCAD	....	....	...	....	....	...	....	...	RTAM	...	...	...
IPBLCCC	....	....	...	....	....	...	....	...	RTAM	...	...	...
IPBRLCNT	....	....	...	....	....	...	....	...	RTAM	...	...	...
IPBUF	....	....	...	INIT	....	NUC	....	...	RTAM	...	...	...
IPBUFST	....	....	...	INIT	....	...	....	...	RTAM	...	...	...
IRPS	....	....	...	INIT	....	NUC	....	...	....	...	...	...
UCBALOC	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBADI	....	COMM	...	INIT	....	NUC	....	...	....	SVC	...	XEQ
UCBBALB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBJLB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBNUL	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBPRV	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBPUB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBSTR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBBSVL	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBCHA	....	....	...	INIT	....	...	....	...	....	...	...	...
UCBCHGS	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDADI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDBBNR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDCALL	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDDMCT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDEV	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDJBNR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDMCT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDSECT	....	COMM	...	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDSTAB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDSTAT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDUSER	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDVOLI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBDVTOC	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBERADR	....	COMM	...	INIT	....	...	....	...	....	...	...	...
UCBFL1	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBFL2	....	....	...	INIT	....	...	....	...	....	...	...	...
UCBFSCT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBFSEQ	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBHPDV	....	....	...	....	....	...	....	...	....	...	...	XEQ
UCBJBNR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBMDRBF	....	....	...	INIT	....	...	....	...	....	...	...	...
UCBMONT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBNAME	....	COMM	...	INIT	....	NUC	....	...	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
UCBNBRSN	....	....	...	....	....	NUC	....	...	....	...	...	...
UCBNOTRD	....	....	...	....	....	...	PRPU	...	....	...	...	...
UCBONLI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBPRES	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBRESV	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBSTAB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBSTAT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBSYSR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBTBYT2	....	....	...	INIT	....	...	....	...	....	...	...	...
UCBTBYT3	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
UCBTBYT4	....	COMM	...	INIT	....	...	....	...	....	...	...	XEQ
UCBTYP	....	....	...	....	....	...	PRPU	RDR	....	...	...	...
UCBUCS	....	COMM	...	INIT	....	...	....	...	....	...	...	...
UCBUNLD	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBUSER	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBVOLI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	...	XEQ
UCBXTADR	....	COMM	...	INIT	....	...	....	...	....	...	...	...
UCB3COMM	....	....	...	INIT	....	...	....	...	....	...	...	...
UCB3DACC	....	COMM	...	INIT	....	NUC	....	...	....	...	...	...
UCB3UREC	....	....	...	INIT	....	NUC	....	...	....	...	...	...
UCB3800	....	COMM	...	INIT	....	...	PRPU	...	....	...	...	...
WA	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
WB	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
WC	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
WCMBFD	....	COMM	CON	....	....	...	....	...	....	...	...	...
WCMBFE	....	....	...	....	....	...	....	...	....	...	...	XEQ
WCMBFF	....	COMM	CON	....	....	NUC	....	...	....	...	...	...
WCMBFG	....	COMM	CON	....	....	...	....	RDR	RTAM	...	...	...
WCMBFH	....	....	...	....	....	...	....	...	....	...	...	XEQ
WD	ACCT	COMM	...	INIT	MISC	NUC	PRPU	RDR	RTAM	...	...	XEQ
WE	ACCT	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
WF	....	COMM	...	INIT	....	NUC	PRPU	RDR	RTAM	...	...	XEQ
WG	....	....	...	INIT	....	NUC	PRPU	...	RTAM	...	...	XEQ
XALLOCWT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XDUPBIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XEOJBIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XEOJMES	....	....	...	....	....	...	....	...	....	...	...	XEQ
XEQPCEWS	....	....	...	....	....	NUC	....	...	....	...	...	XEQ
XGETIOT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XIOTWREQ	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOCRMES	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOCRMSG	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOUTCDBF	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOUTENT	....	COMM	CON	....	....	NUC	....	...	....	...	...	XEQ
XOUTEST	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOUTTOTL	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOUTTYPE	....	....	...	....	....	...	....	...	....	...	...	XEQ
XOUTXCES	....	....	...	....	....	...	....	...	....	...	...	XEQ

SYMBOL	<======== REFERENCED IN HASP ASSEMBLY MODULE ========>											
XPCEDCT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEDDB	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEIOT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEJOB	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
XPCEJOBN	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEJST	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEOUTC	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEPIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEPRT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCEPUN	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPCESTAT	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
XPCESTEP	....	....	...	....	....	...	....	...	....	...	...	XEQ
XPOSTBIT	....	....	...	....	....	...	....	...	....	...	...	XEQ
XREXREQ	....	COMM	...	....	....	...	....	...	....	...	...	XEQ
XS	....	....	...	....	....	...	....	...	....	...	...	XEQ
XSYNCFLG	....	....	...	....	....	...	....	...	....	...	...	XEQ
XSYNCREQ	....	....	...	....	....	...	....	...	....	...	...	XEQ
Z	....	COMM	CON	INIT	....	NUC	PRPU	RDR	RTAM	SVC	...	XEQ

SYMBOL	<========== REFERENCED IN HASP ASSEMBLY MODULE ==========											
UCBSTAB	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCBSTAT	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCBSYSR	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCBTBYT2	....	....	....	INIT	....	....	....	....	....	....	....	....
UCBTBYT3	....	COMM	....	INIT	....	NUC	....	....	....	....	....	....
UCBTBYT4	....	COMM	....	INIT	....	....	....	....	....	....	....	XEQ
UCBTYP	....	....	....	....	....	....	PRPU	RDR	....	....	....	....
UCBUNLD	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCBUSER	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCBVOLI	....	COMM	CON	INIT	....	NUC	PRPU	RDR	....	SVC	....	XEQ
UCB3COMM	....	....	....	INIT	....	....	....	....	....	....	....	....
UCB3DACC	....	COMM	....	INIT	....	NUC	....	....	....	....	....	....
UCB3UREC	....	....	....	INIT	....	....	....	....	....	....	....	....
WA	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	....	....	XEQ
WB	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	....	....	XEQ
WC	ACCT	COMM	CON	INIT	MISC	NUC	PRPU	RDR	RTAM	....	....	XEQ
WCMBFD	....	COMM	CON	....	....	....	....	....	....	....	....	....
WCMBFE	....	....	....	....	....	....	....	....	....	....	....	XEQ
WCMBFF	....	COMM	CON	....	....	NUC	....	....	....	....	....	....
WCMBFG	....	COMM	CON	....	....	....	....	RDR	RTAM	....	....	....
WCMBFH	....	....	....	....	....	....	....	....	....	....	....	XEQ
WD	ACCT	COMM	....	INIT	MISC	NUC	PRPU	RDR	RTAM	....	....	XEQ
WE	ACCT	COMM	....	INIT	....	NUC	PRPU	RDR	RTAM	....	....	XEQ
WF	....	COMM	....	INIT	....	NUC	PRPU	RDR	RTAM	....	....	XEQ
WG	....	....	....	INIT	....	NUC	PRPU	....	RTAM	....	....	XEQ
XALLOCWT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XDUPBIT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XEOJBIT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XEOJMES	....	....	....	....	....	....	....	....	....	....	....	XEQ
XEQPCEWS	....	....	....	....	....	NUC	....	....	....	....	....	XEQ
XGETIOT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XIOTWREQ	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOCRMES	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOCRMSG	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOUTCDBF	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOUTENT	....	COMM	CON	....	....	NUC	....	....	....	....	....	XEQ
XOUTEST	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOUTTOTL	....	....	....	....	....	....	....	....	....	....	....	XEQ

SYMBOL	<========== REFERENCED IN HASP ASSEMBLY MODULE ==========>											
XOUTTYPE	....	....	....	....	....	....	....	....	....	....	....	XEQ
XOUTXCES	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEDCT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEDDB	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEIOT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEJOB	....	COMM	....	....	....	....	....	....	....	....	....	XEQ
XPCEJOBN	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEJST	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEOUTC	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEPIT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEPRT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCEPUN	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPCESTAT	....	COMM	....	....	....	....	....	....	....	....	....	XEQ
XPCESTEP	....	....	....	....	....	....	....	....	....	....	....	XEQ
XPOSTBIT	....	....	....	....	....	....	....	....	....	....	....	XEQ
XREXREQ	....	COMM	....	....	....	....	....	....	....	....	....	XEQ
XS	....	....	....	....	....	....	....	....	....	....	....	XEQ
XSYNCFLG	....	....	....	....	....	....	....	....	....	....	....	XEQ
XSYNCREQ	....	....	....	....	....	....	....	....	....	....	....	XEQ
Z	....	COMM	CON	INIT	....	NUC	PRPU	RDR	RTAM	SVC	....	XEQ

SECTION 5

HASP

DATA AREAS

HASP BUFFER

The Buffer is a resident control block which consists of three basic parts: (1) a standard OS Input/Output Block (IOB), (2) Buffer control information, and (3) a work space. There are three types of buffers which differ only in the format and length of the three basic parts. These are: (1) the basic HASP Buffer, which is used as a general work area, or to contain control blocks, data blocks, or input card images; (2) the HASP Remote Job Entry (RJE) Buffer, which is used to transmit and receive text blocks to and from RJE terminals; and (3) the HASP Overlay Area, which is used for reading and executing nonresident HASP Overlay routines.

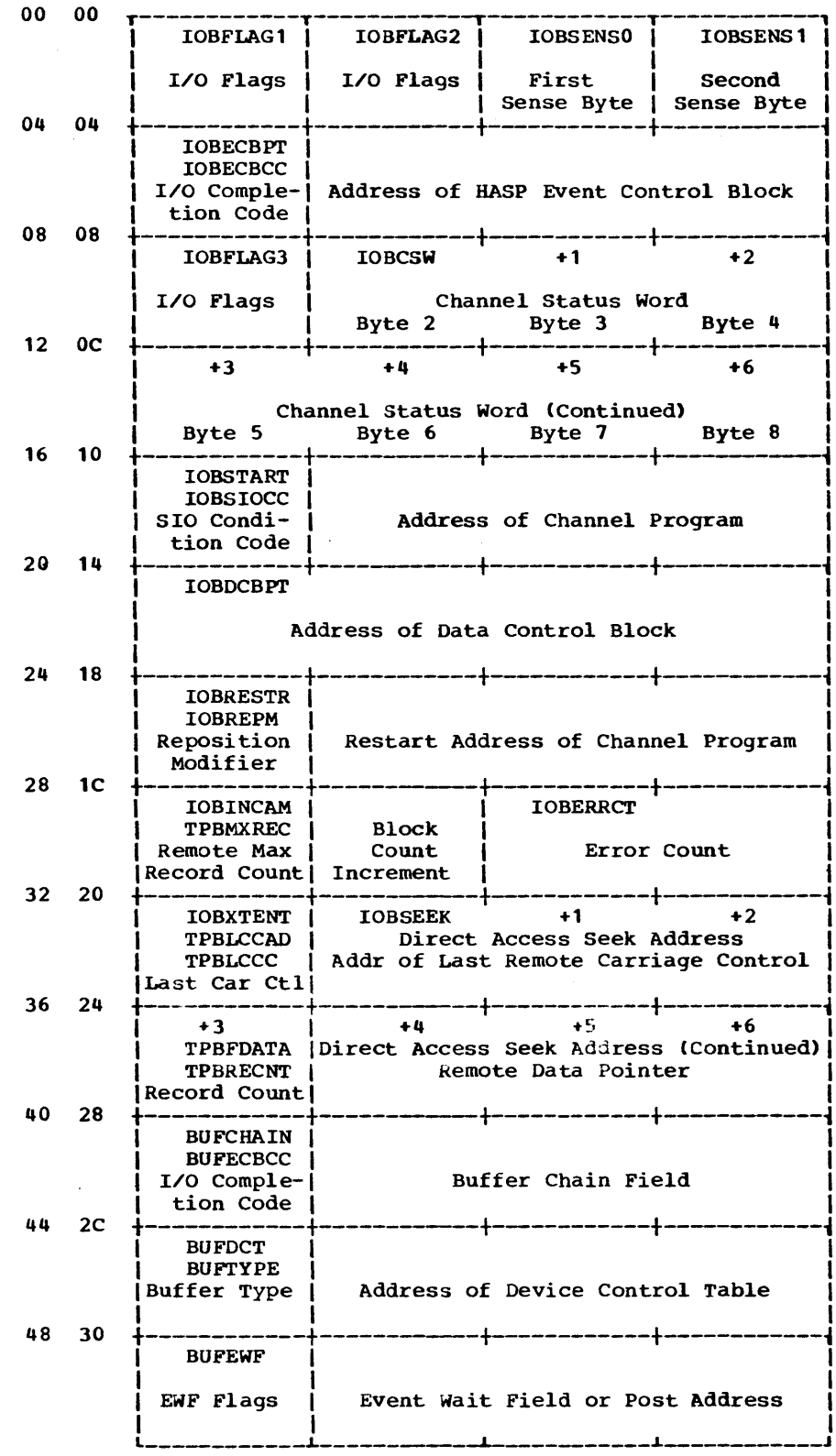
Offset	Bytes and Bits	Field Name	Description
00 (00)	1	IOBFLAG1	I/O flags
01 (01)	1	IOBFLAG2	I/O flags
02 (02)	1	IOBSENS0	First sense byte
03 (03)	1	IOBSENS1	Second sense byte
04 (04)	1	IOBECBCC	I/O completion code
04 (04)	4	IOBECBPT	HASP Event Control Block address
08 (08)	1	IOBFLAG3	I/O flags
09 (09)	7	IOBCSW	Channel status word
16 (10)	1	IOBSIOCC	SIO condition code
16 (10)	4	IOBSTART	Channel program address
20 (14)	4	IOBDCBPT	Data control block address
24 (18)	1	IOBREPM	Reposition modifier
24 (18)	4	IOBRESTR	Channel program restart address
28 (1C)	1	TPBMXREC	Maximum RJE output record count
28 (1C)	2	IOBINCAM	Block count increment
30 (1E)	2	IOBERRCT	Error count
32 (20)	1	TPBLCCC	Last remote output command operation
32 (20)	4	TPBLCCAD	Last remote carriage control address

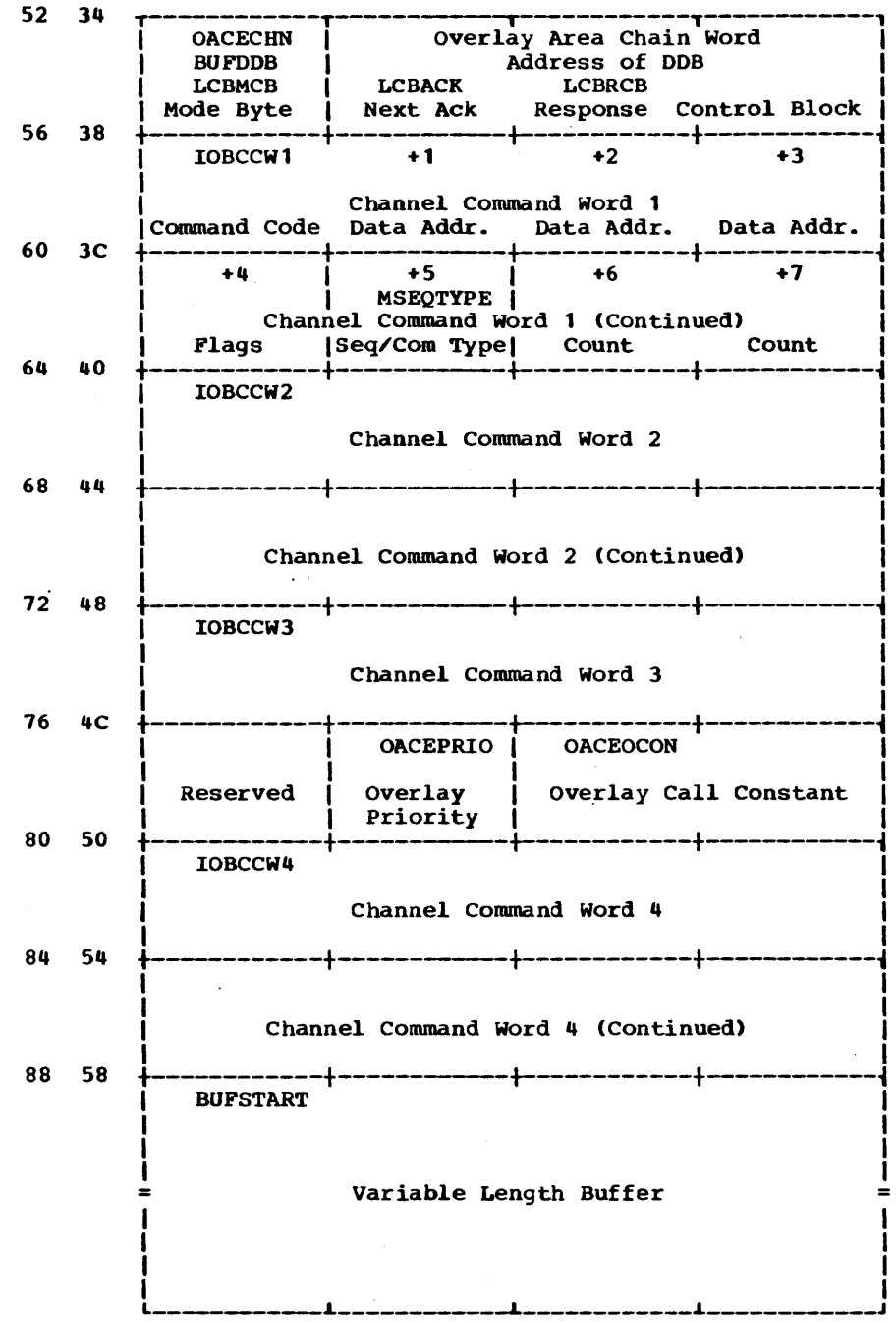
HASP BUFFER

Offset	Bytes and Bits	Field Name	Description
32 (20)	1	IOBXTENT	DEB extent
33 (21)	7	IOBSEEK	Direct-access seek address
36 (24)	1	TPBRECNT	Current remote output record count
36 (24)	4	TPBFDATA	Remote data pointer
40 (28)	1	BUFECBCC	I/O completion code
40 (28)	4	BUFCHAIN	Buffer chain field
44 (2C)	1	BUFTYPE	Buffer type
44 (2C)	4	BUFDCT	Device Control Table address
48 (30)	4	BUFEWF	Event Wait Field or $POST address
52 (34)	4	OACECHN	Overlay area chain word
52 (34)	4	BUFDDB	DDB address
52 (34)	1	LCBMCB	Remote mode byte
53 (35)	1	LCBACK	Remote next acknowledgment
54 (36)	2	LCBRCB	Remote response control block
56 (38)	8	IOBCCW1	Channel Command Word 1
64 (40)	8	IOBCCW2	Channel Command Word 2
72 (48)	8	IOBCCW3	Channel Command Word 3
77 (4D)	1	OACEPRIO	Overlay routine priority in this area
78 (4E)	2	OACEOCON	Overlay call constant
80 (50)	8	IOBCCW4	Channel Command Word 4
88 (58)	0	BUFSTART	Start of buffer work space
88 (58)	4	OACENAME	Overlay routine name
88 (58)	4	HDBNXTRK	HASP data block chain track
88 (58)	8	IOBCCW5	Channel Command Word 5
92 (5C)	4	OACEASMO	Assembly origin of overlay routine
92 (5C)	4	HDBDSKEY	HASP data block data set key

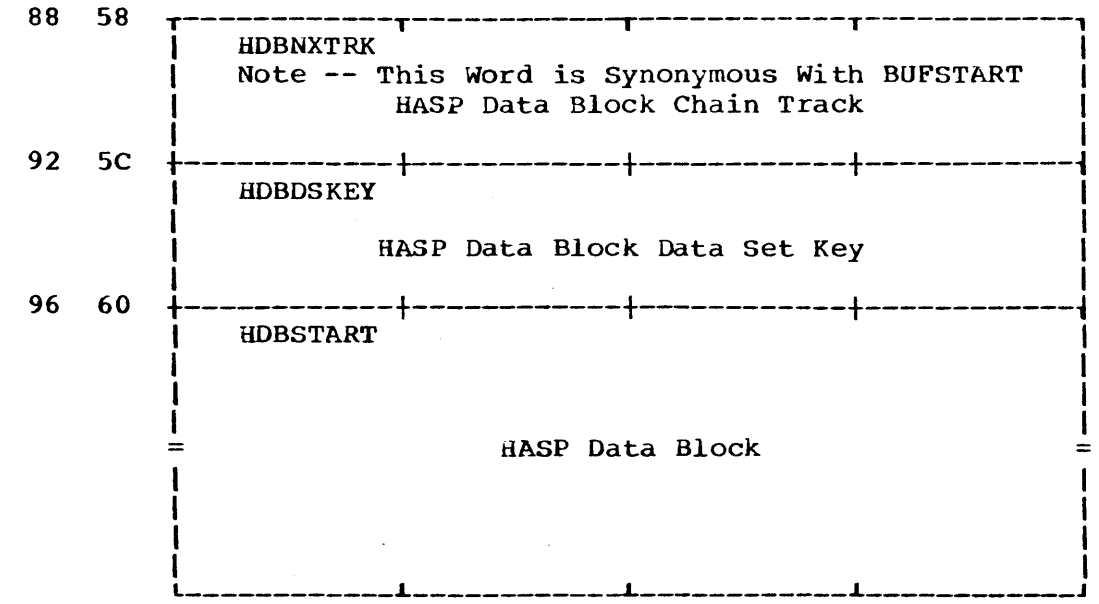
Offset	Bytes and Bits	Field Name	Description
96 (60)	4	OACEPROG	Overlay routine entry point
96 (60)	4	HDBSTART	HASP data block start
96 (60)	8	IOBCCW6	Channel Command Word 6
104 (68)	8	IOBCCW7	Channel Command Word 7
112 (70)	8	IOBCCW8	Channel Command Word 8
120 (78)	0	TPBUFST	Remote buffer work space start

HASP BUFFER


DEC HEX
00 00
IOBFLAG1
I/O Flags
IOBFLAG2
I/O Flags
IOBSENS0
First Sense Byte
IOBSENS1
Second Sense Byte
04 04
IOBECBPT
IOBECBCC
I/O Comple-tion Code
Address of HASP Event Control Block
08 08
IOBFLAG3
I/O Flags
IOBCSW
+1
+2
Channel Status Word
Byte 2
Byte 3
Byte 4
12 0C
+3
+4
+5
+6
Channel Status Word (Continued)
Byte 5
Byte 6
Byte 7
Byte 8
16 10
IOBSTART
IOBSIOCC
SIO Condi-tion Code
Address of Channel Program
20 14
IOBDCBPT
Address of Data Control Block
24 18
IOBRESTR
IOBREPM
Reposition Modifier
Restart Address of Channel Program
28 1C
IOBINCAM
TPBMXREC
Remote Max Record Count
Block Count Increment
IOBERRCT
Error Count
32 20
IOBXTENT
TPBLCCAD
TPBLCCC
Last Car Ctl
IOBSEEK
+1
+2
Direct Access Seek Address
Addr of Last Remote Carriage Control
36 24
+3
+4
+5
+6
TPBFDATA
TPBRECNT
Record Count
Direct Access Seek Address (Continued)
Remote Data Pointer
40 28
BUFCHAIN
BUFECBCC
I/O Comple-tion Code
Buffer Chain Field
44 2C
BUFDCT
BUFTYPE
Buffer Type
Address of Device Control Table
48 30
BUFEWF
EWF Flags
Event Wait Field or Post Address

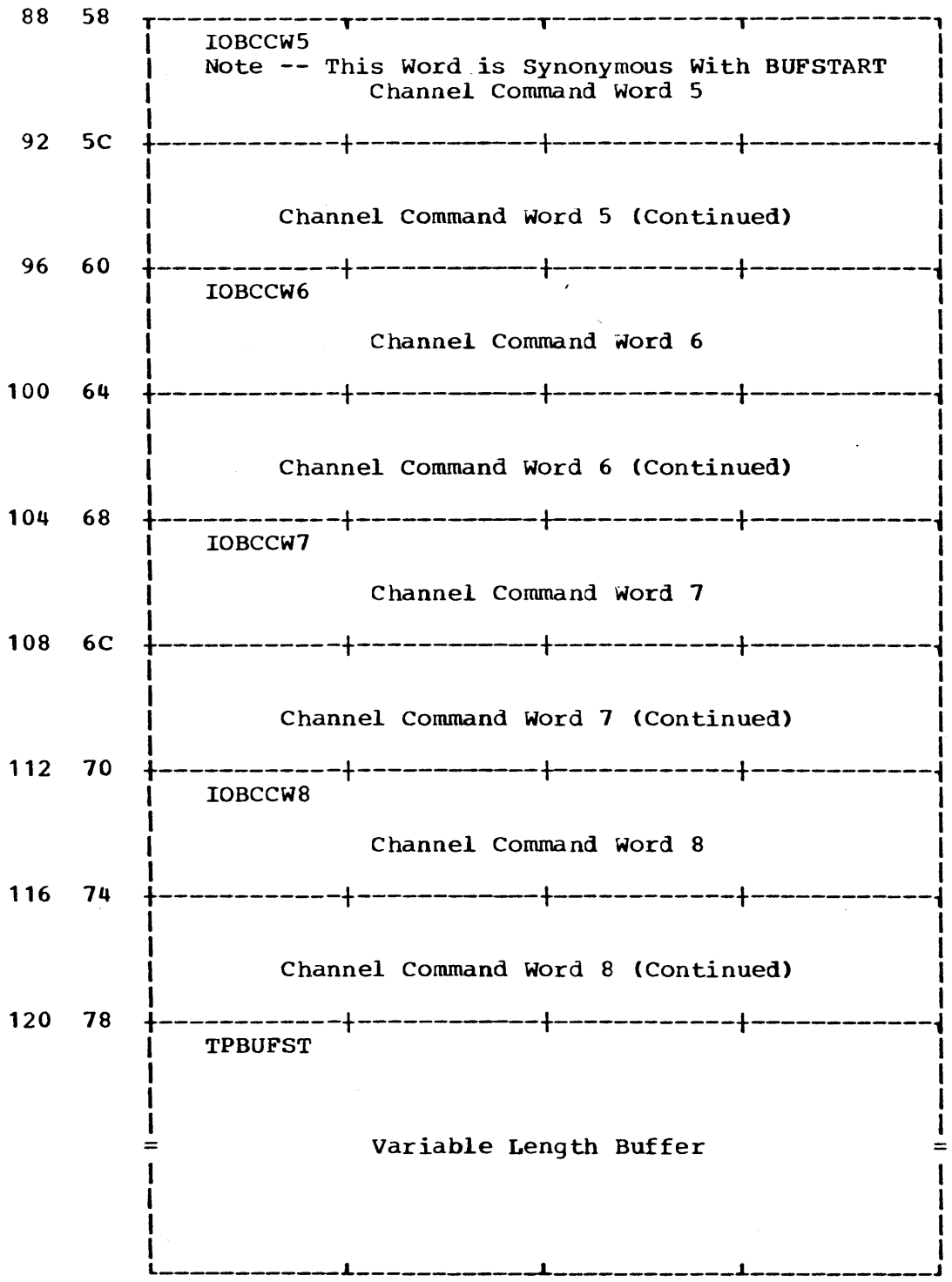

DEC HEX
52 34
OACECHN
BUFDDB
LCBMCB
Mode Byte
Overlay Area Chain Word
Address of DDB
LCBACK
Next Ack
LCBRCB
Response
Control Block
56 38
IOBCCW1
+1
+2
+3
Channel Command Word 1
Command Code
Data Addr.
Data Addr.
Data Addr.
60 3C
+4
+5
MSEQTYPE
+6
+7
Channel Command Word 1 (Continued)
Flags
Seq/Com Type
Count
Count
64 40
IOBCCW2
Channel Command Word 2
68 44
Channel Command Word 2 (Continued)
72 48
IOBCCW3
Channel Command Word 3
76 4C
OACEPRIO
OACEOCON
Reserved
Overlay
Priority
Overlay Call Constant
80 50
IOBCCW4
Channel Command Word 4
84 54
Channel Command Word 4 (Continued)
88 58
BUFSTART
Variable Length Buffer

HASP Data Block Buffer Extension

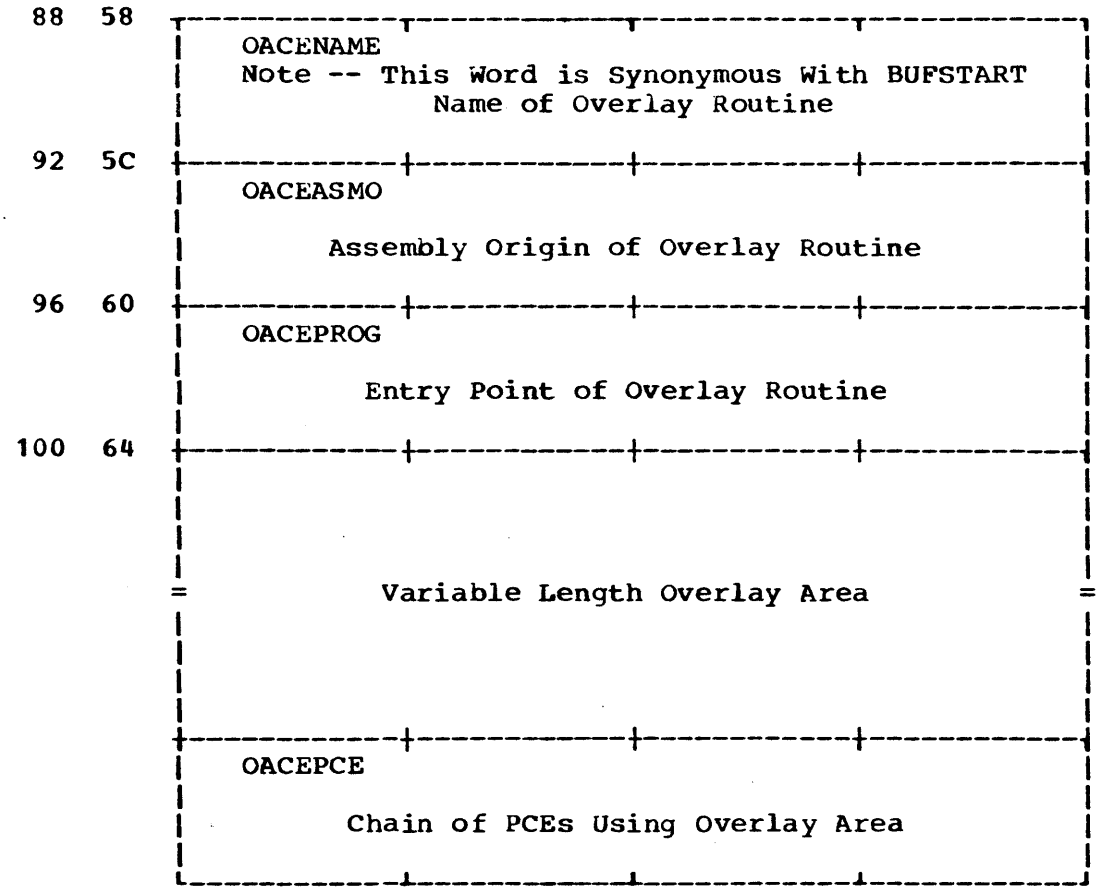


HASP BUFFER

HASP Remote Job Entry Buffer Extension



HASP Overlay Area Buffer Extension



HASP CONSOLE MESSAGE BUFFER

The Console Message Buffer (CMB) is used for four basic functions:

1. Hold a command entered from OS console, HASP reader device or HASP remote console device.

2. Hold responses from the Command Processor while queued to the HASP communications subtask or Remote Control Processor.

3. Hold $WTO or WTO messages while queued to the HASP Job Log Processor, communications subtask or Remote Console Processor.

4. Hold $WTO messages and DOM ID that require immediate operator action, allowing the redisplaying of the message and DOM for deletion of the message from OS display devices.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	CMBCHAIN	ADDR of next Console Message Buffer
04 (04)	1	CMBFLAGS	Flag byte
	1x.1	WCMBFD	End of out-of-line WTO (MLWTO command processing response)
	.x1.	WCMBFE	Message for HASP Log only
	.x.1	WCMBFF	CMBCONS contains UCMID (used for command input and response)
	.x.. 1...	WCMBFG	CMBCONS contains remote number
	.x.. 01..	WCMBFA	Input source does not have job authority
	.x.. 0.1.	WCMBFB	Input source does not have system authority
05 (05)	1	CMBCONS	Consoles specified (remote number, UCMID, or logical console routing)
06 (06)	1	CMBMSGL	Message length
07 (07)	1	CMBCLASS	Message class or display area ID

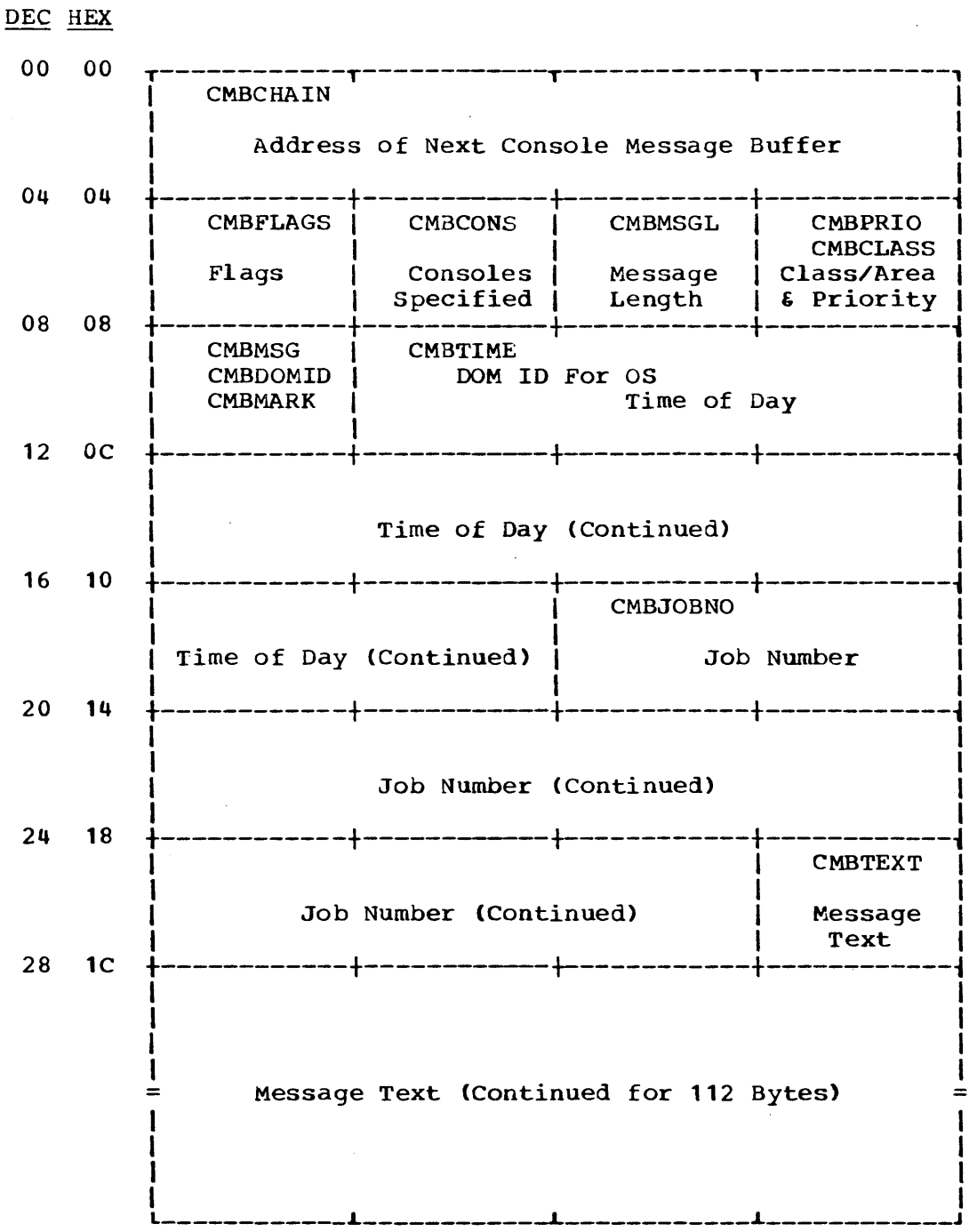
Bit Definitions When WCMBFF = 1

0000	Normal command response (area = Z)
1111	Out-of-line display area A to O (0001 = A, 0010 = B, 0011 = C, etc.)

HASP CONSOLE MESSAGE BUFFER

Offset	Bytes and Bits	Field Name	Description
			Bit Definitions When WCMBFF = 0
	.111		Message class levels 0 to 7 (levels) may be screened out by $TCON)
			When WCMBFE and WCMBFG Also = 0
	1...		Message class requires immediate operator (unconditional display on OS console device)
07 (07)	1	CMBPRIO	Message queueing priority
	 1111		Message priority 0 to 15
08 (08)	132	CMBMSG	Console message
08 (08)	4	CMBDOMID	DOM ID for CMB
08 (08)	1	CMBMARK	Attention indicator
09 (09)	9	CMBTIME	'Time of day HH.MM.ssb'
18 (12)	9	CMBJOBNO	Job number 'JOBbnnnnb'
27 (1B)	113	CMBTEXT	Message text

HASP CONSOLE MESSAGE BUFFER



HASP COMMAND PCE WORK AREA

The Command Work Area is an extension to the Command processor PCE. It is used during the processing of a command to provide space for flags, pointers, message text, and work areas. It resides in HASPNUC and is generated during the assembly of that module.

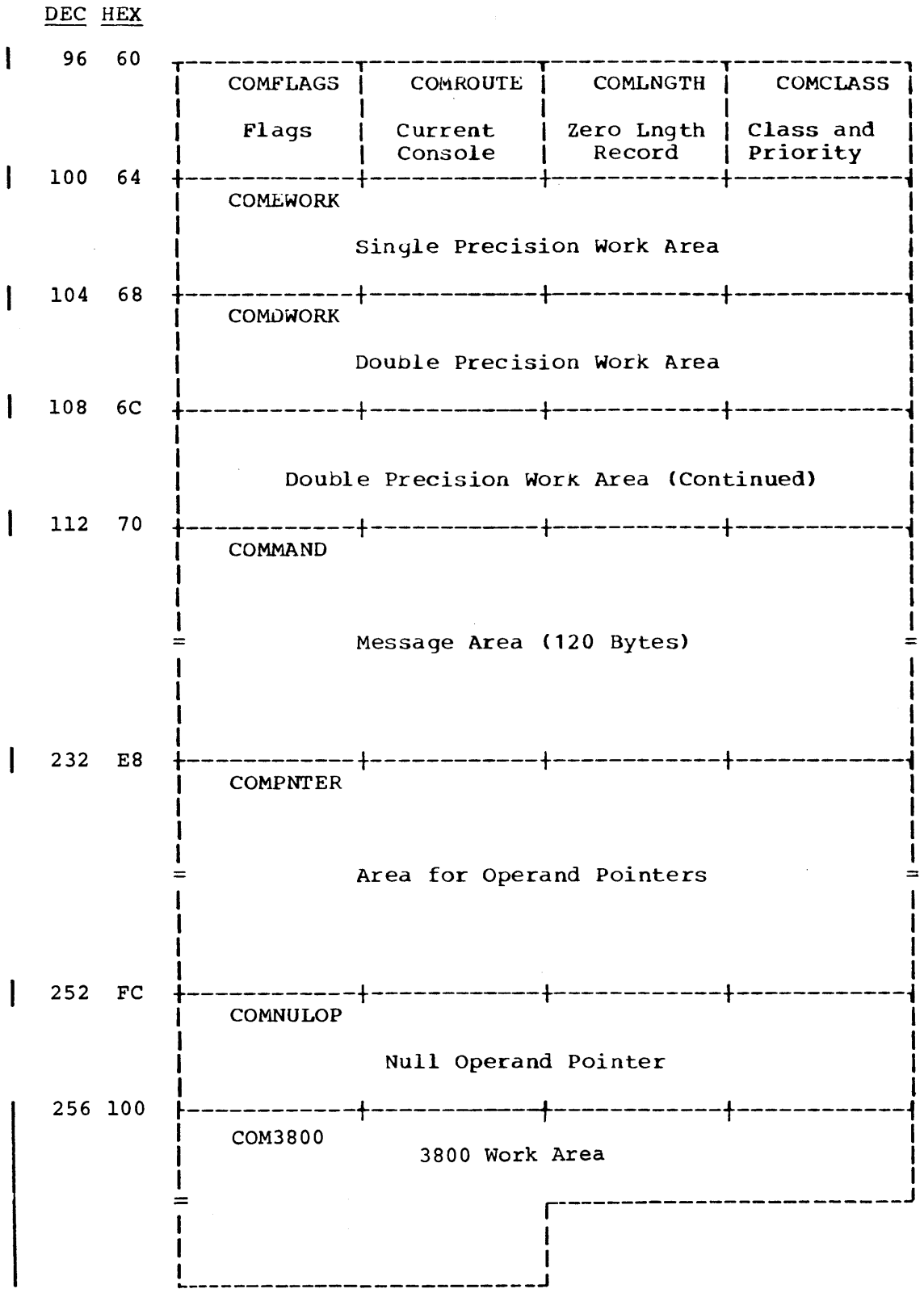
Offset	Bytes and Bits	Field Name	Description
96 (60)	1	COMFLAGS	Byte 1 of list form of $WTO parameter list.
	1001		End of out-of-line MLWTO
	0000 0...		Logical console route code
	0000 1...		Source console/response console is a remote.
	.... 01..		Source console not authorized for job commands (reset prior to $WTO execution)
	.... 0.1.		Source console not authorized for device commands (reset prior to $WTO execution)
	.... 0..1		Source console not authorized for system commands (reset prior to $WTO execution)
97 (61)	1	COMROUTE	Byte 2 of list form of $WTO parameter list contains remote number, UCMID, or logical routings for $WTO.
98 (62)	1	COMLNGTH	Byte 3 of list form of $WTO parameter list (contents=0).
99 (63)	1	COMCLASS	Byte 4 of list form of $WTO parameter list contains x'77' for logical and remote consoles and contains x'07' for UCMID area Z (normal WTO responses). For UCMID responses to out-of-line areas the contents are as follows:

X'17'-Area A	X'97'-Area I
X'27'-Area B	X'A7'-Area J
X'37'-Area C	X'B7'-Area K
X'47'-Area D	X'C7'-Area L
X'57'-Area E	X'D7'-Area M
X'67'-Area F	X'E7'-Area N
X'77'-Area G	X'F7'-Area O
X'87'-Area H	

HASP COMMAND PCE WORK AREA

Offset	Bytes and Bits	Field Name	Description
100 (64)	4	COMEWORK	4 byte work area used by subprocessors and macro generated routines as follows: $CFCVE - last character set to blank $CFDCTL - first characters of requested device name $CFJDCT - address of HASP job queue element for requested job $CFJMSG - same as $CFCVE
104 (68)	8	COMDWORK	8 byte work area used by subprocessors and macro generated routines as follows: $CFCVE - 5-character number in EBCDIC with leading blanks $CFDCTL - last four characters of requested device name $CFJMSG - same as $CFCVE
112 (70)	120	COMMAND	120 byte work area used to hold up to 119 character command, responses to commands, and general scratch work information
113 (71)	1	COMVERB	Single character verb
114 (72)	1	COMOPRMD	First character of the first operand
232 (E8)	20	COMPNTER	Five full words containing pointers to the first character of each operand (pointers right adjusted if fewer than 5 operands). Also used by subprocessors as scratch work areas after locating all required operands.
252 (FC)	4	COMNULOP	Address of the end of the current command plus two bytes.
256(100)	6	COM3800	3800 work area

HASP COMMAND PCE WORK AREA



HASP DEVICE CONTROL TABLE

The Device Control Table is a control block which represents a real unit record device or a portion of the total direct-access space available. The DCT is used for three purposes: (1) allocation and deallocation of a unit record device to HASP processors, (2) communication of operator commands between the communication processor and the processor using the unit record device, and (3) as a parameter list to the $EXCP subroutine.

Offset	Bytes and Bits	Field Name	Description
00 (00)	1	DCTSTAT	Device Control Table status
	1... xxxx	DCTINUSE	DCT is in use
	.1.. xxxx	DCTDRAIN	DCT is drained
	..1. xxxx	DCTHOLD	DCT is held
	...1 xxxx	DCTEJECT	Device is at channel 1 position
00 (00)	4	DCTPCE	Address of Processor Control Element
04 (04)	4	DCTBUFAD	Current buffer address
08 (08)	1	DCTPSTAT	Remote Job Entry flags
	1...	DCTLOGAL	Log every channel end
	.1..	DCTLEASE	Leased line
	..1.	DCTETX	An ETX has been received
	...1	DCTSOFF	A /*SIGNOFF card has been received
	...1	DCTEOF	An EOF has been detected
	.... 1...	DCTSINON	Remote DCT is attached to line DCT
	.... .1..	DCTPOST	I/O complete flag
	.... ..1.	DCTABORT	Transmission was aborted
	1	DCTPBUF	Remote output buffer indication
08 (08)	4	DCTDCB	Address of Data Control Block
08 (08)	4	DCTSEEK	Current track address
12 (0C)	1	MDCTOPCT	Count of open RJE processors
12 (0C)	4	MDCTOBUF	RJE output buffer chain
12 (0C)	4	DCTEWF	Event Wait Field or post address
16 (10)	1	DCTBUFCT	Active buffer count
17 (11)	1	MDCTATTN	RJE line action pending flags
17 (11)	1	DCTNO	DCT number

HASP DEVICE CONTROL TABLE

Offset	Bytes and Bits	Field Name	Description
18 (12)	1	DCTDEVTP	Device type
	0000 0000	DCTDA	Direct-access device
	0000 0001	DCTOLAY	Overlay device
	0000 0010	DCTLNE	Remote Job Entry line
	0001 0000	DCTRDR	Local card reader
	0001 0010	DCTRJR	Remote card reader
	0001 0100	DCTINR	Internal Reader
	0010 0000	DCTPRT	Local printer
	0010 0010	DCTRPR	Remote printer
	0011 0000	DCTPUN	Local punch
	0011 0010	DCTRPU	Remote punch
	0100 0010	DCTRCON	Remote console
19 (13)	1	DCTIOTYP	I/O request type
	1...	DCTREAD	Read request
	.1..	DCTWRITE	Write request
	.... 1...	DCTREJRM	Remote restriction (always 0)
	.... .1..	DCTREJJB	Restricted from job commands
	.... ..1.	DCTREJDV	Restricted from device commands
	1	DCTREJSY	Restricted from system commands
19 (13)	1	DCTPCODE	Line/Remote code
	00	DCTPHALF	Half-duplex line
	01	DCTPFULL	Full-duplex line
	10	DCTPWIDE	Wide-band line
	..1.	DCTPPRES	Hardware compress feature
	...1	DCTPCON	Remote terminal console
	...1	DCTPMRF	Multiple-record feature
	.... 1...	DCTPTAB	Horizontal format control
	.... .1..	DCTPROG	Programmable interface
	.... ..1.	DCTPVAR	Variable-length records
	1	DCTPBLK	Blocked records
20 (14)	1	DCTFLAGS	Operator command flags
	1...	DCTSTOP	$Z ($STOP) command
	.1..	DCTDELET	$C ($DELETE) command
	..1.	DCTRSTRT	$E ($RESTART) command
	...1	DCTRPT	$N ($REPEAT) command
	.... 1...	DCTBKSP	$B ($BACKSPACE) command
	.... .1..	DCTHOLDJ	$T...,H command
	.... .1..	DCTSPACE	$T...,C=1 command
20 (14)	4	DCTCHAIN	Address of next DCT
24 (18)	8	DCTDEVN	EBCDIC device name
28 (1C)	2	DCTOTC	Overlay device tracks/cylinder
30 (1E)	2	DCTOTT	Overlay extent beginning TT
32 (20)	4	MDCTCODE	Address of RJE Code Table

HASP DEVICE CONTROL TABLE

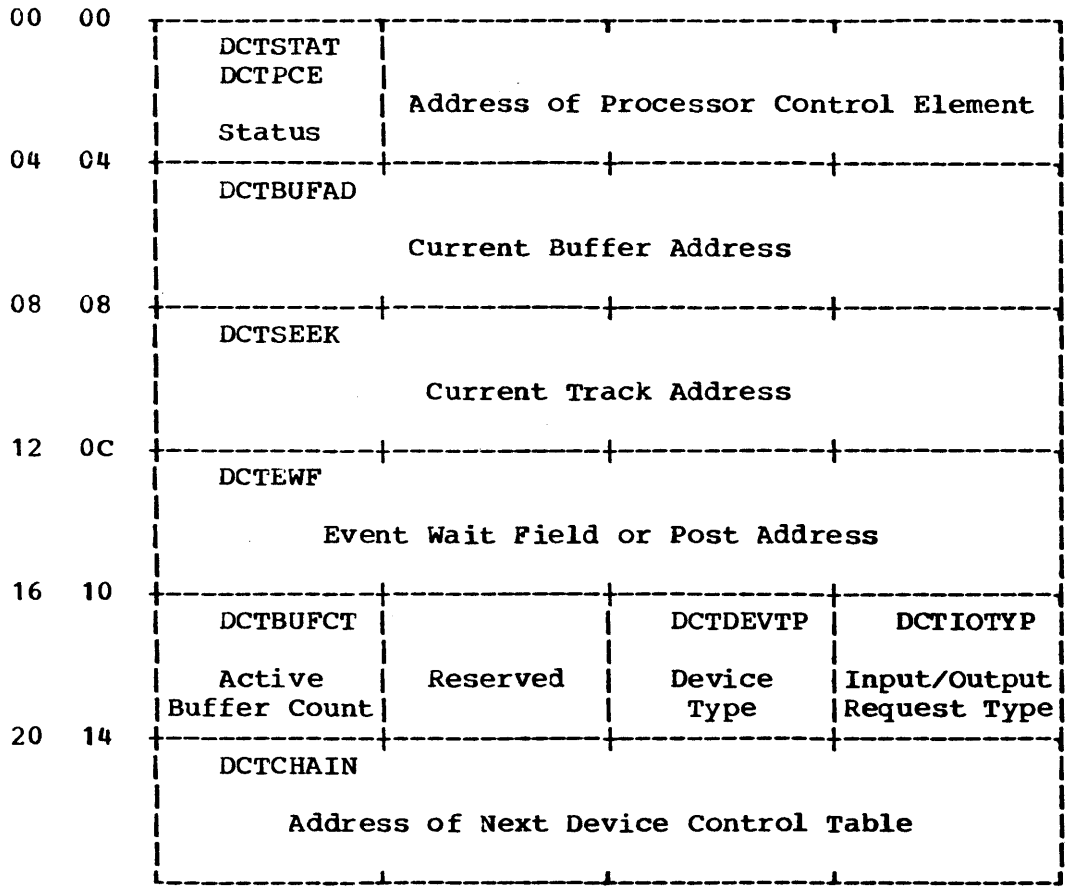
Offset	Bytes and Bits	Field Name	Description
32 (20)	4	DCTFORMS	Print/Punch forms identification
32 (20)	1	DCTPRINT	Default print routing
33 (21)	1	DCTPUNCH	Default punch routing
34 (22)	1	DCTPRINC	Priority increment
35 (23)	1	DCTPRLIM	Priority limit
36 (24)	4	DCTFCB	Print FCB identification
40 (28)	4	DCTUCS	Print UCS identification
44 (2C)	4	DCTDCTE	Pointer to DCT extension for 3800
48 (30)	1	DCTPPSW	Print/punch switches
	1...	DCTPPSWC	FCB carriage altered
	.1..	DCTPPSWF	Forms controlled by operator
	..1.	DCTPPSWG	Queue classes are altered
	...1	DCTPPSWS	Suppress separator pages
	.... 1...	DCTPPSWT	UCS train altered
	.... .1..	DCTPPSWU	UCS not standard
	.... ..1.	DCTPPSWI	Device idle message issued
	1	DCTPPSWO	Operator action allowed
49 (31)		DCTCLASS	Variable length print/punch class mask
	1		Class mask terminator
	0	DCTWORK	Start of device work area
	4	RIDUCB	Internal Reader UCB address
	2	MDCTFCS	Remote terminal Function Control Seq
	1	MDCTERCT	Remote terminal error count
	1	DCTPRLEN	Remote terminal data width
	1	DCTPLINE	Remote terminal line characteristics
	 0000	DCTP2770	1009, 2770, 3780
	 0001	DCTPHARD	1978, 2780
	 0010	DCTP20	360/20 submodel 5 or 6
	 0100	DCTP360	360/22, 25, 30, 40, etc.
	 0110	DCTP20S2	360/20 submodel 2 or 4, 2922
	 1000	DCTP1130	1130
	 1010	DCTPSYS3	System/3
	..1.	DCTPASCI	USASCII code
	.1..	DCTPTRSP	Transparency
	1...	DCTPBSC	Binary synchronous line

HASP DEVICE CONTROL TABLE

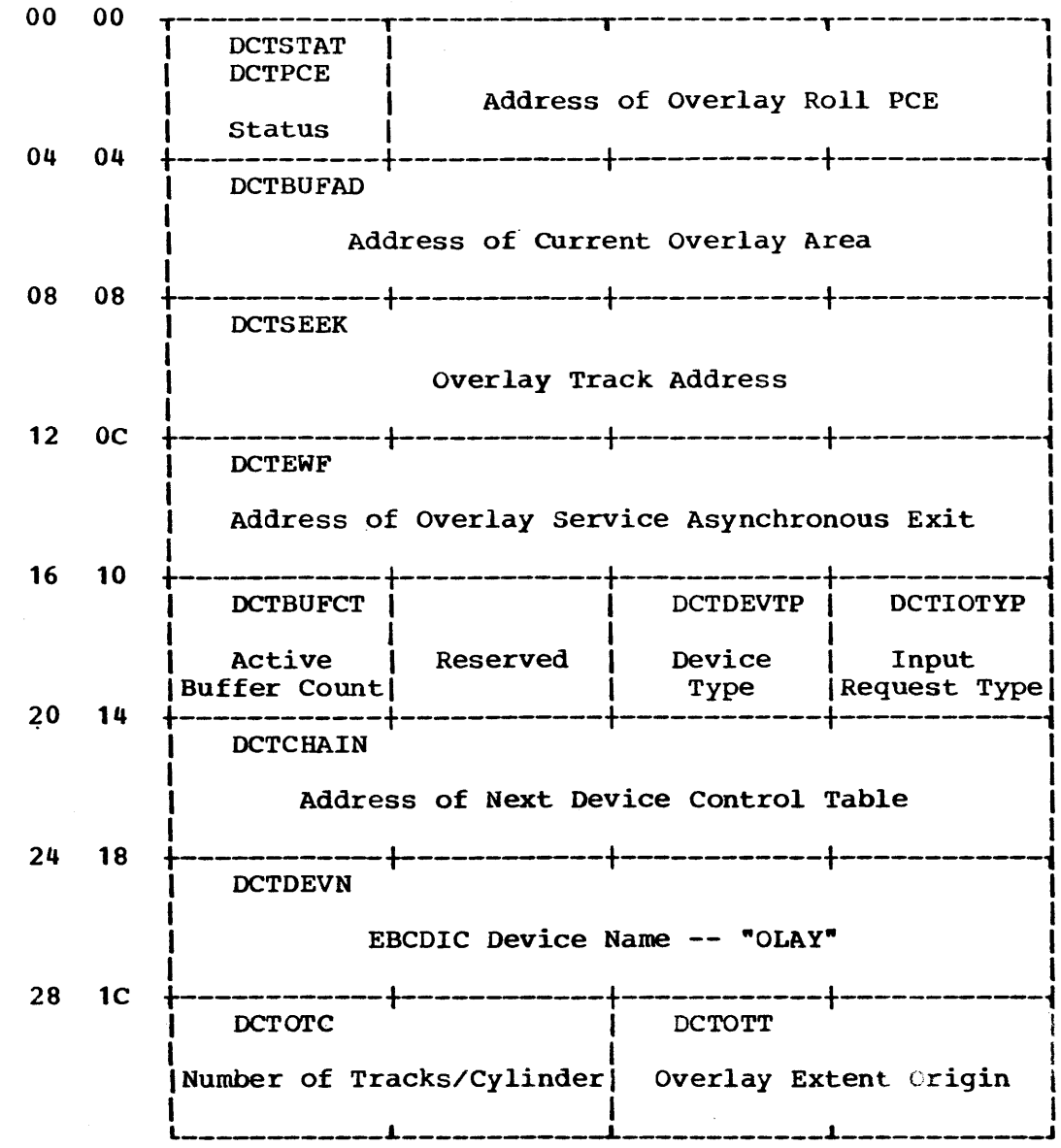
Offset	Bytes and Bits	Field Name	Description
	1	MDCTRCB	Remote terminal record control byte
	4	MDCTDCT	Remote terminal DCT chain field
	2	RIDFLAGS	Internal Reader sync flags
	1...	RIDPOST	User waiting for post
	.1..	RIDBUSY	I/O simulation in process
	2	RIDTJID	Reserved
	4	RIDECB	Internal Reader ECB address
	1	MDCTRSEQ	Remote transmit sequence count
	1	MDCTTSEQ	Remote receive sequence count
	8	MDCTPSWD	Remote terminal password
	4	RIDTCB	Internal Reader TCB address
	80	RIDDATA	Internal Reader data area
	0	DCTEND	Symbol for end of DCT

HASP DEVICE CONTROL TABLE

DIRECT-ACCESS

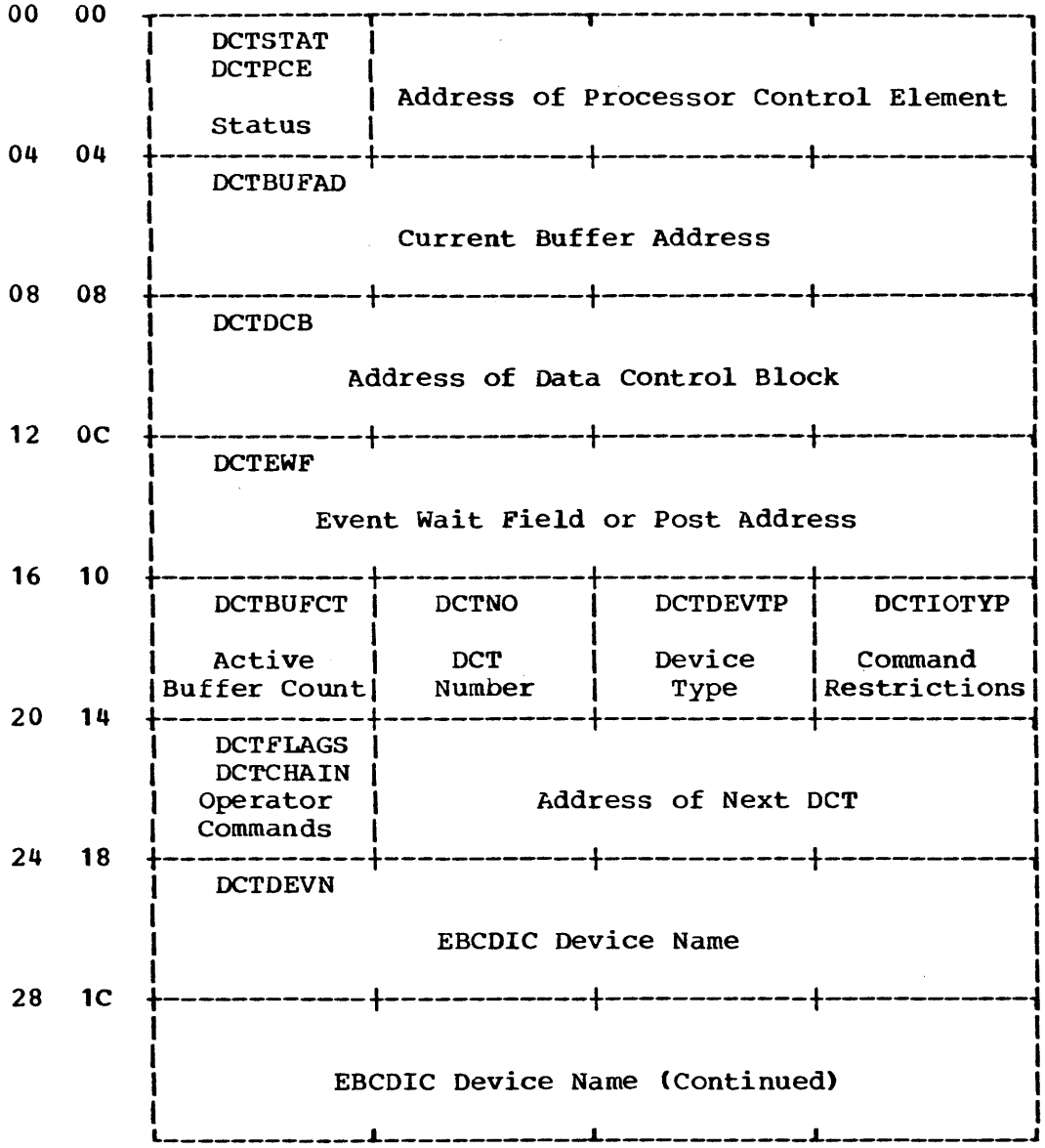


HASP DEVICE CONTROL TABLE

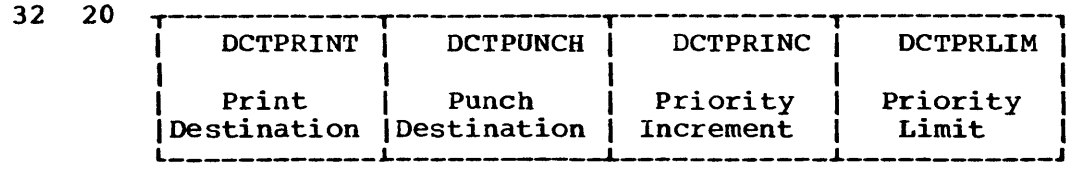


HASP DEVICE CONTROL TABLE

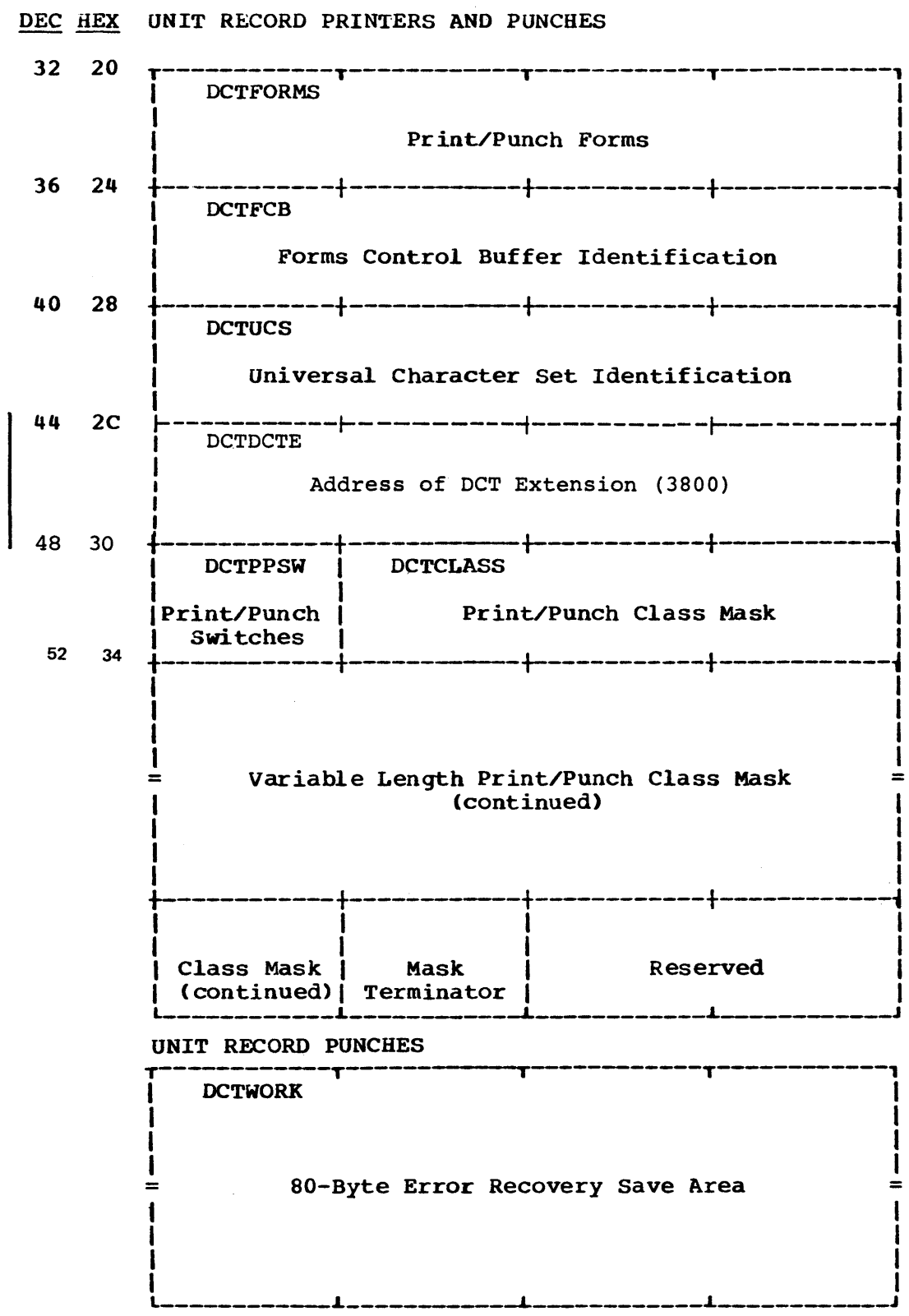
UNIT RECORD READERS, PRINTERS, AND PUNCHES



DEVICES OTHER THAN UNIT RECORD PRINTERS AND PUNCHES

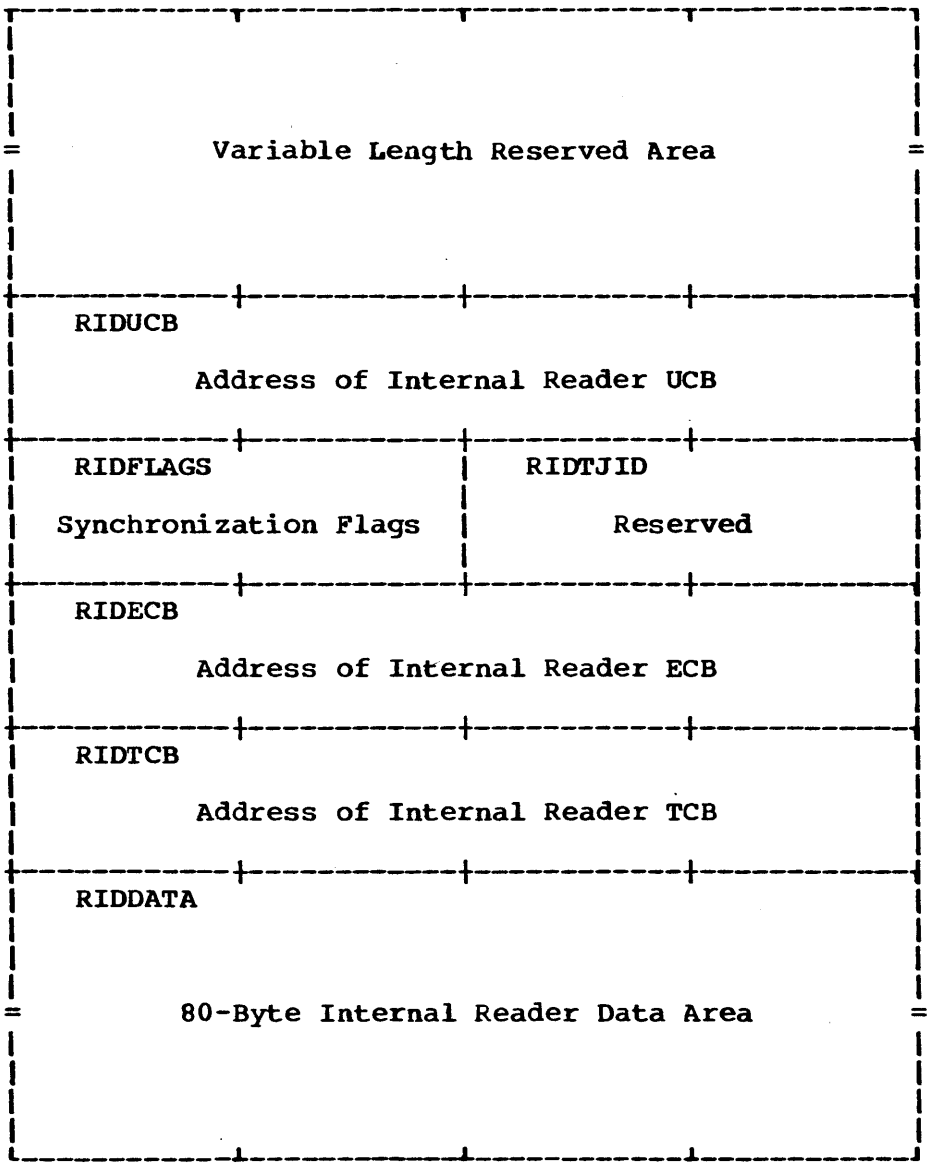


HASP DEVICE CONTROL TABLE



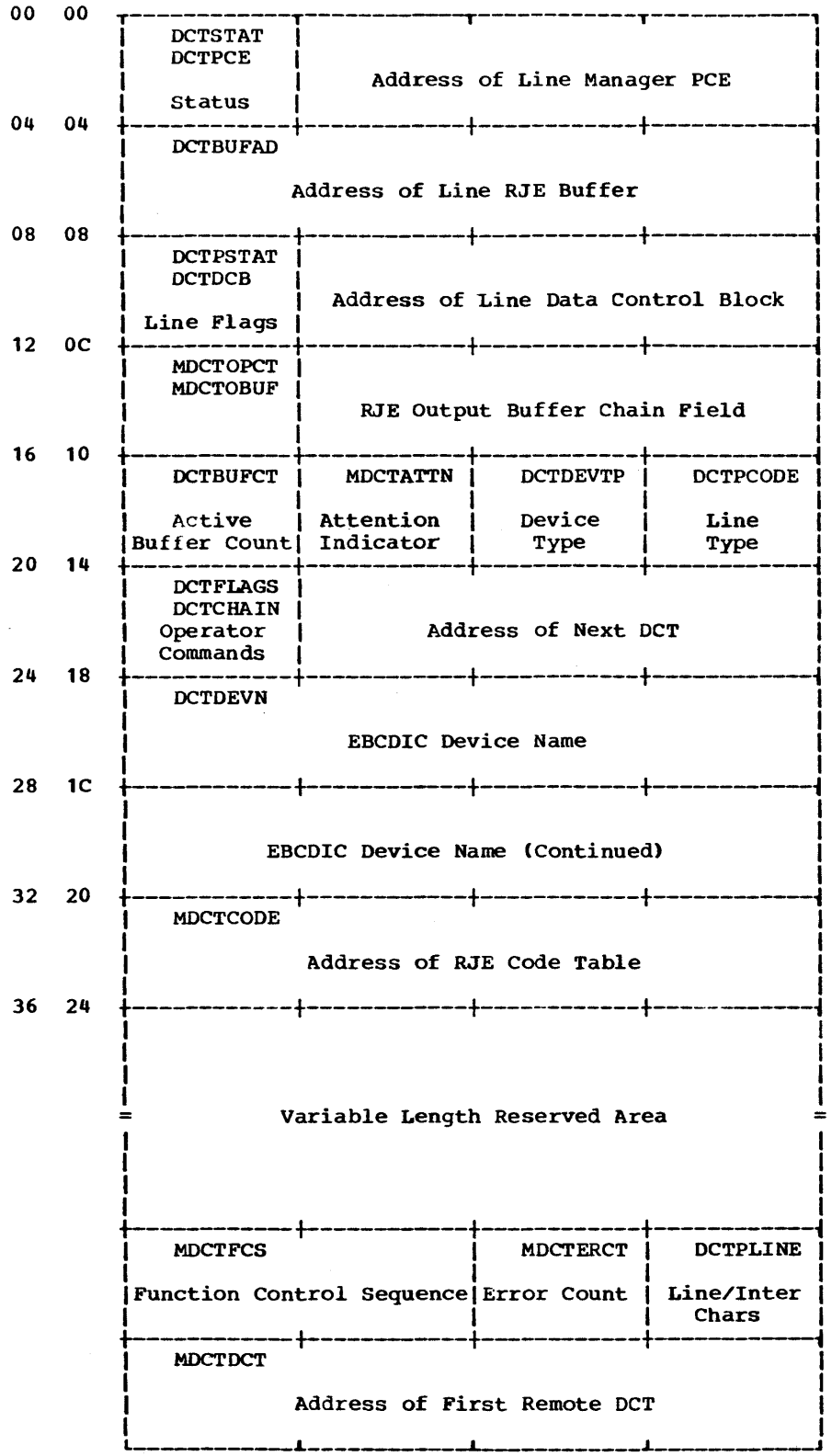
DEC HEX INTERNAL READERS

36 24

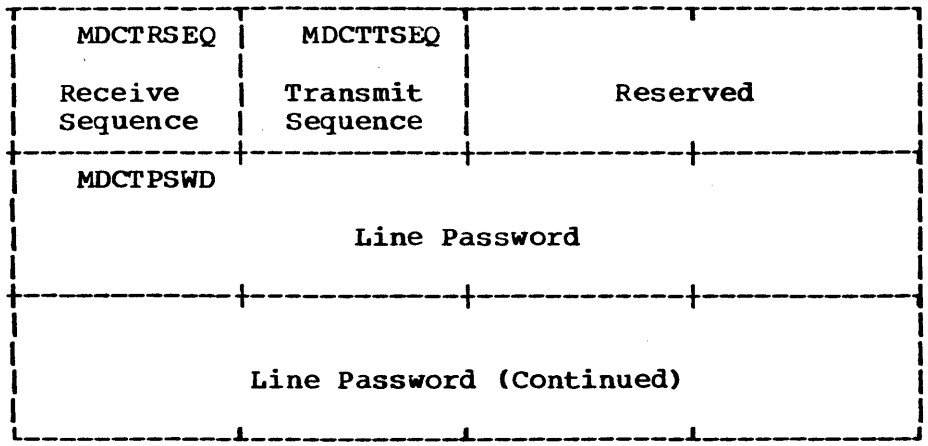


HASP DEVICE CONTROL TABLE

REMOTE JOB ENTRY LINES

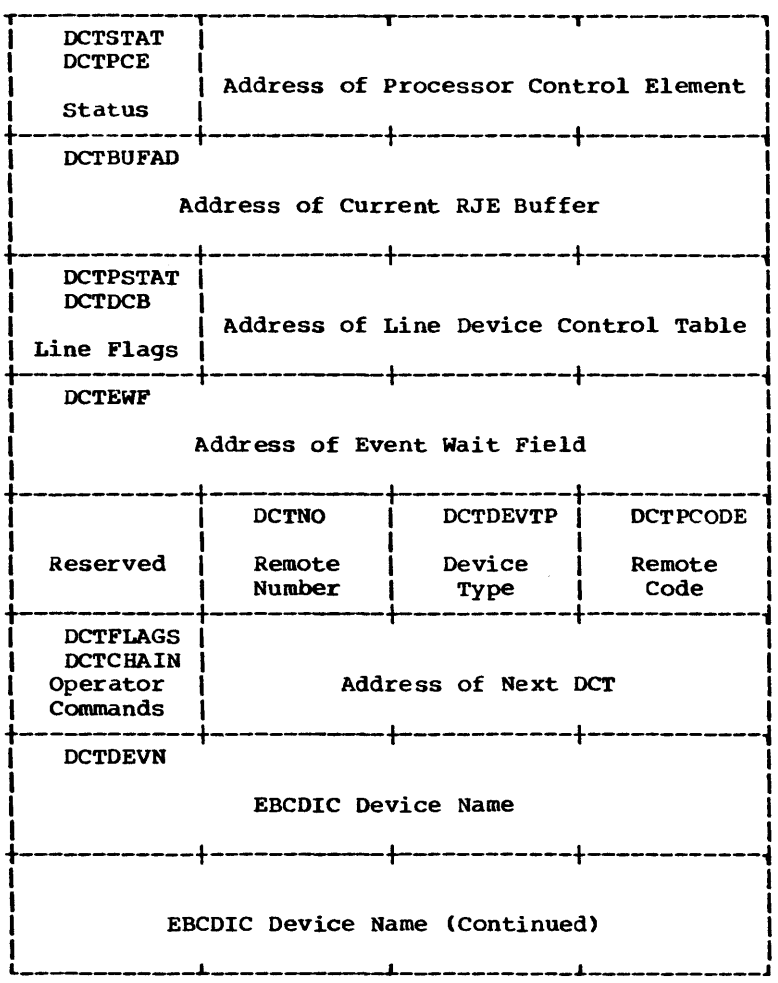


DEC HEX REMOTE JOB ENTRY LINES (Continued)

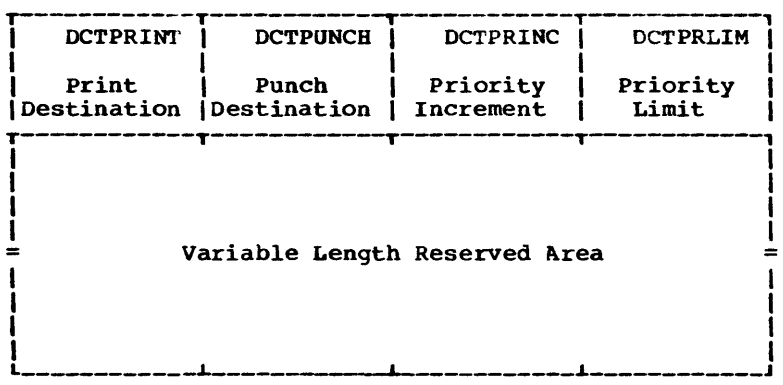


HASP DEVICE CONTROL TABLE

ALL REMOTE DEVICES

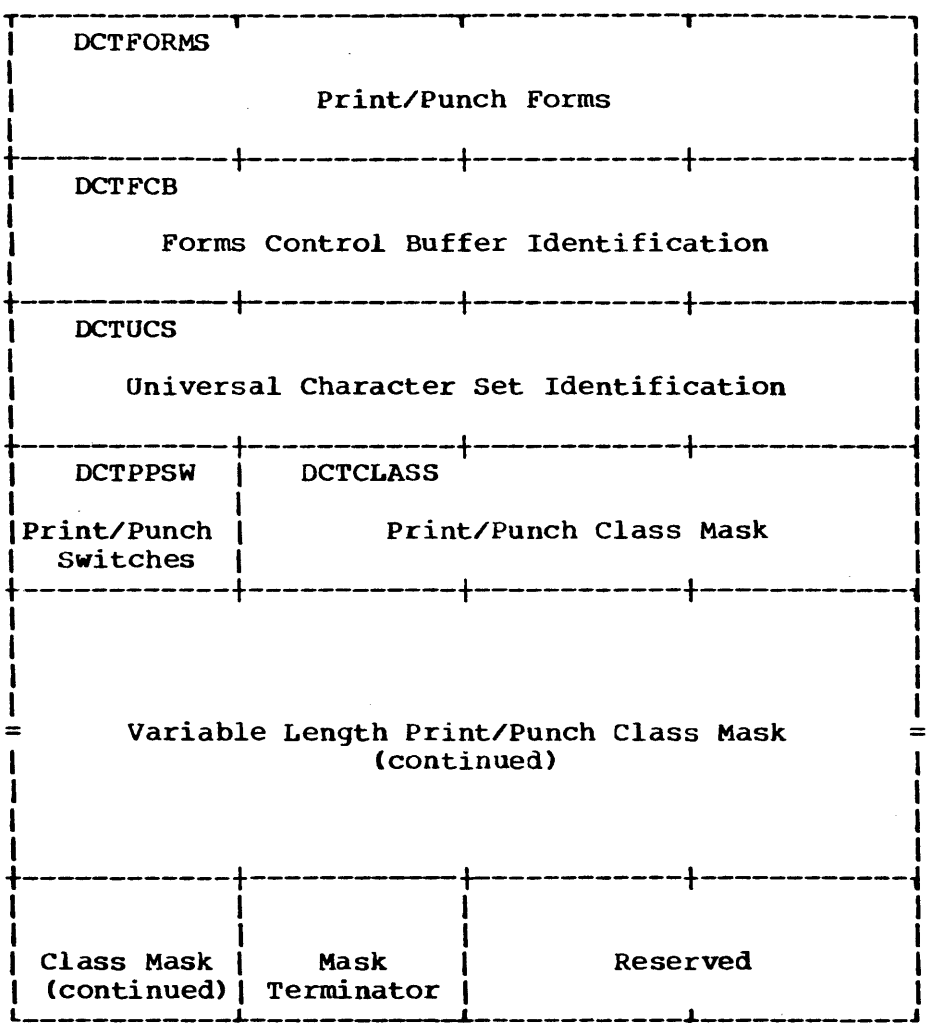


REMOTE READERS AND REMOTE CONSOLES

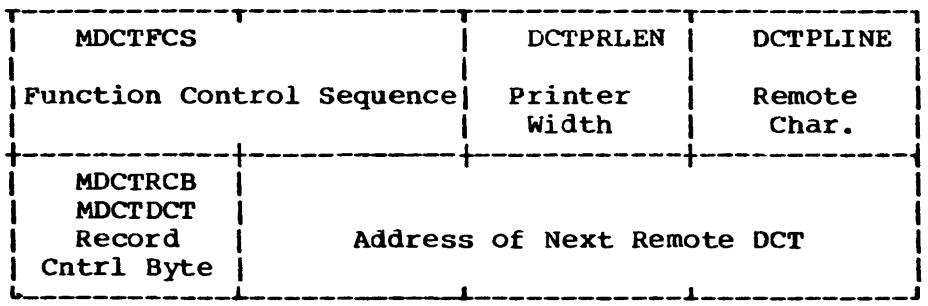


HASP DEVICE CONTROL TABLE

DEC HEX REMOTE PRINTERS AND REMOTE PUNCHES



ALL REMOTE DEVICES



HASP DEVICE CONTROL TABLE EXTENSION (3800 Printer)

The Device Control Table Extension (DCTE) is an extension to the DCT which is present only for 3800 printers. The DCTE contains the SETPRT parameter lists, as well as information on the current setup of the 3800.

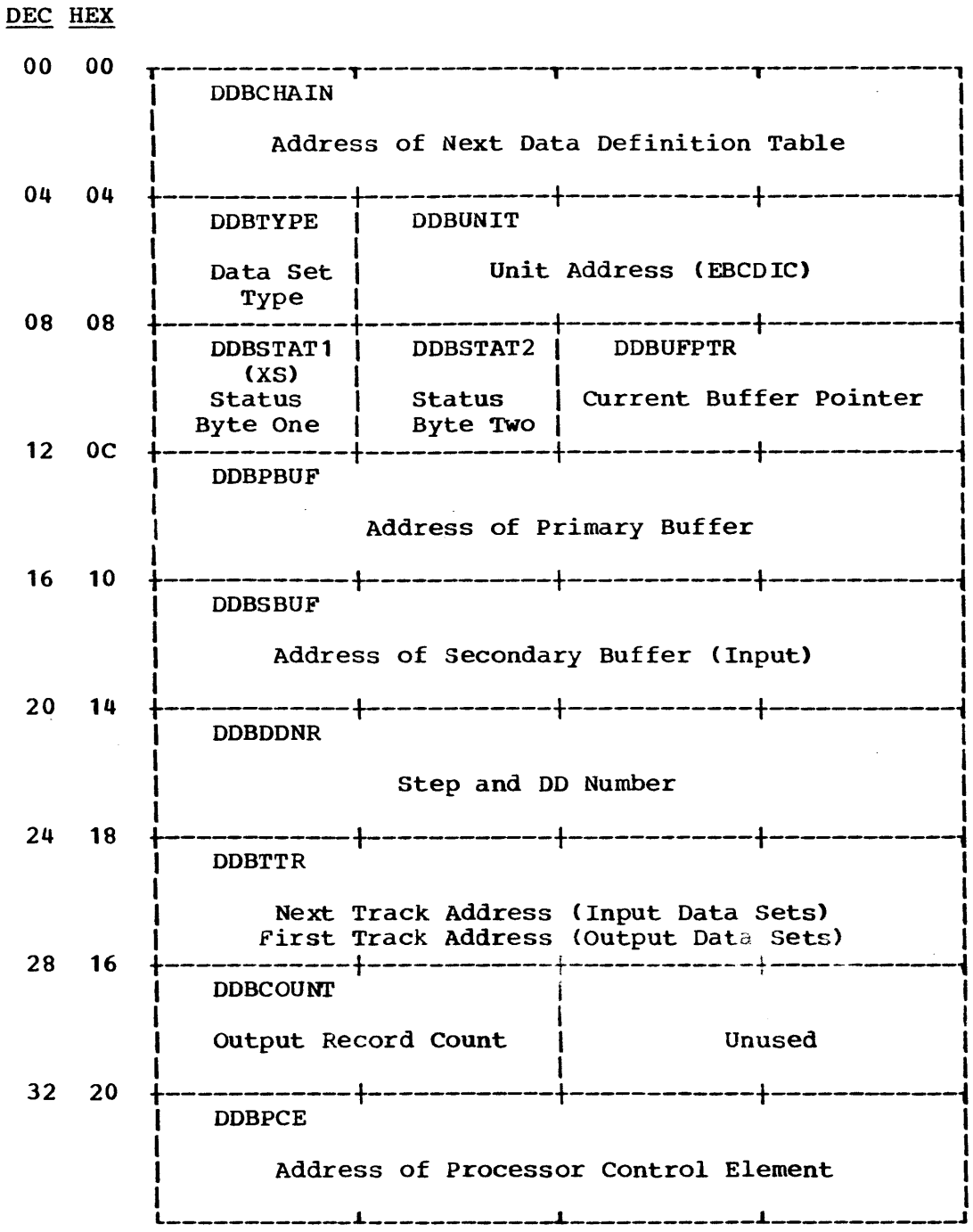
Offset	Bytes and Bits	Field Name	Description
00 (00)	4	DCTEDCT	Pointer to DCT
04 (04)	4		Reserved
08 (08)	48	DCTESPL1	SETPRT Parameter list for Separator page setup
56 (38)	48	DCTESPL2	SETPRT Parameter list for Job output
104 (68)	4	DCTEUCBX	UCB extension address
108 (6C)	4	DCTECPCD	SETPRT Completion Code
112 (70)	1	DCTEFLAG	
	1...	DCTEFLOP	Operator has issued $TPRT
	.1..	DCTEJAM	Paper jam
	..1.	DCTEMARK	Marking forms
	.... .1..	DCTECM	Setup change required
	.... ..1.	DCTERST	Restart after SETPRT error
113 (71)	1	DCTEFLCT	FLASH count
114 (72)	1	DCTEJAMC	Paper jam page count
115 (73)	1	DCTEFL2	Reserved
116 (74)	8	DCTECPYG	Copy Groups
124 (7C)	4	DCTEFCB	Current FCB
128 (80)	16	DCTECHAR	Current Character Arrangement Tables
144 (90)	4	DCTECPYM	Current Copy Modification ID
148 (94)	1	DCTETRC	Current Copy Modification reference Character
149 (95)	1	DCTEFRMN	Current Flash count Loaded
150 (96)	1	DCTECPYN	Current Copy Count LOADED
151 (97)	1	DCTESTCN	Current Start Copy marker Loaded
152 (98)	52	DCTEDCB	DCB for SETPRT

HASP DATA DEFINITION TABLE

The Data Definition Block (DDB or DDT) contains for an active SPOOL data set (except an Internal Reader data set) current buffer information, I/O activity status, output record count, and direct-access location. It is used only in module HASPXEQ.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	DDBCHAIN	Next Data Definition Table address
04 (04)	1	DDBTYPE	Data set type
05 (05)	3	DDBUNIT	Unit address (EBCDIC)
08 (08)	1	DDBSTAT1	Status byte one
08 (08)	1	XS	For shorthand (EQU)
09 (09)	1	DDBSTAT2	Status byte two
10 (0A)	2	DDBUFPTR	Current buffer pointer
12 (0C)	4	DDBPBUF	Address of primary buffer
16 (10)	4	DDBSBUF	Address of secondary buffer (INPUT)
20 (14)	4	DDBDDNR	Step and DD number
24 (18)	4	DDBTTR	Next/first track address
28 (1C)	2	DDBCOUNT	Output record count
30 (1E)	2		Unused
32 (20)	4	DDBPCE	Address of Processor Control Element
36 (24)	0	DDBEND	End of DDT
36 (24)	0	DDBLNG	Length of DDT (EQU)

HASP DATA DEFINITION TABLE



HASP COMMUNICATION TABLE

The HASP Communication Table (HCT) performs a central organizational role in the HASP System. It assembles as actual code in the HASPNUC assembly and is physically at the beginning of the HASPNUC CSECT, which in turn is physically at the beginning of the HASP load module, which is forced (by link edit control cards) to begin on a 4K page boundary. In other HASP assembly modules, the HCT expands as a DSECT. All HASP processors are provided with addressability to the HCT in register BASE1 (which is register 11). The HCT contains several major categories of information, as follows:

The HASP version is included as printable characters for easy recognition in a storage printout.

The HASP Vector Table ($HVT) contains a list of addresses used by OS/VS2 formal exit interface coding to pass control to HASP at various stages of I/O, job, or command processing. The address of $HVT is placed in the OS/VS2 Communication Vector Table (CVT) in a word named CVTHJES during HASP initialization. This indicates that HASP is active.

Direct branch entry to most HASP service subroutines is provided. Services macro-instructions in assemblies other than HASPNUC expand branches to symbolic locations in the HCT DSECT. Branches in the actual HCT transfer to the entry points of the services subroutines.

The HCT contains counts of active functions, jobs in execution, I/O requests, and commands to be processed. Status bytes record the operator options when HASP was last initialized and overall system status at the current time.

Many HASP queues and chains of control blocks have their beginning addresses in the HCT; for example, buffers, RJE buffers, Console Message Buffers, SMF buffers, Device Control Tables, Processor Control Elements, and I/O completion queues.

Addresses of several OS/VS2 nucleus routines, which are used by direct branch entry, are contained in the HCT.

The last part of the HCT is a group of HASP job queue pointers and direct-access information which is regularly written to the first HASP checkpoint record and restored on any warm start.

HASP COMMUNICATION TABLE

Offset	Bytes and Bits	Field Name	Description
00 (00)	8	$VERSION	HASP version
			HASP Vector Table
08 (08)	0	$HVT	Beginning of HASP Vector Table
08 (08)	4	$HVTEXCP	IOS pseudo device exit address
12 (0C)	4		Interpreter exit address
16 (10)	4		MGCR exit address
20 (14)	4		WTO(R)/WTL exit 1 address
24 (18)	4		WTO(R) exit 2 address
28 (1C)	4		Job initiation exit address
32 (20)	4		Step initiation exit address
36 (24)	4		Termination exit address
40 (28)	4		TSO status/cancel exit address
44 (2C)	2	$EXITNOP	An SR 15,15 instruction
46 (2E)	2		A BCR 15,14 instruction
			Entry To HASP Dispatcher
48 (30)	4	$WAIT	Vector to $WAIT routine
			Entries To HASP Buffer Services
52 (34)	4	$GETBUF	Vector to $GETBUF routine
56 (38)	4	$GETPBUF	Vector to $GETPBUF routine
60 (3C)	4	$FREEBUF	Vector to $FREEBUF routine
			Entries To HASP Unit Services
64 (40)	4	$GETUNIT	Vector to $GETUNIT routine
68 (44)	4	$FREUNIT	Vector to $FREUNIT routine
			Entries To HASP Job Queue Services
72 (48)	4	$QADD	Vector to $QADD routine
76 (4C)	4	$QGET	Vector to $QGET routine

Offset	Bytes and Bits	Field Name	Description
80 (50)	4	$QPUT	Vector to $QPUT routine
84 (54)	4	$QREM	Vector to $QREM routine
88 (58)	4	$QSIZ	Vector to $QSIZ routine
92 (5C)	4	$QLOC	Vector to $QLOC routine
96 (60)	4	$QJITLOC	Vector to $QJITLOC routine
			Entries To DASD Space Services
100 (64)	4	$TRACK	Vector to $TRACK routine
104 (68)	4	$PURGER	Vector to $PURGER routine
			Entries To HASP I/O Services
108 (6C)	4	$EXCP	Vector to $EXCP routine
112 (70)	4	$EXTPOPE	Vector to $EXTPOPE routine
116 (74)	4	$EXTPGET	Vector to $EXTPGET routine
120 (78)	4	$EXTPPUT	Vector to $EXTPPUT routine
124 (7C)	4	$EXTPCLO	Vector to $EXTPCLO routine
128 (80)	4	$RESTORE	Vector to $RESTORE routine
			Entries To HASP Overlay Services
132 (84)	4	$ODEL	Vector to $ODEL routine
136 (88)	4	$ORET	Vector to $ORET routine
140 (8C)	4	$OLINK	Vector to $OLINK routine
144 (90)	4	$OXCTL	Vector to $OXCTL routine
148 (94)	4	$OLOAD	Vector to $OLOAD routine
			Entries To HASP Console Services
152 (98)	4	$WTO	Vector to $WTO routine
156 (9C)	4	$FREEMSG	Vector to $FREEMSG routine
160 (A0)	4	$DOM	Vector to $DOM routine

Offset	Bytes and Bits	Field Name	Description
			Entries To HASP SMF Buffer Services
164 (A4)	4	$QUESMFB	Vector to $QUESMFB routine
168 (A8)	4	$GETSMFB	Vector to $GETSMFB routine
			Entries to HASP Timer Services
172 (AC)	4	$STIMER	Vector to $STIMER routine
176 (B0)	4	$TTIMER	Vector to $TTIMER routine
			Entries To HASP Error Services
180 (B4)	4	$IOERROR	Vector to $IOERROR routine
184 (B8)	4	$ERROR	Vector to $ERROR routine
188 (BC)	4	$DISTERR	Vector to $DISTERR routine
			Miscellaneous Control Fields
192 (C0)	1		Reserved
193 (C1)	1	$OPTSTAT	HASP initialization options
	1... ..xx	$OPTFMT	Format SYS1.HASPACE
	.1.. ..xx	$OPTCOLD	Cold start HASP
	..1. ..xx	$OPTREQ	HASP requests
	...1 ..xx	$OPTREP	REP cards used
	.... 1.xx	$OPTLIST	List REP cards
	.... .1xx	$OPTRACE	Trace
194 (C2)	1	$STATUS	HASP system status
	1... ...x	$RDRPEND	OS reader is pending
	.1.. ...x	$ALMSGSW	All avail. funct. message issued
	..1. ...x	$DRAINED	System has been $DRAINED
	...1 ...x	$CKPTACT	Checkpoint is in progress
	.... 1..x	$JITCKPT	Checkpoint Job Information Table
	.... .1.x	$SYSEXIT	HASP system in termination process
	.... ..1x	$JOTCKPT	Checkpoint Job Output Table
195 (C3)	1	$CURPCE	Current PCE ID
196 (C4)	1	$HASPECF	Master event control field
197 (C5)	1	MHASPECF	Line manager event control field
198 (C6)	1	$XEQACT	Count of jobs in OS execution
199 (C7)	1	$ACTIVE	Count of active functions

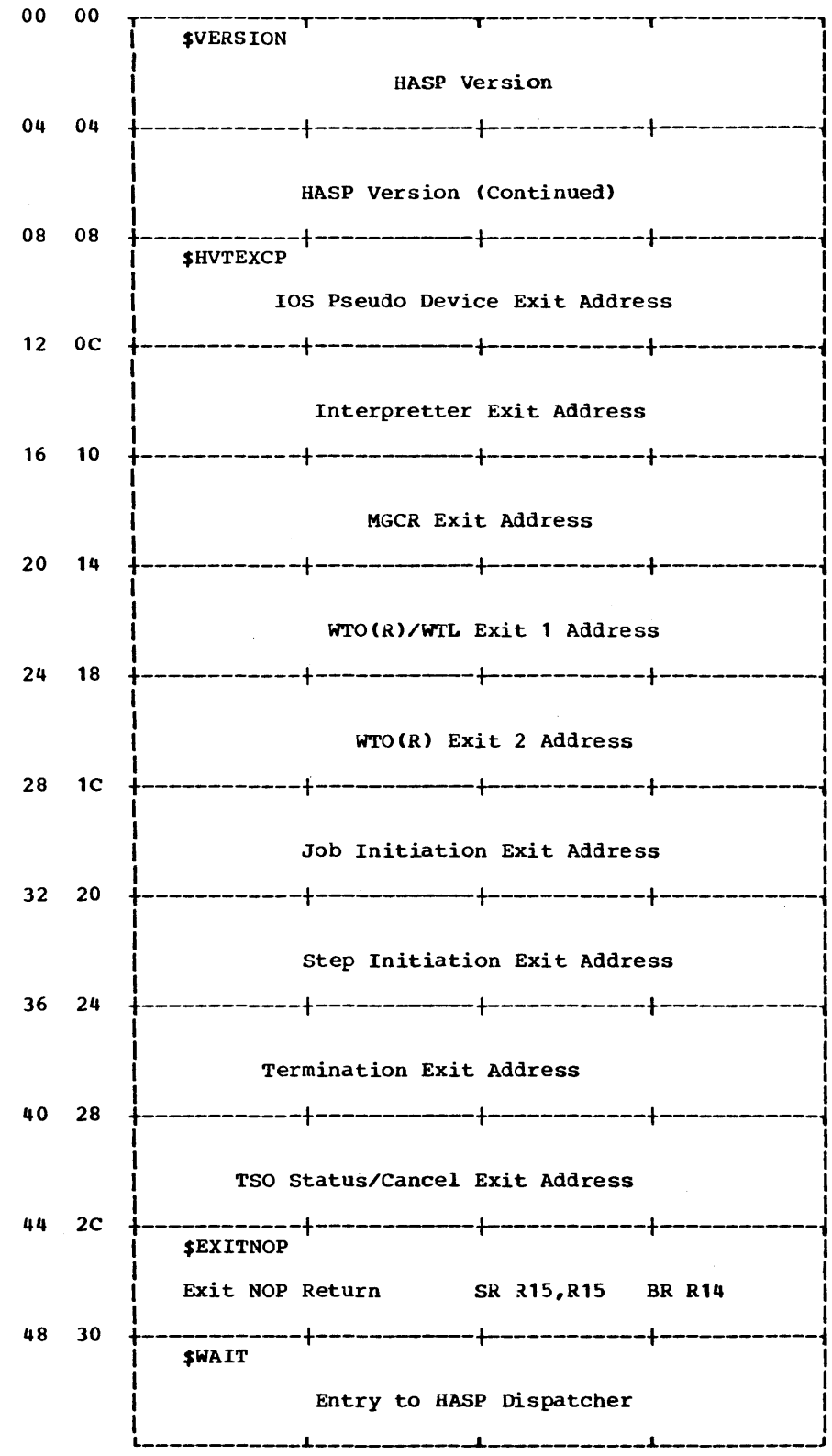
HASP COMMUNICATION TABLE

Offset	Bytes and Bits	Field Name	Description
200 (C8)	1	$ENBALL	Enable all mask
201 (C9)	1	$DISALL	Disable all mask
202 (CA)	1	$DISINT	Disable interval timer mask
203 (CB)	1		Reserved
204 (CC)	2	$EXCPCT	Active HASP I/O count
206 (CE)	2	$COMMCT	Active HASP command count
208 (D0)	2	$CKPTRAK	HASP checkpoint track address
210 (D2)	2		Reserved
212 (D4)	4	$HASPTCB	HASP task control block address
216 (D8)	4	$HASPECB	Common HASP ECB
220 (DC)	4	$ASYNCQ	ASYNC I/O completion queue
224 (E0)	4	$RJECHEQ	RJE I/O completion queue
228 (E4)	4	$PCEORG	First HASP PCE address
232 (E8)	4	$BUFPOOL	First available HASP buffer address
236 (EC)	4	$TPBPOOL	First available RJE buffer address
240 (F0)	4	$DCTPOOL	First HASP DCT address
244 (F4)	4	$JITABLE	HASP Job Information Table address
248 (F8)	4	$JOTABLE	HASP Job Output Table address
252 (FC)	4	$CYLMAP	First track group bit map address
256 (100)	4	$TEDADDR	First extent data table address
260 (104)	4	$DCBLIST	Direct-access DCB address
264 (108)	4	$FREEQUE	First free console msg buffer address
268 (10C)	4	$BUSYQUE	Console msg buffers queued for I/O
272 (110)	4	$LOGQUE	CMB's queued for Log Processor
276 (114)	4	$COMMQUE	CMD's queued for Command Processor

HASP COMMUNICATION TABLE

Offset	Bytes and Bits	Field Name	Description
280 (118)	4	$DOMQUE	CMB's awaiting action
284 (11C)	4	$SMFFREE	First free SMF buffer address
288 (120)	4	$SMFBUSY	SMF buffers queued for I/O
292 (124)	4	$PRCHKPT	Print Checkpoint Table address
296 (128)	4	$TIMEARG	Fake SVRB+36 for IGC011
			Nucleus Address Table Start
300 (12C)	4	$XFRHASP	Entry point HASP Task $XFER
304 (130)	4	$XFRSTPT	Entry point STPT Task $XFER
308 (134)	0	$NUCTABL	Nucleus Address Table start
308 (134)	4	$STATENT	Entry to status routine
312 (138)	4	$TIMENT	Entry to time routine
316 (13C)	4	$XSMFENT	Entry to SMF EXCP counting routine
320 (140)	4	$SVCRSET	Entry to HASP SVC reset routine
			Checkpointed Variables Restored On Any Warm Start
324 (144)	0	$SAVEBEG	Start of save area
324 (144)	4	$JOBQPTR	HASP job queue address
328 (148)	4	$JQFREE	Start of free queue chain
332 (14C)	4	$JQENT	Start of active queue chain
336 (150)	4	$DATAKEY	Master peripheral data set key
340 (154)	2	$JOBNO	HASP job number
342 (156)	2	$MSGRPNO	Last console msg track group
344 (158)	n	$DACKPT	DA checkpoint for warm start
	0	$SAVEEND	End of save area

HASP COMMUNICATION TABLE



HASP COMMUNICATION TABLE

DEC	HEX	
52	34	$GETBUF Entry to HASP Buffer 'GET' Routine
56	38	$GETPBUF Entry to HASP RJE Buffer 'GET' Routine
60	3C	$FREEBUF Entry to HASP Buffer 'FREE' Routine
64	40	$GETUNIT Entry to HASP Unit 'GET' Routine
68	44	$FREUNIT Entry to HASP Unit 'FREE' Routine
72	48	$QADD Entry to HASP Job Queue Element 'ADD' Routine
76	4C	$QGET Entry to HASP Job Queue Element 'GET' Routine
80	50	$QPUT Entry to HASP Job Queue Element 'PUT' Routine
84	54	$QREM Entry to HASP Job Queue Element 'REMOVE' Routine
88	58	$QSIZ Entry to HASP Job Queue 'SIZE' Routine
92	5C	$QLOC Entry to HASP Job Queue Element 'LOCATE' Routine
96	60	$QJITLOC Entry to HASP JIT Element 'LOCATE' Routine
100	64	$TRACK Entry to HASP Track Allocation Routine

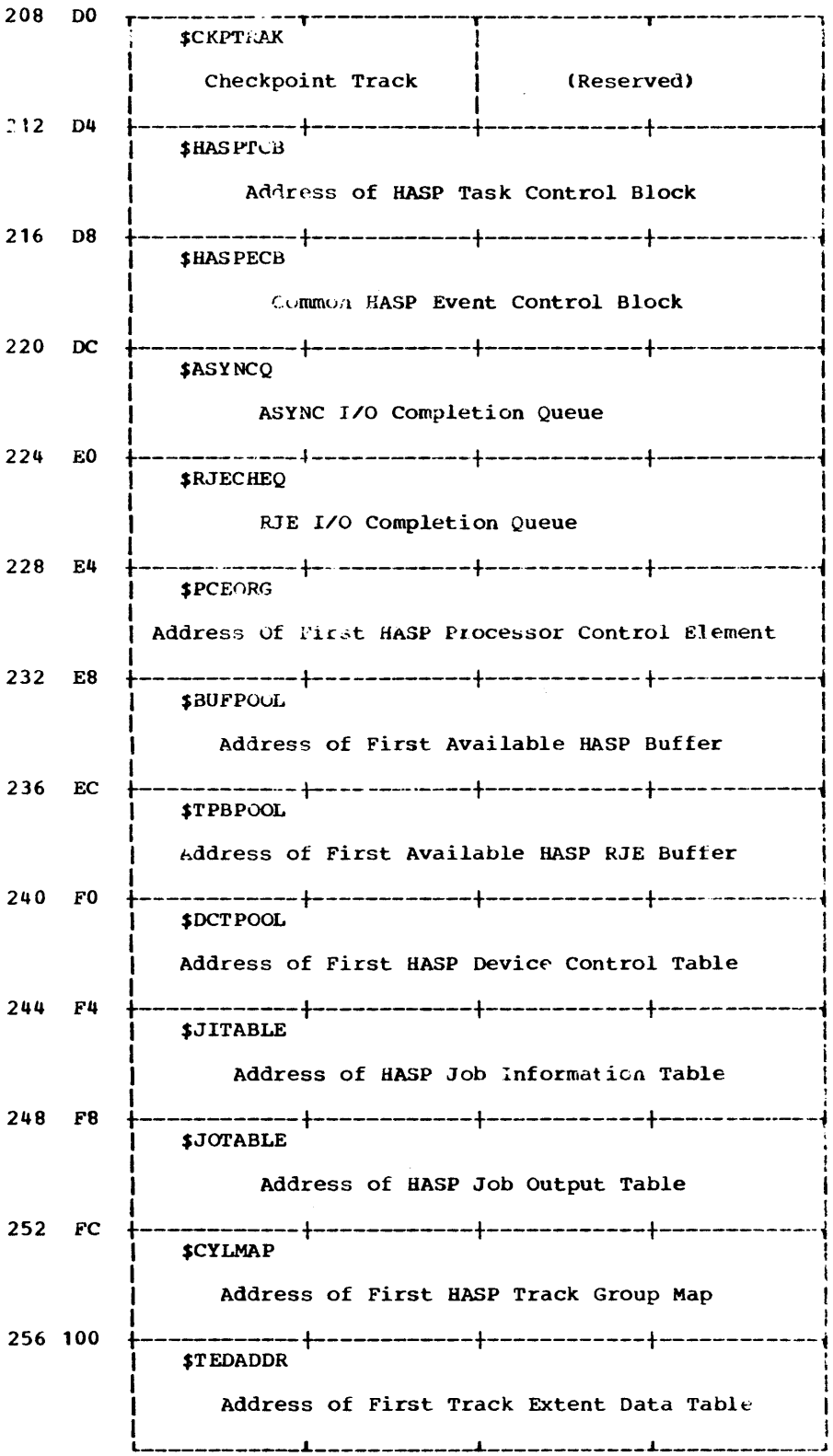
HASP COMMUNICATION TABLE

DEC	HEX		
104	68	$PURGER	Entry to HASP Track Purge Routine
108	6C	$EXCP	Entry to HASP Input/Output Supervisor
112	70	$EXTPOPE	Entry to HASP RTAM Open Routine
116	74	$EXTPGET	Entry to HASP RTAM GET Routine
120	78	$EXTPPUT	Entry to HASP RTAM PUT Routine
124	7C	$EXTPCLO	Entry to HASP RTAM Close Routine
128	80	$RESTORE	Entry to HASP RTAM Restore Routine
132	84	$ODEL	Entry to Overlay $DELETE Routine
136	88	$ORET	Entry to Overlay $RETURN Routine
140	8C	$OLINK	Entry to Overlay $LINK Routine
144	90	$OXCTL	Entry to Overlay $XCTL Routine
148	94	$OLOAD	Entry to Overlay $LOAD Routine
152	98	$WTO	Entry to HASP Write-to-Operator Routine

HASP COMMUNICATION TABLE

DEC	HEX				
156	9C	$FREEMSG Entry to Free HASP Console Message Buffer			
160	A0	$DOM Entry to Delete HASP Operator Message			
164	A4	$QUESMFB Entry to Queue HASP SMF Buffer Routine			
168	A8	$GETSMFB Entry to Remove SMF Buffer From $SMFREE Queue			
172	AC	$STIMER Entry to HASP Set Interval Timer Routine			
176	B0	$TTIMER Entry to HASP Test Interval Timer Routine			
180	B4	$IOERROR Entry to HASP Input/Output Error Logging Routine			
184	B8	$ERROR Entry to HASP Catastrophic Error Routine			
188	BC	$DISTERR Entry to HASP Disastrous Error Routine			
192	C0	Reserved	$OPTSTAT Init Options	$STATUS HASP Status	$CURPCE Current PCE ID
196	C4	$HASPECF Master Event Cntrl Field	MHASPECF RJE Event Cntrl Field	$XEQACT OS Exec Count	$ACTIVE Active Count
200	C8	$ENBALL Enable ALL Mask	$DISALL Disable ALL Mask	$DISINT Disable Int Timer Mask	Reserved
204	CC	$EXCPCT Active I/O Count		$COMMCT Active Command Count	

HASP COMMUNICATION TABLE



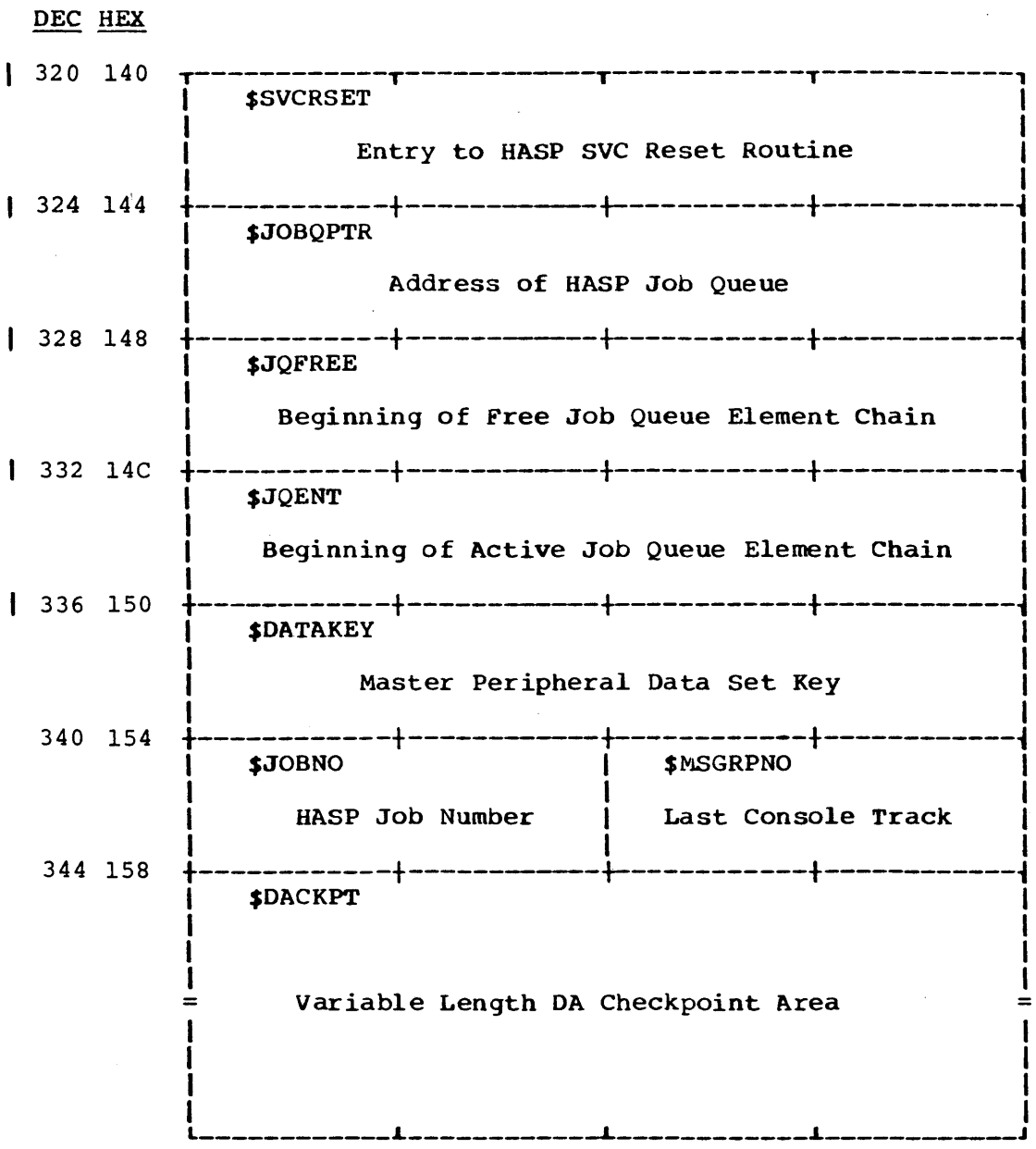
HASP COMMUNICATION TABLE

DEC	HEX	
260	104	$DCBLIST Address of Direct Access DCB
264	108	$FREEQUE Address of First Free HASP Console Message Buffer
268	10C	$BUSYQUE Console Message Buffers Queued for I/O
272	110	$LOGQUE Console Message Buffers Queued for Log Processor
276	114	$COMMQUE HASP Commands Queued for Command Processor
280	118	$DOMQUE Console Message Buffers Awaiting Operator Action
284	11C	$SMFFREE Address of First Free HASP SMF Buffer
288	120	$SMFBUSY HASP SMF Buffers Queued for I/O
292	124	$PRCHKPT Address of HASP Print Checkpoint Table
296	128	$TIMEARG Fake SVRB+36 for IGC011
300	12C	$XFRHASP Entry Point HASP task $XFER
304	130	$XFRSTPT Entry Point STPT task $XFER

HASP COMMUNICATION TABLE

DEC	HEX		
308	134	$STATENT	Entry to Status Routine
312	138	$TIMENT	Entry to Time Routine
316	13C	$XSMFENT	Entry to SMF EXCP Counting Routine

HASP COMMUNICATION TABLE

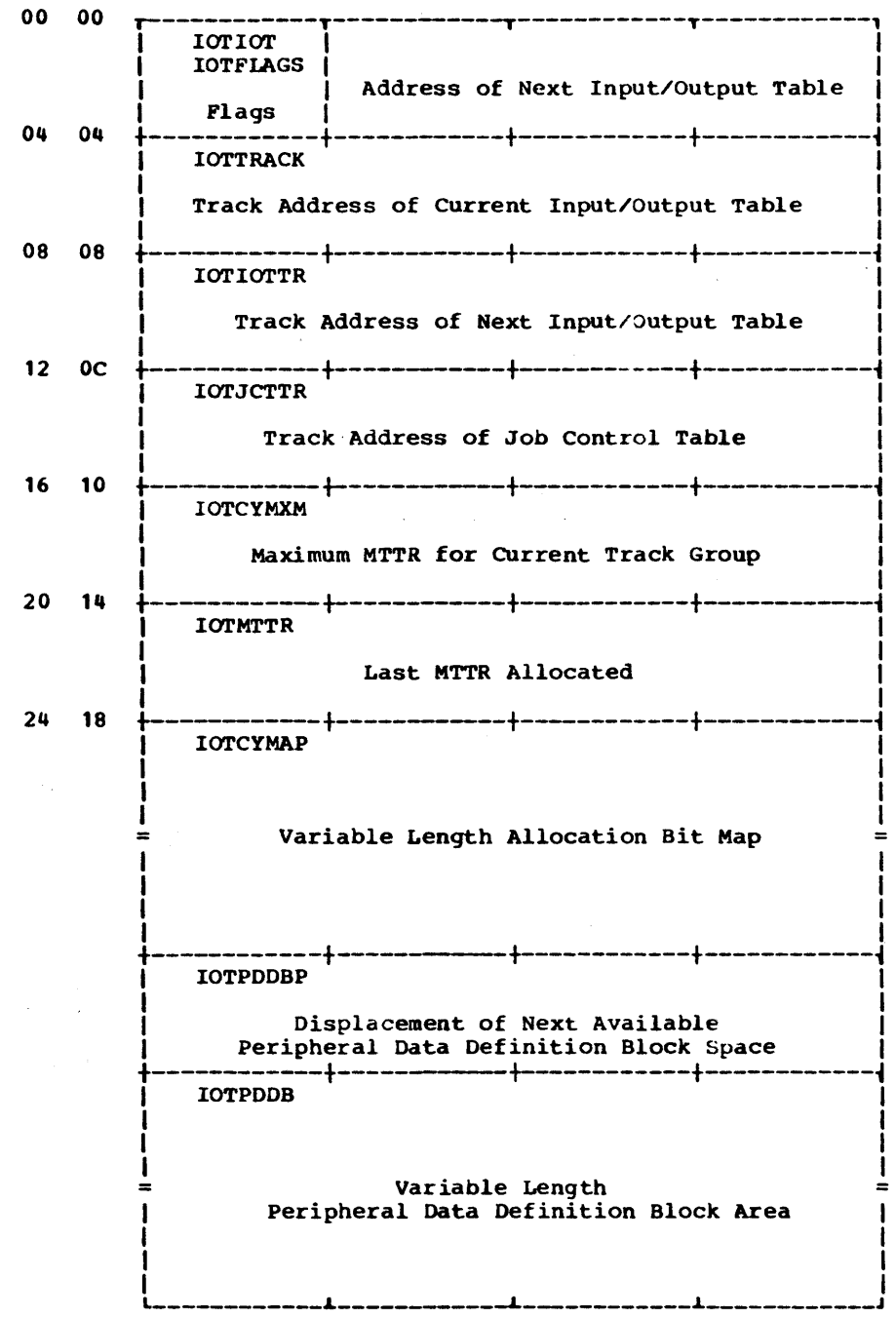


HASP INPUT/OUTPUT TABLE

The Input/Output Table (IOT) contains a record of allocated track groups for a job and a series of PDDBs which describe the job's output data sets or, before execution, the job's JCL file.

Offset		Bytes and Bits	Field Name	Description
88	(58)	1	IOTFLAGS	Miscellaneous flags
		1...	IOTWRITE	Checkpoint IOT
		.1..	IOTFLAG1	Reserved
		..1.	IOTFLAG2	Reserved
		...1	IOTFLAG3	Reserved
		.... 1...	IOTFLAG4	Reserved
		.... .1..	IOTFLAG5	Reserved
		.... ..1.	IOTFLAG6	Reserved
		1	IOTFLAG7	Reserved
88	(58)	4	IOTIOT	Address of next Input/Output Table
92	(5C)	4	IOTTRACK	Track address of current IOT
96	(60)	4	IOTIOTTR	Track address of next IOT
100	(64)	4	IOTJCTTR	Track address of Job Control Table
104	(68)	4	IOTCYMXM	Maximum MTTR for current track group
108	(6C)	4	IOTMTTR	Last MTTR allocated
112	(70)	Variable length	IOTCYMAP	Output allocation bit map
		4	IOTPDDBP	Offset of next available PDDB space
		4	IOTPDDB	Peripheral Data Definition Blocks

HASP INPUT/OUTPUT TABLE



HASP JOB CONTROL TABLE

The Job Control Table (JCT) is the primary job oriented control block. It is created by the HASP Input Service Processor in a HASP Buffer and written to disk. Other processors then read this control block into other HASP Buffers, use or update the information as necessary, and rewrite the control block to disk. The control block contains three types of information: (1) accounting information from the accounting field of the JOB card or from the JOBPARM control card, (2) accounting information gathered during job processing, and (3) the Input Track Group Map which represents all track groups used for JCL and input-stream data sets. This control block is the primary contributer to the HASP SMF Purge Record (Type 26).

Offset	Bytes and Bits	Field Name	Description
			Job Control Table
88 (58)	4	JCTJQE	HASP Job Queue Element offset
92 (5C)	4	JCTIOTTR	First IOT track address
96 (60)	4	JCTDSKEY	Peripheral data set key
100 (64)	4	JCTINJCT	Input JCT track address
104 (68)	1		Reserved
105 (69)	1	JCTSMFLG	SMF flags
106 (6A)	1	JCTPURGE	Start of SMF purge record
107 (6B)	1	JCTJBOPT	HASP job options
	1...	JCTPRICD	/*PRIORITY card present
	.1..	JCTSETUP	/*SETUP card(s) present
	..1.	JCTTHOLD	TYPRUN=HOLD was specified
	...1	JCTNOLOG	No job log option
	.... 1...	JCTXBACH	Execution batching job
108 (6C)	4	JCTJOBEB	HASP-assigned job number (EBCDIC)
112 (70)	8	JCTJNAME	Job name from JOB card
120 (78)	20	JCTPNAME	Programmer name from JOB card
140 (8C)	1	JCTMCLAS	MSGCLASS from JOB card

HASP JOB CONTROL TABLE

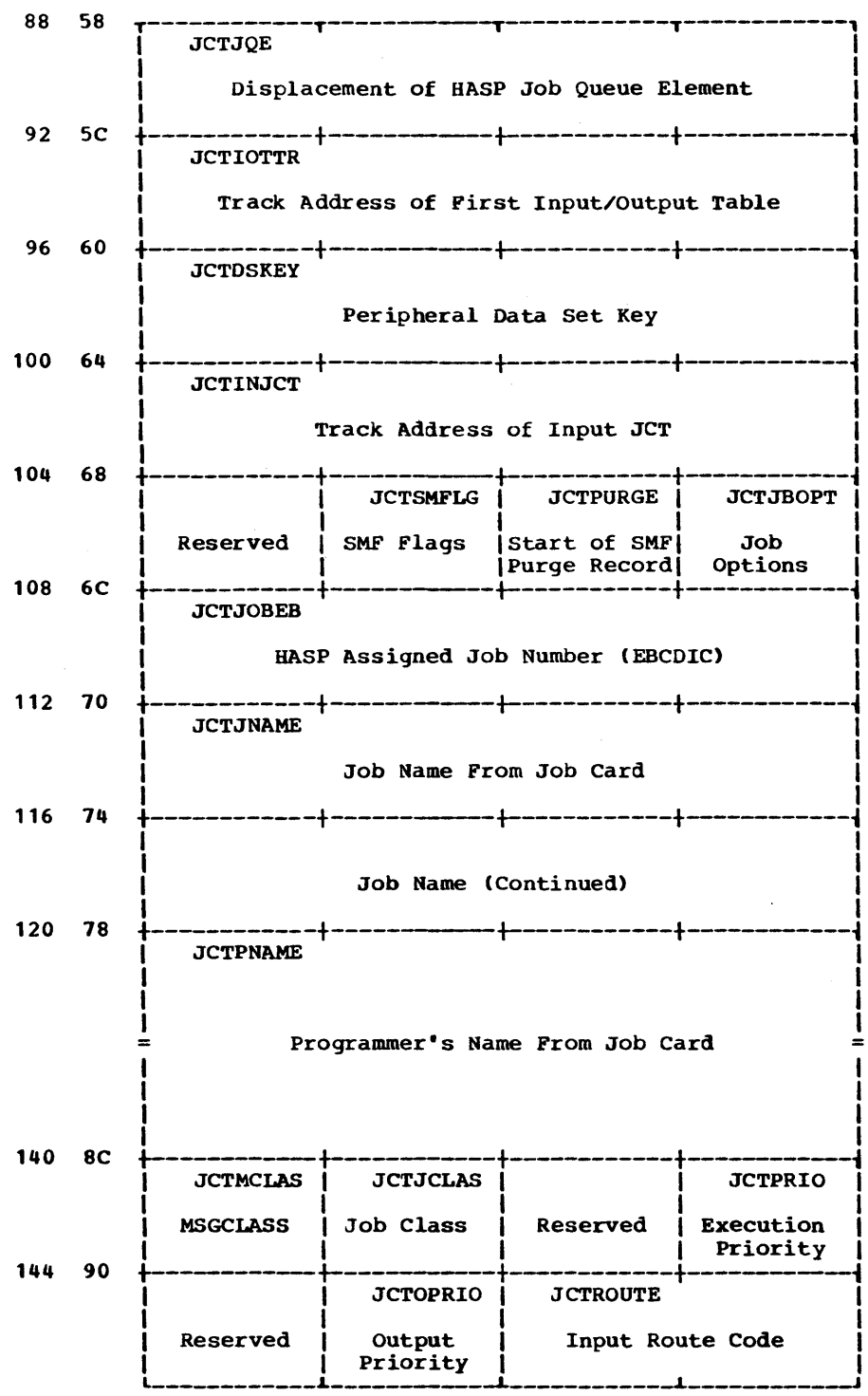
Offset	Bytes and Bits	Field Name	Description
141 (8D)	1	JCTJCLAS	Job class from JOB card
142 (8E)	1		Reserved
143 (8F)	1	JCTPRIO	HASP execution selection priority
144 (90)	1		Reserved
145 (91)	1	JCTOPRIO	HASP output selection priority
146 (92)	2	JCTROUTE	Input route code
148 (94)	8	JCTINDEV	HASP input device name
156 (9C)	4	JCTACCTN	Job accounting number from JOB card
160 (A0)	4	JCTROOMN	Programmer room number
164 (A4)	4	JCTETIME	Estimated execution time
168 (A8)	4	JCTESTLN	Estimated output lines
172 (AC)	4	JCTESTPU	Estimated punched output
176 (B0)	4	JCTFORMS	Job output forms
180 (B4)	1		Reserved
181 (B5)	1	JCTCPYCT	Job print copy count
182 (B6)	1		Reserved
183 (B7)	1	JCTLINCT	Lines per page
184 (B8)	2	JCTPROUT	Job print route code
186 (BA)	2	JCTPUOUT	Job punch route code
188 (BC)	4	JCTXEQON	Time on Execution Processor
192 (C0)	4	JCTXDTON	Date on Execution Processor
196 (C4)	4	JCTXEQOF	Time off Execution Processor
200 (C8)	4	JCTXDTOF	Date off Execution Processor
204 (CC)	4	JCTOUTON	Time on Output Processor
208 (D0)	4	JCTODTON	Date on Output Processor
212 (D4)	4	JCTOUTOF	Time off Output Processor

Offset	Bytes and Bits	Field Name	Description
216 (D8)	4	JCTODTOF	Date off Output Processor
220 (DC)	4	JCTCARDS	Total number of input cards
224 (E0)	4	JCTLINES	Generated output lines
228 (E4)	4	JCTPUNCH	Generated punched output
232 (E8)	4		Reserved
236 (EC)	4	JCTPRTCT	Current number of lines printed
240 (F0)	4	JCTPAGCT	Current number of pages printed
244 (F4)	4	JCTPUNCT	Current number of cards punched
248 (F8)	4	JCTESOUT	Estimated output (lines and cards)
252 (FC)	4	JCTXOUT	Generated output records
256 (100)		JCTCYSAV	Variable length input allocation map
	144	JCTWORK	144-byte work area
	56	JCTJMR	JMR area
	8	JCTJMRJN	JMR job name
	4	JCTRDRON	Time on Input Processor
	4	JCTRDTON	Date on Input Processor
	4	JCTCPUID	JMR CPU identification
	8	JCTUSEID	JMR user identification
	1	JCTSTEP	Current step number
	1	JCTINDC	JMR SMF options
	2		Reserved
	4	JCTUCOM	JMR user communication area
	4	JCTUJVP	JMR address of user exit routine
	4	JCTRDROF	Time off Input Processor
	4	JCTRDTOF	Date off Input Processor
	4	JCTJOBIN	JMR job SYSIN count

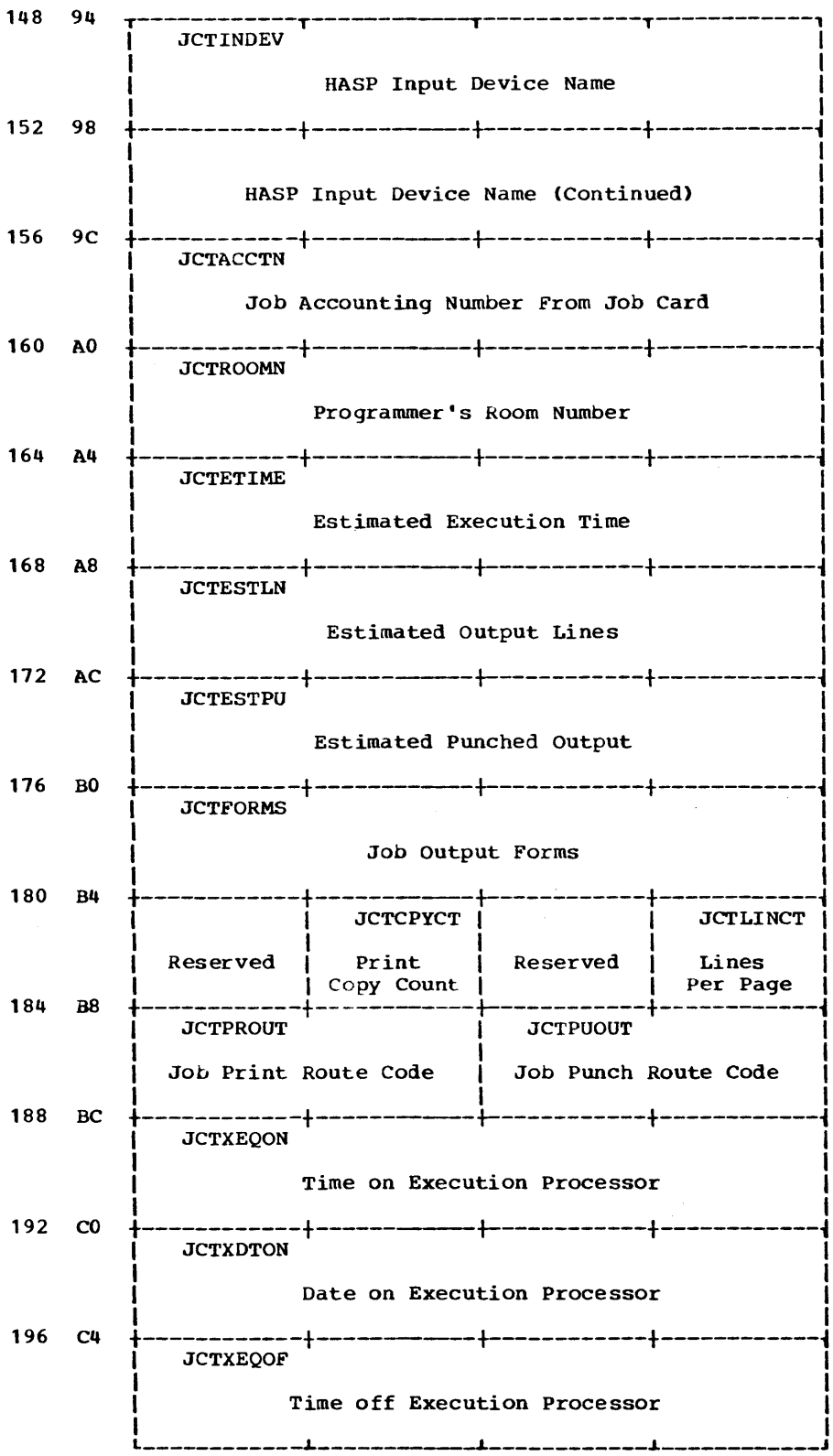
HASP JOB CONTROL TABLE

Bytes and Bits	Field Name	Description
2	JCTRDR	Reader device type and class
1	JCTJMOPT	JMR SMF options
1		Reserved
0	JCTEND	End of JCT

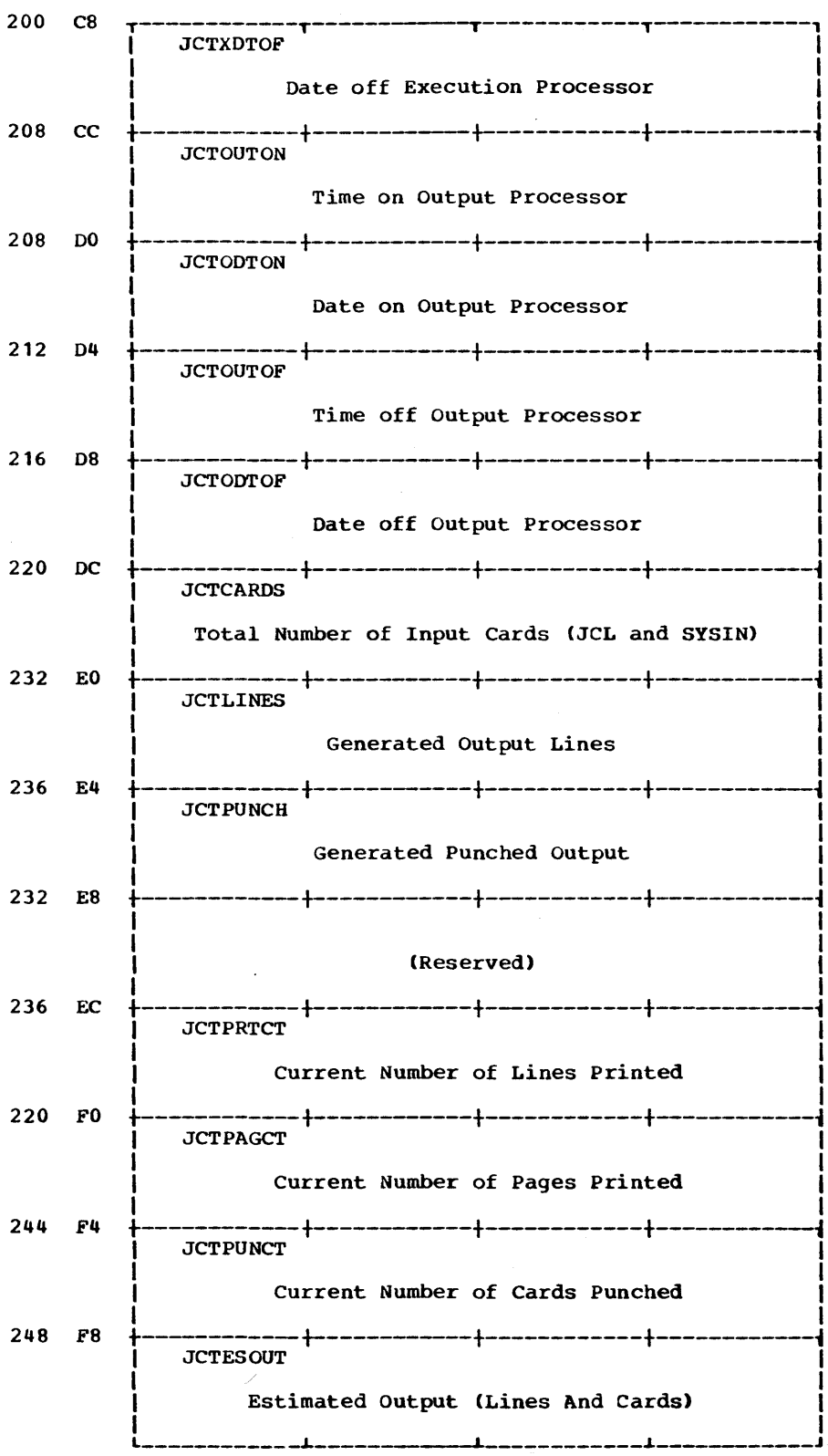
HASP JOB CONTROL TABLE



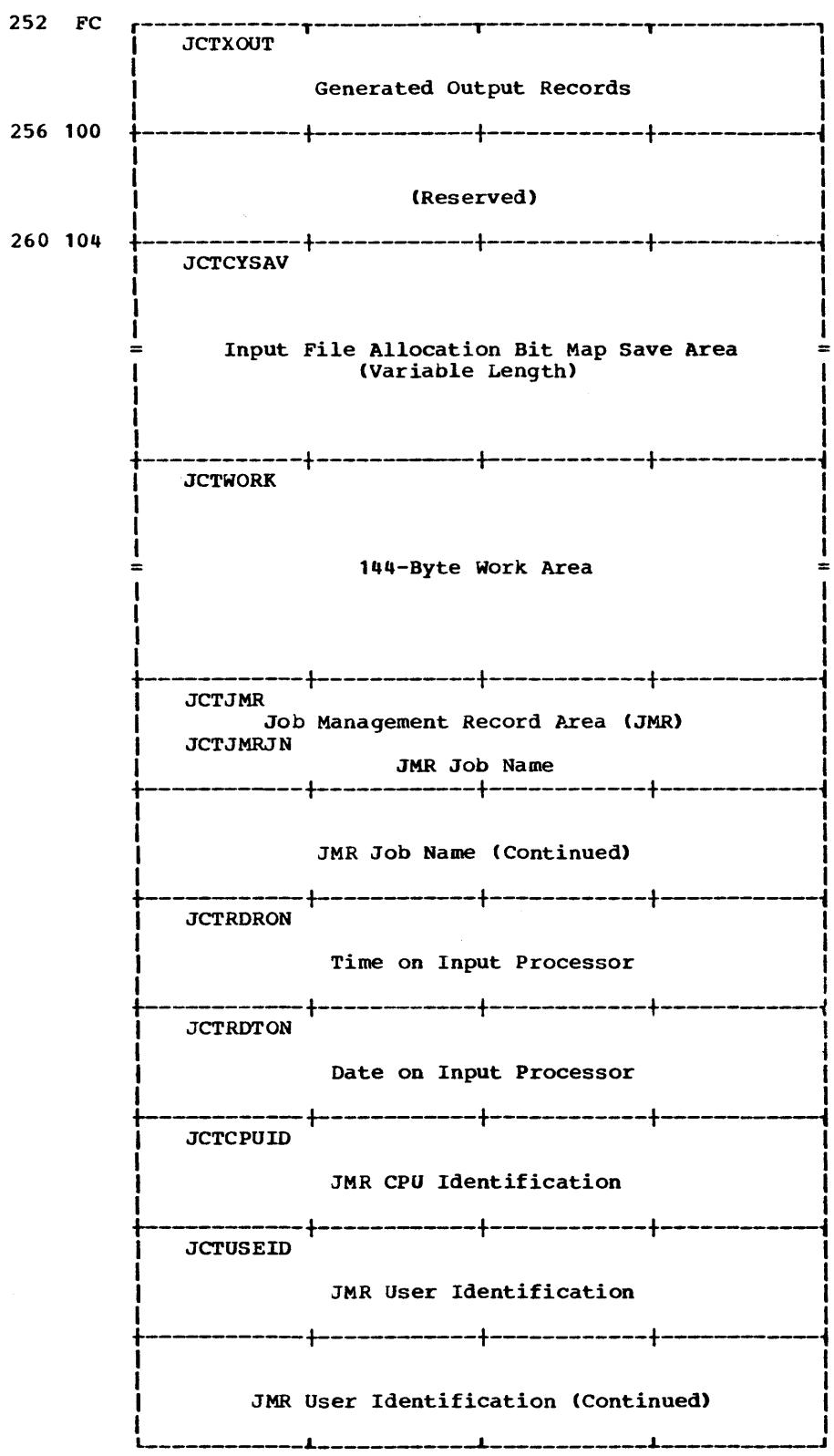
HASP JOB CONTROL TABLE



HASP JOB CONTROL TABLE

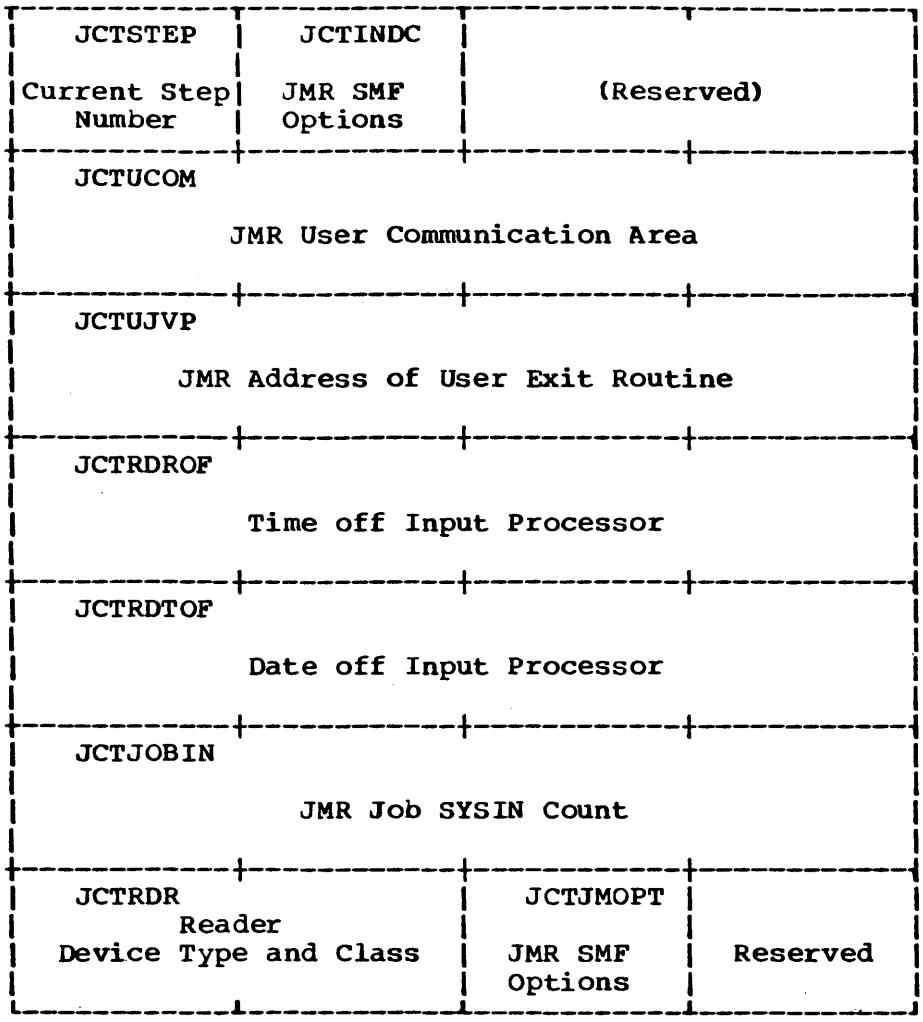


HASP JOB CONTROL TABLE



HASP JOB CONTROL TABLE

DEC HEX



HASP JOB INFORMATION TABLE

Offset	Bytes and Bits	Field Name	Description
0 (0)	8	JITJNAME	Job Name

The Job Information Table (JIT) allows the operator to communicate with HASP concerning a job by OS job name. There are physically the same number of entries in the JIT as in the JQE table. The relationship is a one-to-one correspondence to the JQE table; that is, the first physical entry in the JIT is associated with the first entry in the JQE table, the second with the second, etc.

The symbolic name of the JIT (displacement '00') is JITJNAME. Each entry is 8 bytes long and contains a job name.

HASP JOB OUTPUT ELEMENT

The Job Output Element (JOE) is constructed by the HASP Output Processor after a job has been placed in the HASP output queue. The JOE can serve one of three functions:

1. As a work element, the JOE relates an item of print/punch activity to its HASP job control blocks.
2. As a characteristics element, the JOE relates an item of print/punch activity to device setup requirements of other items.
3. As a checkpoint element, the JOE saves the status of partially completed print/punch activities for restart.

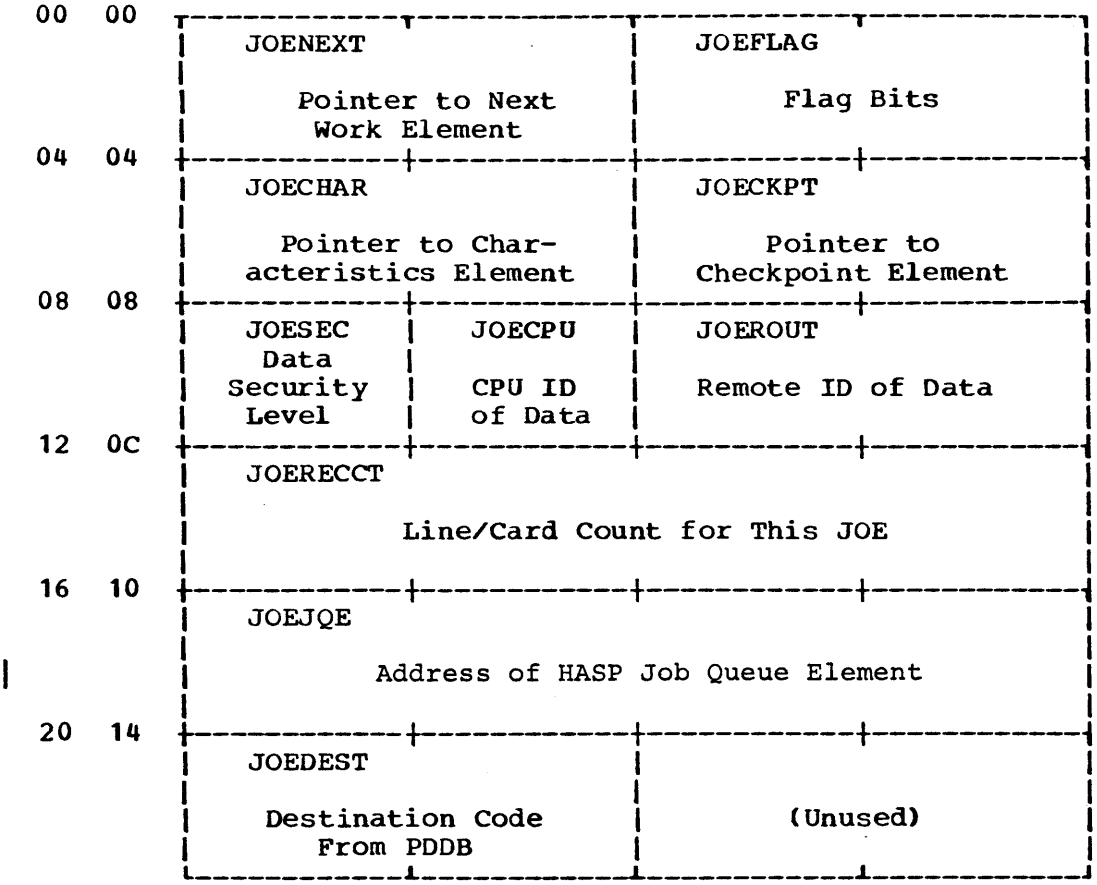
All JOEs reside in the JOB Output Table prior to and during actual device processing. The next figure illustrates the format of a JOE with descriptions of the fields.

Offset	Bytes and Bits	Field Name	Description
			Work Element
0 (0)	2	JOENEXT	PTR to next work element
2 (2)	2	JOEFLAG	Flag bits
	Byte 0		
	1.xx xxxx	$JOEBUSY	Busy
	.1xx xxxx	$JOECKV	Checkpoint element valid flag
	Byte 1		
	xxxx xxxx		Reserved
4 (4)	2	JOECHAR	Characteristics element pointer
6 (6)	2	JOECKPT	Checkpoint element pointer
8 (8)	1	JOESEC	Security level of data
9 (9)	1	JOECPU	CPU ID of data
10 (A)	2	JOEROUT	Remote ID of data
	Byte 0		
	1xxx xxxx	$JOEIRTE	Implicit routing
	Byte 1		
	xxxx xxxx		Reserved

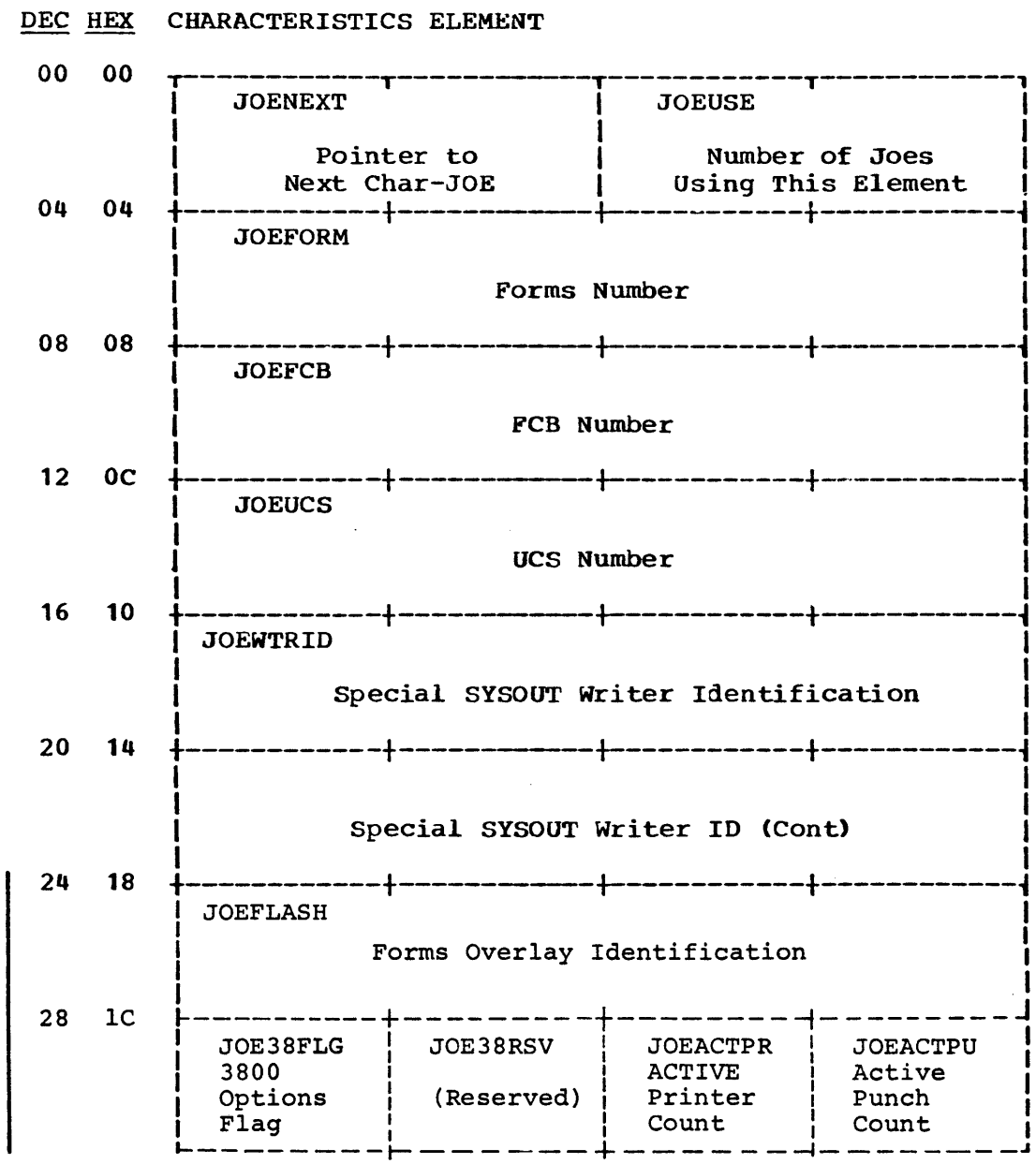
HASP JOB OUTPUT ELEMENT

Offset	Bytes and Bits	Field Name	Description
12 (C)	4	JOERECCT	Copies and maxlines/cards for this JOE
16 (10)	4	JOEJQE	Address of HASP Job Queue Element
20 (14)	2	JOEDEST	Destination code from PDDB
			Characteristics Element
0 (0)	2	JOENEXT	PTR to next characteristics element
2 (2)	2	JOEUSE	# of JOEs using this element
4 (4)	4	JOEFORM	Forms number
8 (8)	4	JOEFCB	FCB number
12 (C)	4	JOEUCS	UCS number
16 (10)	8	JOEWTRID	Special SYSOUT writer identification
24 (18)	4	JOEFLASH	Forms overlay identification
28 (1C)	1 1xxx xxxx	JOE38FLG JOEBURST	3800 options flag Burster-Trimmer-Stacker threading
29 (ID)	1	JOE38RESV	Reserved
30 (1E)	1	JOEACTPR	Number of active printers
31 (1F)	1	JOEACTPU	Number of active punches
			Checkpoint Element
0 (0)	1 1.xx xxxx .1xx xxxx	JOECKFLG PRCHKUSE PRCHKJOB	Checkpoint flags Checkpoint entry is in use Job active indicator
1 (1)	1	JOECOPY	Copy number in progress
2 (2)	2	JOEJRCB	Offset into eject buffer
4 (4)	2	JOEPDDB	Offset of PDDB in IOT
6 (6)	2	JOEPPCT	PDDB page count
8 (8)	4	JOETLNC	Total JOE line count
12 (C)	4	JOETPCT	Total JOE page count
16 (10)	4	JOEMTTR	Last eject buffer track address
20 (14)	4	JOEIOTTR	Current IOT track address
24 (18)	4	JSMF6PGE	Page Count of last SMF Record Type-6
28 (1C)	4	JSMF6NLR	Line Count of last SMF Record Type-6
32 (20)	1	JJNDS	Number of data sets processed
33 (21)	1	JSTCNR	Starting Copy Number
34 (22)	2	--	Unused

HASP JOB OUTPUT ELEMENT

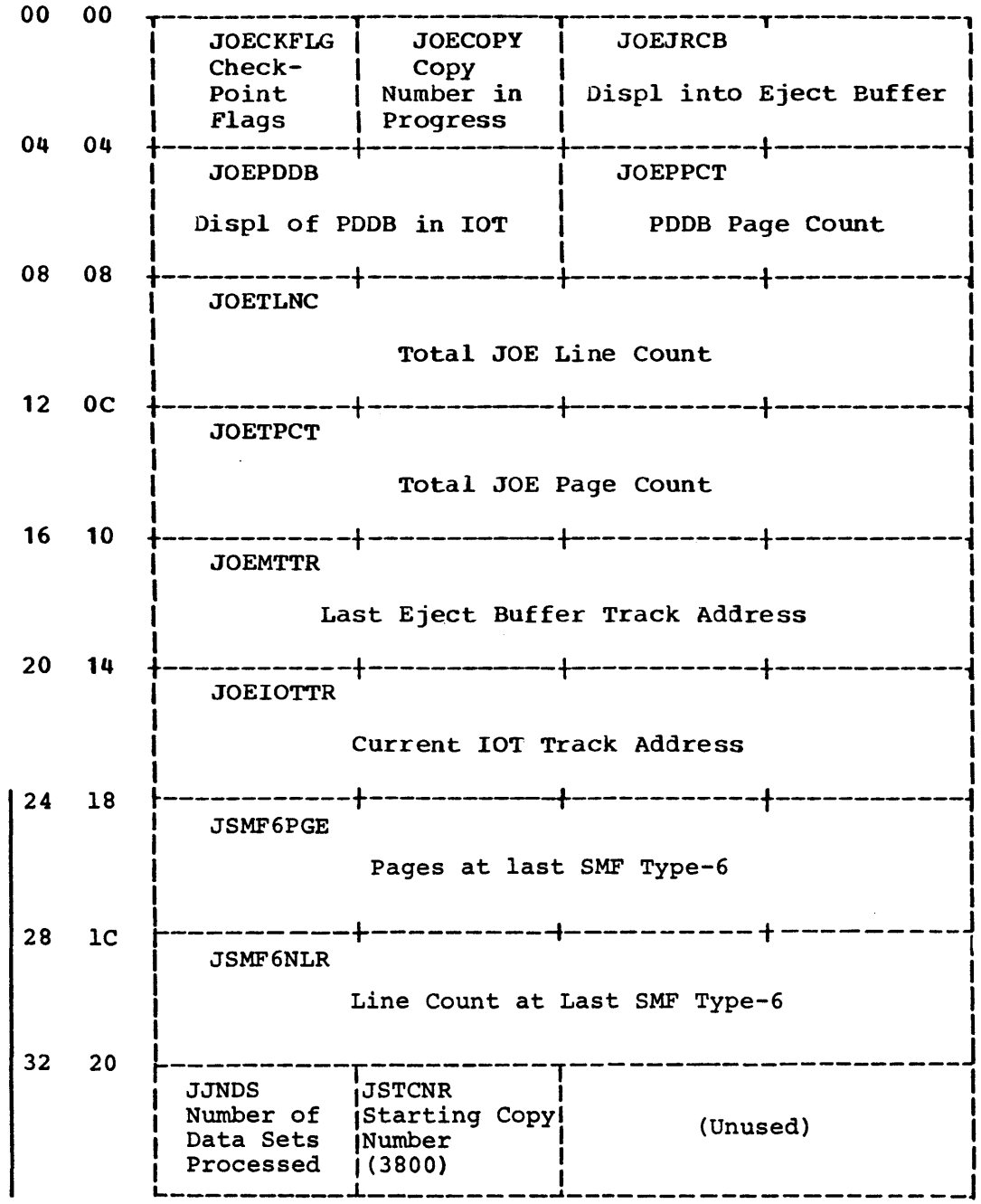


HASP JOB OUTPUT ELEMENT



HASP JOB OUTPUT ELEMENT

CHECKPOINT ELEMENT

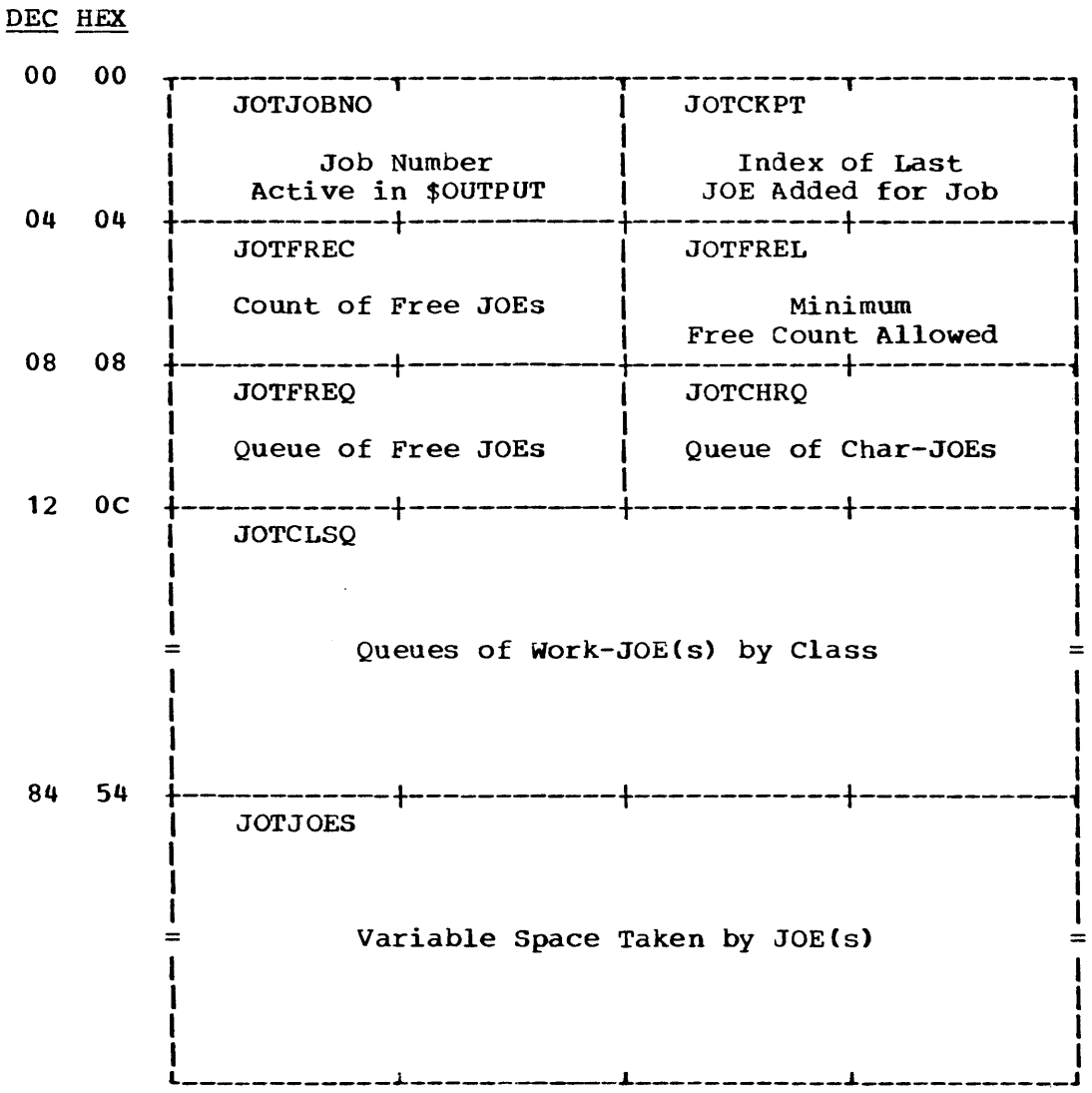


HASP JOB OUTPUT TABLE

The JOB Output Table (JOT) is maintained by a set of service routines which add, remove, and alter class queue entries. Each class queue entry represents an item of print/punch work. Thirty-six class queues are supported (A-Z, 0-9) with an additional queue of characteristics elements to facilitate work assignment to Print/Punch Processors. Following a fixed section, the JOT is composed of JOEs which are described in another figure. The JOT is checkpointed on demand of the service routines after every change to allow warm start after a system crash or a planned shutdown.

Offset	Bytes and Bits	Field Name	Description
00 (00)	2	JOTJOBNO	Job number active in $OUTPUT
02 (02)	2	JOTCKPT	Index of last JOE added for job
04 (04)	2	JOTFREC	Count of free JOEs
06 (06)	2	JOTFREL	Minimum free count allowed
08 (08)	2	JOTFREQ	Queue of free JOEs
10 (0A)	2	JOTCHRQ	Queue of char-JOEs
12 (0C)	72	JOTCLSQ	Queues of work-JOEs by class
84 (54)		JOTJOES	Start of JOEs
84 (54)	44		Space taken by JOEs

HASP JOB OUTPUT TABLE



HASP JOB QUEUE ELEMENT

The HASP Job Queue consists of several (&MAXJOBS HASPGEN parameter) Job Queue Elements (JQEs) in consecutive virtual storage.

Each JQE is on either the active chain, representing a job in the system, or on the free chain, available for new jobs which enter the system. The JQEs on the active chain are maintained in priority order and in one of five logical queues as indicated by a type field, with a bit to indicate if the job is currently being processed.

The JQE contains the most basic information about the job such as priority, number, output routings, etc. It also contains the direct-access address of the Job Control Table (JCT) for the job, which contains a much more comprehensive collection of information about the job and HASP processing of it.

HASP processors access JQEs using the Job Queue Services macro instructions $QADD, $QGET, $QPUT, $QREM, $QSIZ, and $QLOC. The Checkpoint Processor writes a copy of the Job Queue to direct access when the status of anything in it is changed.

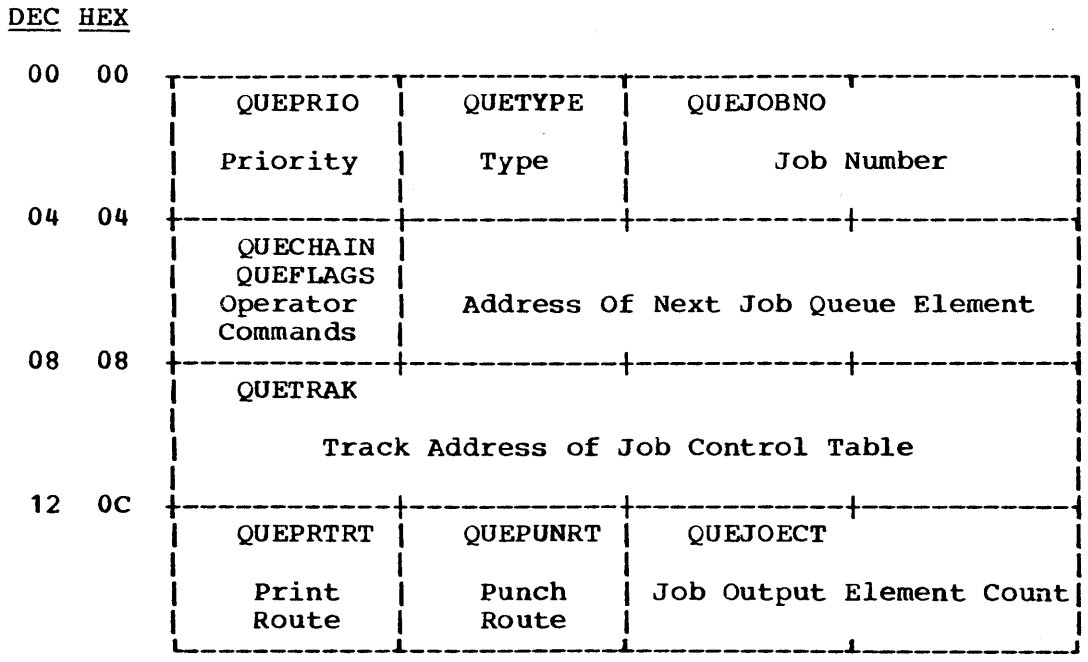
Other information about the job is maintained in virtual storage in the Job Information Table (JIT), and also in the Job Output Table (JOT) when the JQE indicates that the job is in the hardcopy logical queue.

Offset	Bytes and Bits	Field Name	Description
00 (00)	1	QUEPRIO	Job priority
01 (01)	1	QUETYPE	Logical queue type
	1...	QUETBY	Queue entry busy bit
	.1..	$XEQ	OS execution queue
	..1.	$INPUT	Input queue
	...1	$SETUP	Setup queue
	.... 1...	$BRKDOWN	Breakdown queue
	.... .1..	$OUTPUT	Output queue
	.... ..1.	$HARDCPY	Output in-progress queue
	1	$PLOT	Plot queue
	0000 0000	$PURGE	Purge queue
02 (02)	2	QUEJOBNO	HASP job number
04 (04)	1	QUEFLAGS	Job queue flags
	1... ...x	QUEHOLDA	Hold all jobs
	.1.. ...x	QUEHOLD1	Hold single job
	..1. ...x	QUEHOLD2	Hold for duplicate job name
	...1 ...x	QUEPURGE	Job is to be purged
	.... 1..x	QUEOPCAN	Operator issued $C or $P job
	.... .1.x	QUEOPRTE	Operator has issued $R all
	.... ..1x	QUEJCTSW	Interlock JCT for update

HASP JOB QUEUE ELEMENT

Offset	Bytes and Bits	Field Name	Description
04 (04)	4	QUECHAIN	Next Job Queue Element address
08 (08)	4	QUETRAK	Job Control Table track address
12 (0C)	1	QUEPRTRT	Print route
13 (0D)	1	QUEPUNRT	Punch route
14 (0E)	2	QUEJOECT	Job Output Element count

HASP JOB QUEUE ELEMENT

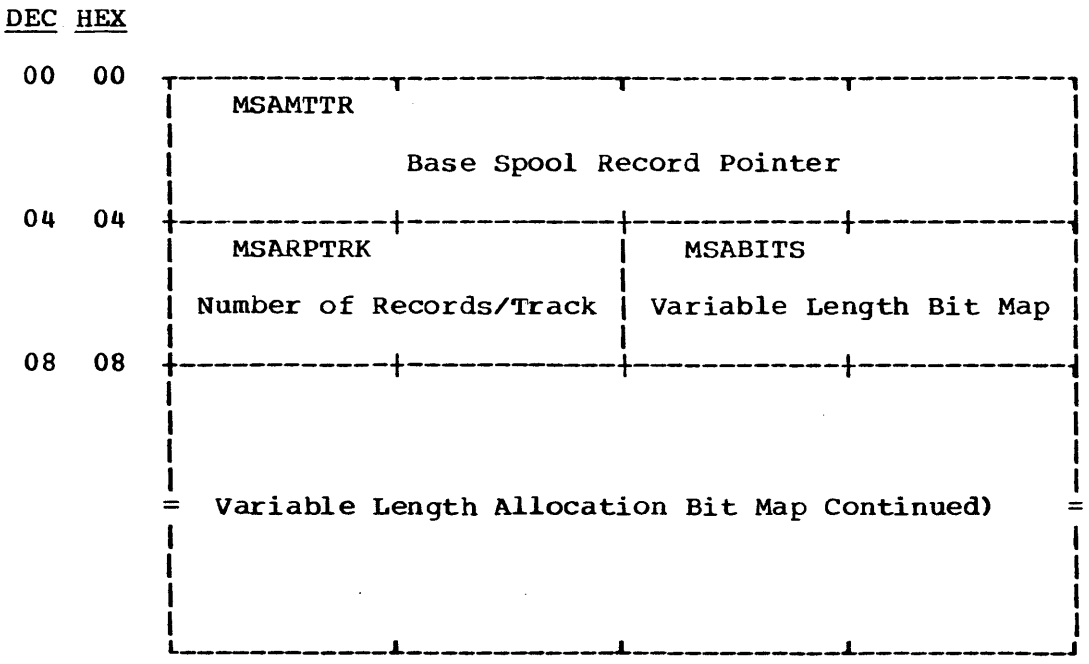


HASP MESSAGE SPOOLING ALLOCATION BLOCK

The Message SPOOLing Allocation Control Block (MSA) is used to allocate data records on a record basis from a dedicated area of the primary SPOOL volume. The base MTTR and number of records per track are set at HASP initialization time. Bits in the allocation map are reset by the HASP Remote Console Processor as records are allocated and then written to the SPOOL volume. Bits are set on by the HASP Print Processor after records are read and deallocated. The control block is located at entry $MSALLOC in the HASPRTAM assembly.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	MSAMTTR	The HASP MTTR for the first record of the message SPOOLing area on the primary SPOOL volume (used in converting a bit position in MSABITS to MTTR and vice versa)
04 (04)	2	MSARPTRK	The number of records per track on the primary SPOOL volume (used in converting a bit position in MSABITS to MTTR and vice versa)
06 (06)	n	MSABITS	Bit map indicating the status of records in the message SPOOLing area in the primary SPOOL volume. Bits that are on indicate that the corresponding record is available.
	0	MSABITL	Length of bit map (EQU)

HASP MESSAGE SPOOLING ALLOCATION BLOCK



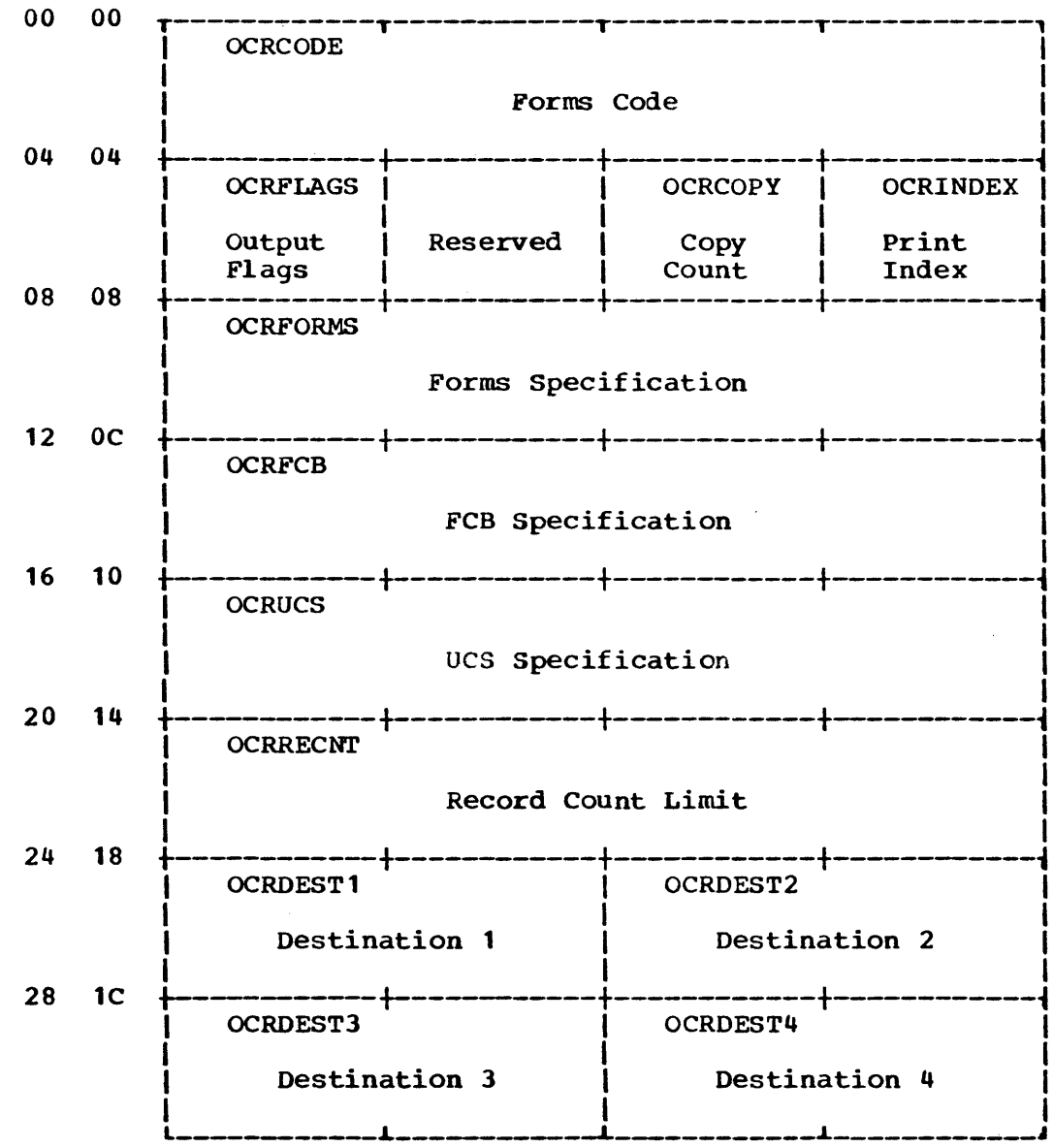
HASP OUTPUT CONTROL RECORD

The Output Control Record is a fixed-format record which represents the data which can be punched on an OUTPUT control card. It is created by the Input Service Processor and added to the JCL file as an 80-character record, identified as an OCR by a record type code of X'43'. At the time the Execution Service Processor is passing the JCL file to the OS Reader/Interpreter, the OCR is discovered and saved in a special buffer which is used later in the construction of the PDDBs for the referenced SYSOUT data sets.

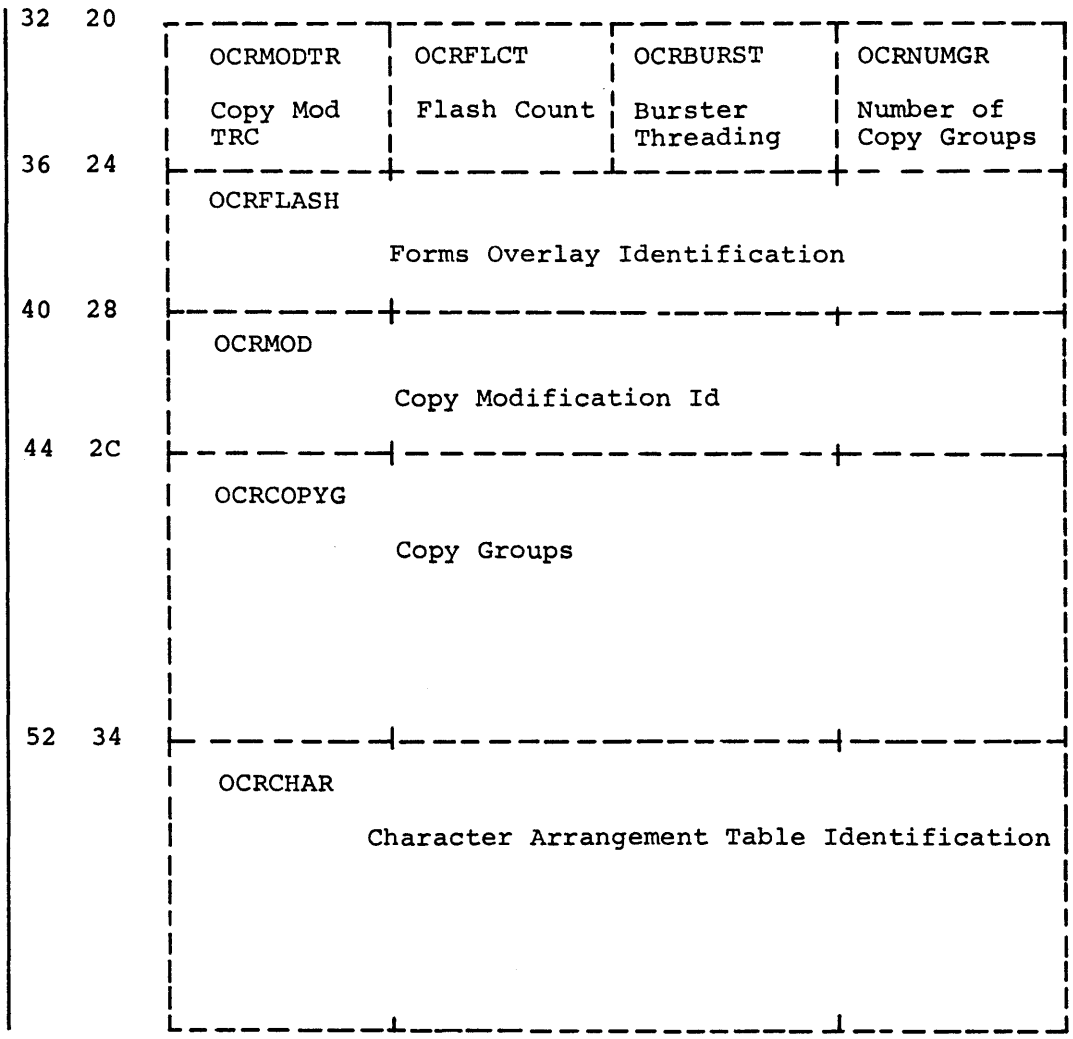
Offset	Bytes and Bits	Field Name	Description
00 (00)	4	OCRCODE	Forms code
04 (04)	1	OCRFLAGS	Output flags
	1...	OCRFLAG0	Reserved
	.1..	OCRFLAG1	Reserved
	..1.	OCRFLAG2	Reserved
	...1	OCRFLAG3	Reserved
	.... 1...	OCRFLAG4	Reserved
	.... .1..	OCRFLAG5	Reserved
	.... ..1.	OCRFLAG6	Reserved
	1	OCRFLAG7	Reserved
05 (05)	1		Reserved
06 (06)	1	OCRCOPY	Copy count
07 (07)	1	OCRINDEX	Print index
08 (08)	4	OCRFORMS	Forms specification
12 (0C)	4	OCRFCB	FCB specification
16 (10)	4	OCRUCS	UCS specification
20 (14)	4	OCRRECNT	Record count limit
24 (18)	2	OCRDEST1	Destination 1
26 (1A)	2	OCRDEST2	Destination 2
28 (1C)	2	OCRDEST3	Destination 3
30 (1E)	2	OCRDEST4	Destination 4
32 (20)	1	OCRMODTR	Copy Modification Table Reference Character
33 (21)	1	OCRFLCT	Flash Count
34 (22)	1	BURST	Burster-Trimmer-Stacker Threading (Y/N)
35 (23)	1	OCRNUMGR	Number of Copy Groups

HASP OUTPUT CONTROL RECORD

Offset	Bytes and Bits	Field Name	Description
36 (24)	4	OCRFLASH	Forms Overlay Identification
40 (28)	4	OCRMOD	Copy Modification Identification
44 (2C)	8	OCRCOPYG	Copy Groupings
52 (34)	16	OCRCHAR	Character Arrangement Table Identification
68 (44)	0	ORCEND	End of HASP Output Control Record



HASP OUTPUT CONTROL RECORD



HASP OVERLAY TABLE

The HASP Overlay Table is created as a new CSECT (HASPOTAB) by the Overlay Build Utility Program (HASPOBLD) and passed to the OS Linkage Editor for inclusion in the HASP load module. It is used by the Overlay Service routines to locate overlays on direct access during HASP operation.

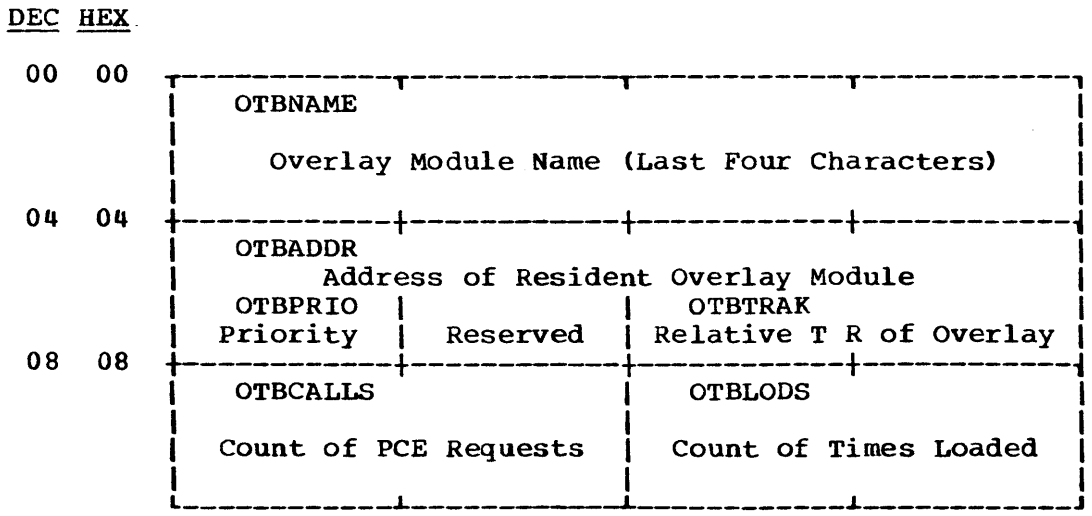
The Overlay Table is a series of fixed length entries (4 bytes each, or 12 bytes each if &DEBUG=YES), one for each overlay CSECT in HASP. A table entry is located by multiplying the overlay calling constant assigned to the overlay (see listing provided by the Overlay Build Utility, the column labeled OCON) by 4 or 12 and adding this to the storage address of HASPOTAB.

Each entry contains the storage address of the overlay (if permanently resident) or the priority and relative T R direct-access address in the overlay data set. If &DEBUG=YES, each entry also contains the printable name of the overlay and two statistical counts concerning usage of the overlay during HASP operation.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	OTBNAME	Overlay module name (last 4 chars.)
04 (04)	4	OTBADDR	Address of resident overlay module
04 (04)	1	OTBPRIO	Priority of nonresident overlay or X'FF' if resident
05 (05)	1		Reserved
06 (06)	2	OTBTRAK	Relative disk T R of overlay module
08 (08)	2	OTBCALLS	Number of PCE requests to use module
10 (0A)	2	OTBLODS	Number of times module read from DA

NOTE: The above definition is correct if the HASPGEN parameter &DEBUG=YES. If &DEBUG=NO, then only fields OTBADDR, OTBPRIO, and OTBTRAK are present, at offsets 0, 0, and 2 respectively.

HASP OVERLAY TABLE



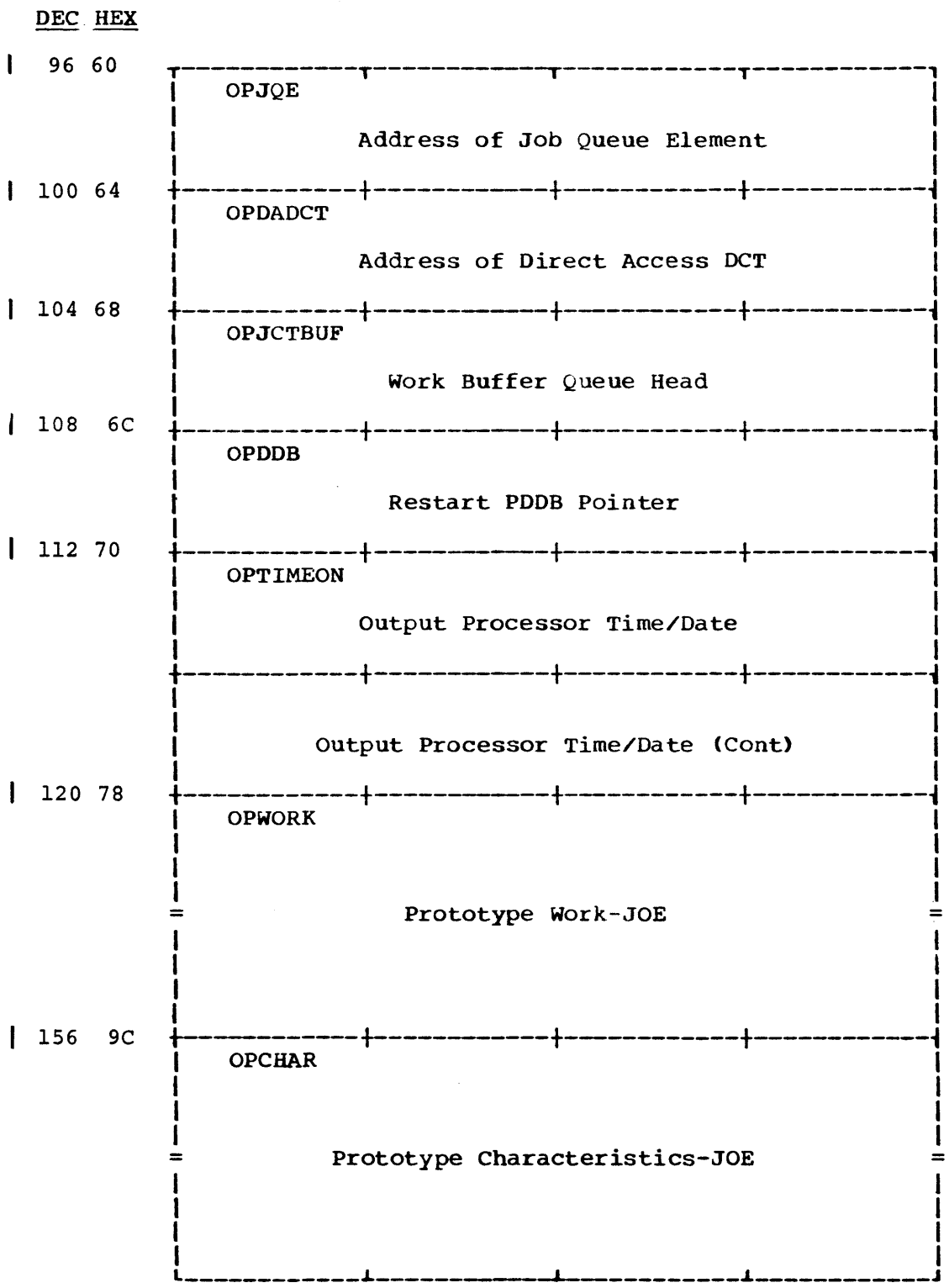
Note: The above format assumes HASPGEN parameter &DEBUG=YES. If &DEBUG=NO, the first and third words above are not present. Only the second word is present and its offset is zero.

HASP OUTPUT PCE WORK AREA

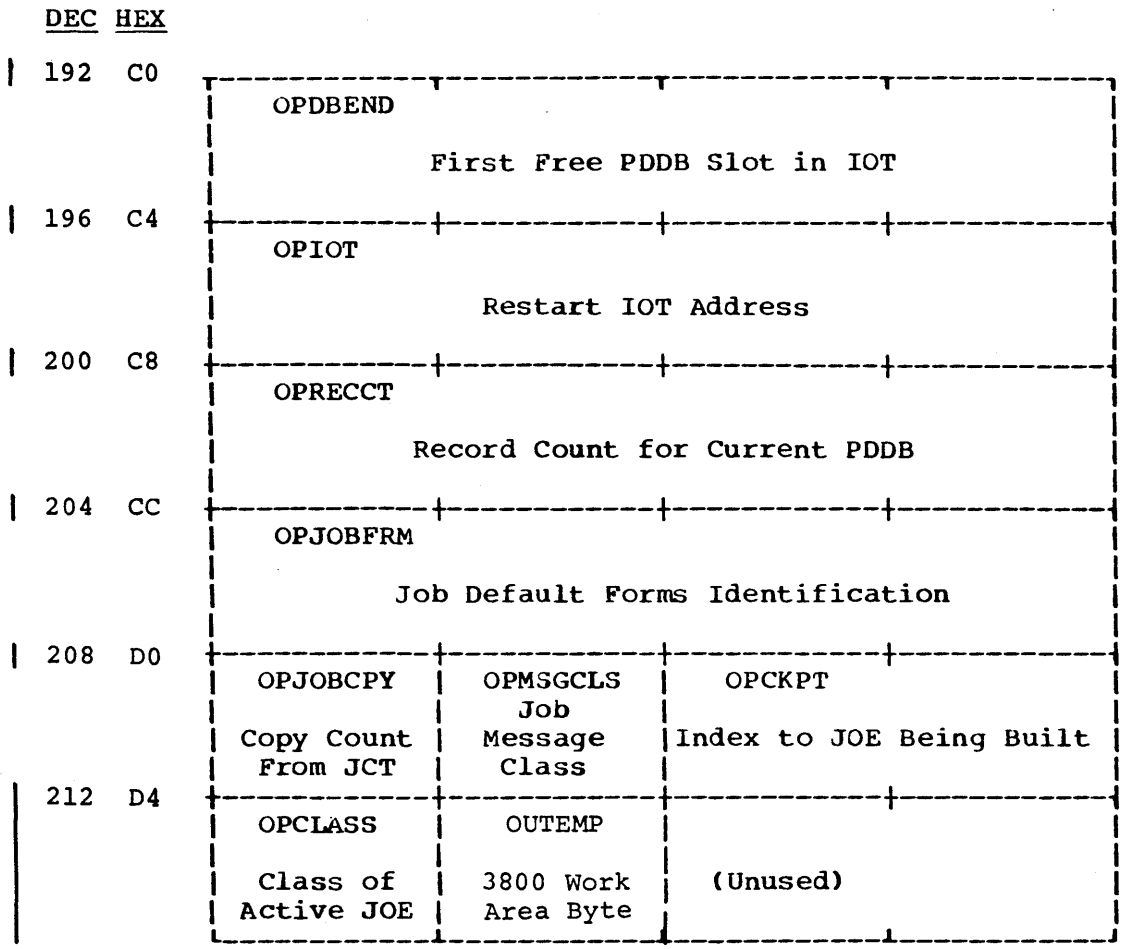
The next figure shows the format and describes each field in the Output Processor PCE Work Area (OUTWORK). When a job is placed in the output queue, the output requirements for the job are scanned and appropriate entries for it are added to the JOB Output Table. Since this process may require more than one HASP dispatch, interim results of the scan are stored by the Output Processor in its work area.

Offset	Bytes and Bits	Field Name	Description
96 (60)	4	OPJQE	Address of Job Queue Element
100 (64)	4	OPDADCT	Address of Direct-Access DCT
104 (68)	4	OPJCTBUF	Work Buffer Queue Head
108 (6C)	4	OPDDB	Restart PDDB pointer
112 (70)	8	OPTIMEON	Output processor time/date
120 (78)	36	OPWORK	Prototype work-JOE
156 (9C)	36	OPCHAR	Prototype char-JOE
192 (C0)	4	OPDBEND	First free PDDB slot in IOT
196 (C4)	4	OPIOT	Restart IOT address
200 (C8)	4	OPRECCT	Record count for current PDDB
204 (CC)	4	OPJOBFRM	Job default forms ID
208 (D0)	1	OPJOBCPY	Job level copy count from JCT
209 (D1)	1	OPMSGCLS	Job message class
210 (D2)	2	OPCKPT	Index to JOE being built
212 (D4)	1	OPCLASS	Class of active JOE
213 (D5)	1	OUTEMP	3800 Work area byte

HASP OUTPUT PCE WORK AREA



HASP OUTPUT PCE WORK AREA



HASP PROCESSOR CONTROL ELEMENT

The HASP Processor Control Element (PCE) has three roles in the system. It is the CPU dispatching control block, it is a register save area, and it provides unique work areas for those processors which can simultaneously operate multiple devices or functions. In the first two roles, the PCE is analogous to the Task Control Block (TCB) of OS.

The PCEs are chained together in the forward and backward directions just as are OS save areas. The top PCE is chained from and to the OS system-provided save area. The register save portion of each PCE is compatible with OS save areas.

The HASP Dispatcher processes the PCE chain in a forward direction looking for a dispatchable processor, as indicated by a processor event wait field of all zeros. Each processor returns to the dispatcher by issuing the $WAIT macro instruction which sets a bit in the event wait field. These bits are cleared as a result of $POST macro instructions issued when certain system events occur or when a specific processor is to be made dispatchable.

Many of the major processors of HASP (e.g., input, execution, print/punch) can control multiple simultaneous devices, terminals, or processes. In this case, there are several PCEs controlled by one copy of processor coding. Each PCE then has a variable length work area extension to provide unique storage for control of each device or process.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	PCESAVEA	Reserved
04 (04)	4	PCEPREV	Address of previous PCE
08 (08)	4	PCENEXT	Address of next PCE
12 (0C)	4	PCELINK	Register 14 (LINK) storage
16 (10)	4	PCER15	Register 15 storage
20 (14)	4	PCER0	Register 0 storage
24 (18)	4	PCER1	Register 1 storage
28 (1C)	4	PCEWA	Register 2 (WA) storage
32 (20)	4	PCEWB	Register 3 (WB) storage
36 (24)	4	PCEWC	Register 4 (WC) storage
40 (28)	4	PCEWD	Register 5 (WD) storage

Offset	Bytes and Bits	Field Name	Description
44 (2C)	4	PCEWE	Register 6 (WE) storage
48 (30)	4	PCEWF	Register 7 (WF) storage
52 (34)	4	PCEWG	Register 8 (WG) storage
52 (34)	4	PCEBASE3	Register 8 (BASE3) storage
56 (38)	4	PCER9	Register 9 storage
60 (3C)	4	PCEJCT	Register 10 (JCT) storage
64 (40)	4	PCEBASE1	Register 11 (BASE1) storage
68 (44)	4	PCEBASE2	Register 12 (BASE2) storage
72 (48)	2	PCEEWF	Processor event wait field (byte 0)
	1...	$EWFPOST	A PCE has been $POSTed
	.1..	$EWFBUF	Waiting for a buffer
	..1.	$EWFTRAK	Waiting for a track
	...1	$EWFJOB	Waiting for a job
	.... 1...	$EWFUNIT	Waiting for a unit
	.... .1..	$EWFCKPT	Waiting for a checkpoint
	.... ..1.	$EWFCMB	Waiting for a Console Message Buffer
	.... ..1.	$EWFSMF	Waiting for an SMF buffer
	1	$EWFJOT	Waiting for JOT service
		PCEEWF+1	Processor event wait field (byte 1)
	1... ..x.	$EWFOPER	Waiting for an operator response
	.1.. ..x.	$EWFIO	Waiting for I/O
	..1. ..x.	$EWFWORK	Waiting to be redirected
	....1 ..x.	$EWFHOLD	Waiting for a $S command
	.... 1.x.	$EWFDDB	Waiting for a DDT
	.... ..x1	$EWFOROL	Gave up overlay area
74 (4A)	2	PCEID	Processor number (byte 0)
		PCEID+0	Processor type (byte 0)
	1.xx x...	$PCEPRSID	Print special PCE ID
	.1xx x...	$PCEPUSID	Punch special PCE ID
	..xx x1..	$PCEINRID	Internal special PCE ID
	..xx x.1.	$PCERJEID	Remote special PCE ID
	..xx x..1	$PCELCLID	Local special PCE ID

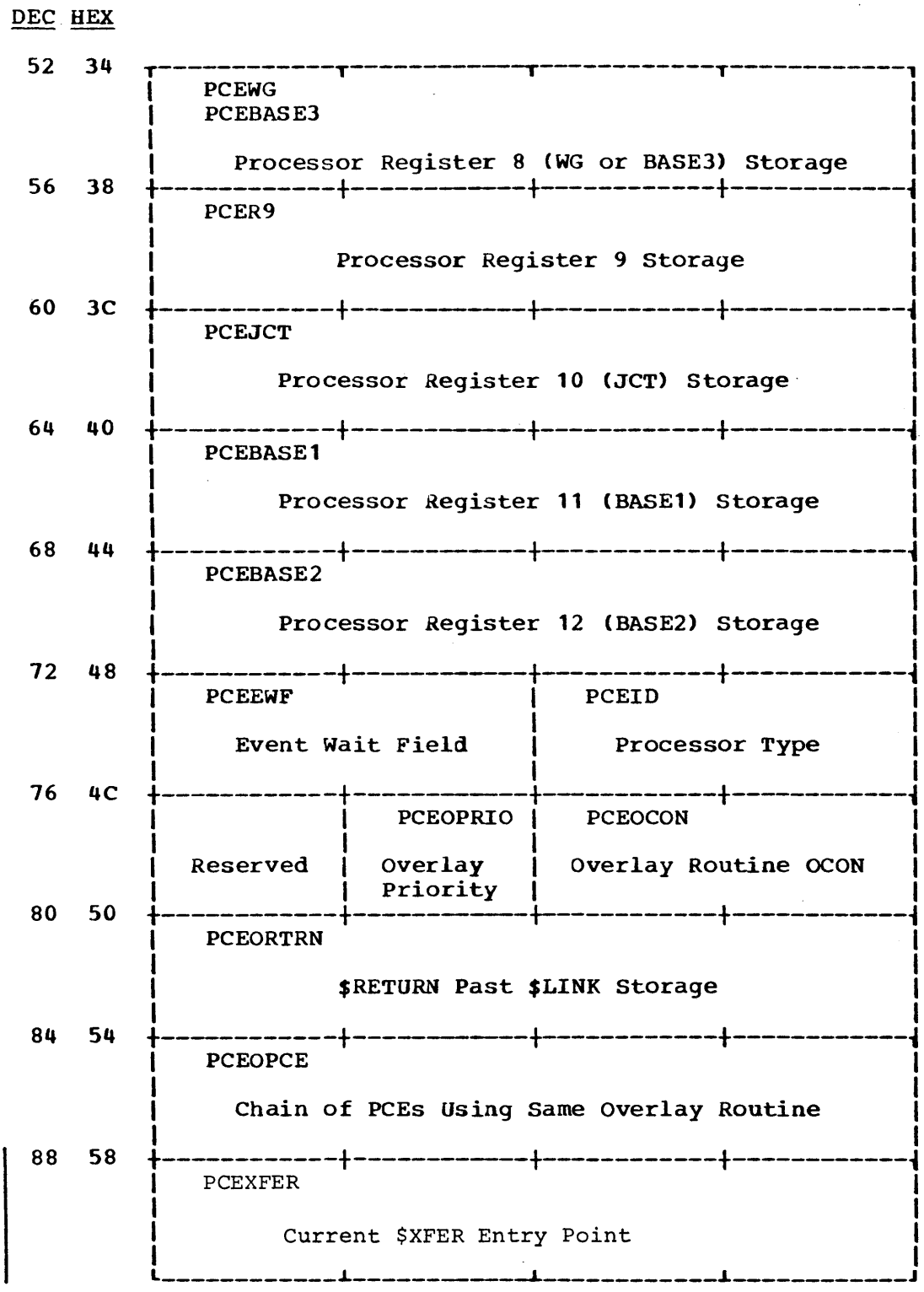
HASP PROCESSOR CONTROL ELEMENT

Offset	Bytes and Bits	Field Name	Description
		PCEID+1	Processor type (byte 1)
	00	PCEASYID	Asynch PCE ID
	01	PCERDRID	Input PCE ID
	03	PCEXEQID	Execution PCE ID
	04	PCETHWID	Execution Thaw PCE ID
	06	PCEOUTID	Output PCE ID
	07	PCEPRTID	Print PCE ID
	08	PCEPUNID	Punch PCE ID
	09	PCEPRGID	Purge PCE ID
	0A	PCECONID	Console PCE ID
	0B	PCEMLMID	Line Manager PCE ID
	0C	PCETIMID	Timer PCE ID
	0D	PCECKPID	Checkpoint PCE ID
	0E	PCEGPRID	Priority Aging PCE ID
76 (4C)	1		Reserved
77 (4D)	1	PCEOPRIO	Requested overlay routine priority
78 (4E)	2	PCEOCON	Requested overlay routine OCON
80 (50)	4	PCEORTRN	$RETURN past $LINK storage
84 (54)	4	PCEOPCE	Chain of PCEs using same overlay
88 (58)	4	DCEXFER	Address of XFER routine
92 (5C)	4		Reserved
96 (60)	n	PCEWORK	Variable length processor work area

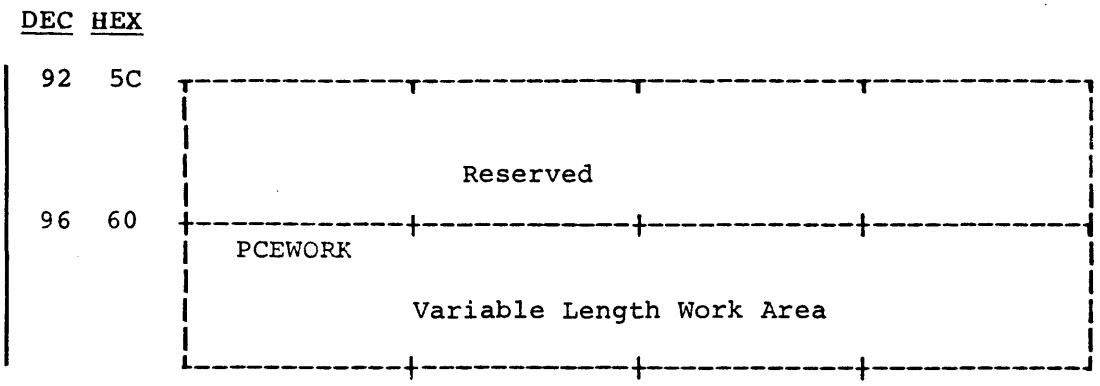
HASP PROCESSOR CONTROL ELEMENT

DEC	HEX	
00	00	PCESAVEA Reserved
04	04	PCEPREV Address of Previous Processor Control Element
08	08	PCENEXT Address of Next Processor Control Element
12	0C	PCELINK Processor Register 14 (LINK) Storage
16	10	PCER15 Processor Register 15 Storage
20	14	PCER0 Processor Register 0 Storage
24	18	PCER1 Processor Register 1 Storage
28	1C	PCEWA Processor Register 2 (WA) Storage
32	20	PCEWB Processor Register 3 (WB) Storage
36	24	PCEWC Processor Register 4 (WC) Storage
40	28	PCEWD Processor Register 5 (WD) Storage
44	2C	PCEWE Processor Register 6 (WE) Storage
48	30	PCEWF Processor Register 7 (WF) Storage

HASP PROCESSOR CONTROL ELEMENT



HASP PROCESSOR CONTROL ELEMENT



HASP PERIPHERAL DATA DEFINITION BLOCK

The Peripheral Data Definition Block (PDDB), contained in an IOT, is a variable-length control block which completely describes one output data set for a job. In addition to containing the data set's initial track address, it can also contain forms information, SYSOUT class, copy count, FCB identifier, and other information.

Offset	Bytes and Bits	Field Name	Description
00 (00)	1	PDBFLAG1	First flag byte
	1...	PDB1FLG2	FLAG2 (options) exists (see PDBFLAG2)
	.1..	PDB1NULL	This is a null PDDB
	..1.	PDB1LOG	This PDB is for the HASP Job Log
	...1	PDB1MDES	This PDB is followed by others indicating multiple destinations
	.... 1...	PDB3800	3800 options word present
	.... .1..	PDBOPCDJ	DCB=OPTCD=J Specified
	.... ..1.	PDBBURST	BURST=Y Specified
	1	PDB1RSV7	Reserved
01 (01)	1	PDBCLASS	Output class of this data set
02 (02)	1	PDBLENG	Length of this PDDB
03 (03)	4	PDBMTTR	Starting track of this data set
03 (03)	2	PDBSTNR	Step number of this data set
05 (05)	2	PDBDDNR	DD number of this data set
07 (07)	0	PDBBASLN	Basic PDDB length (EQU)
		Following fields are present only if PDB1FLG2 is set to one.	
07 (07)	1	PDBFLAG2	Optional fields flag byte
	1...	PDB2FORM	PDBFORMS
	.1..	PDB2FCB	PDBFCB
	..1.	PDB2UCS	PDBUCS
	...1	PDB2RECC	PDBRECCT
	.... 1...	PDB2MSC1	PDBMISC1
	.... .1..	PDB2MSC2	PDBMISC2
	11	PDB2WTRI	PDBWTRID
08 (08)	0	PDBFL2LN	Minimum nonbasic length (EQU)

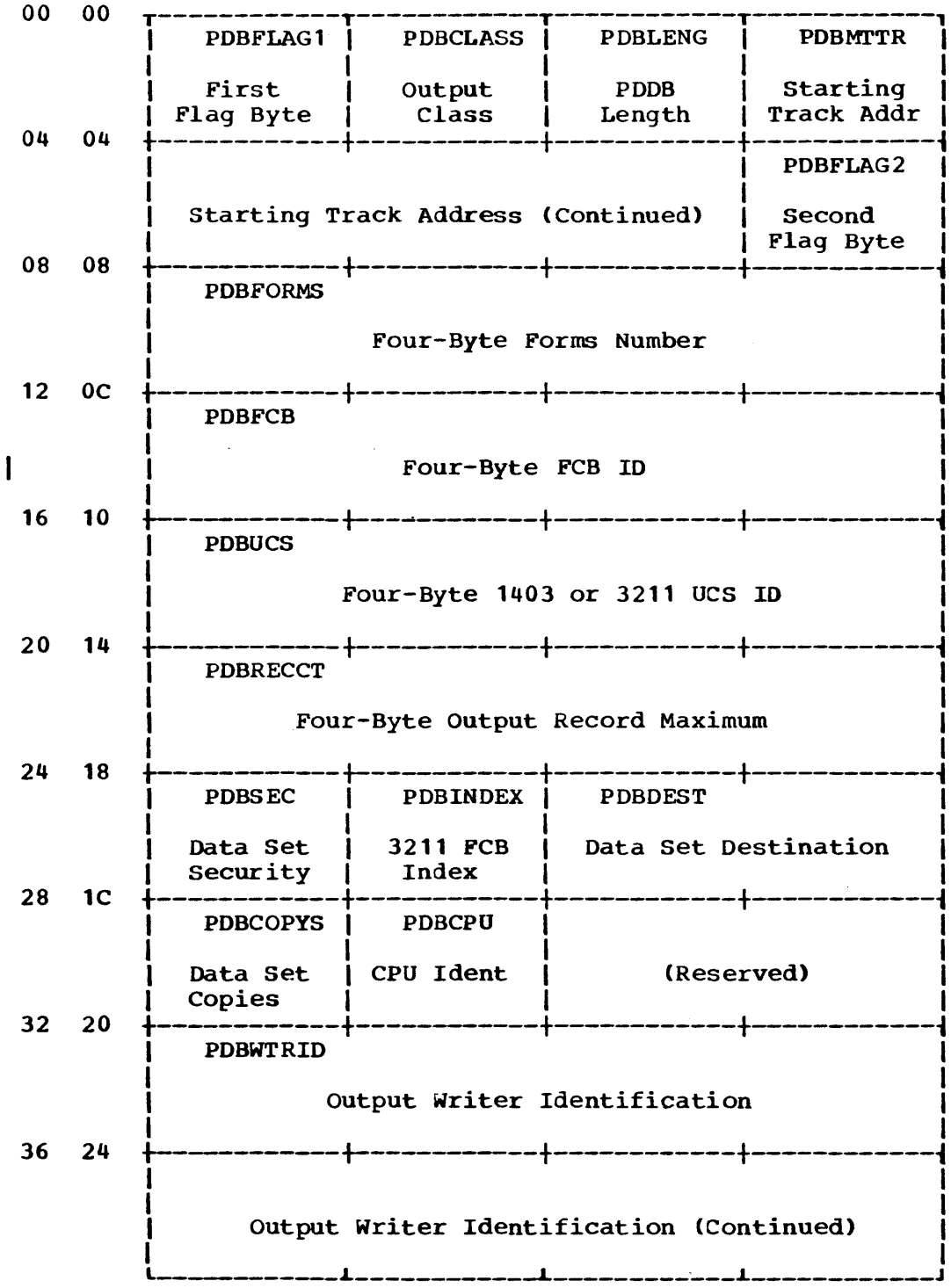
HASP PERIPHERAL DATA DEFINITION BLOCK

Offset	Bytes and Bits	Field Name	Description
			Each of the following 4-byte optional fields is present only if the associated bit in PDBFLAG2 is set to one.
08 (08)	4	PDBFORMS	Four-byte forms number
08 (08)	4	PDBFCB	Four-byte FCB ID
08 (08)	4	PDBUCS	Four-byte 1403 or 3211 UCS ID
08 (08)	4	PDBRECCT	Four-byte output record maximum
08 (08)	4	PDBMISC1	Miscellaneous ---
08 (08)	1	PDBSEC	- Data set security ID
09 (09)	1	PDBINDEX	- 3211 FCB index value
10 (0A)	2	PDBDEST	- Data set output destination
12 (0C)	0		PDBMISC1 (ORG)
08 (08)	4	PDBMISC2	Miscellaneous ---
08 (08)	1	PDBCOPYS	- Copies of this data set
09 (09)	1	PDBCPU	- CPU ID
10 (0A)	2		- Reserved
12 (0C)	0		PDBMISC2 (ORG)
08 (08)	8	PDBWTRID	Eight-byte output writer ID
			The 3800 options word is present only if PDB3800 is set in PDBFLAG1.
08 (08)	4	PDB3800W	3800 options word
08 (08)	1	PDBFLAG3	3800 flag byte
	1...	PDBFLIM	Forms overlay ID specified
	.11.	PDBFLCG	Copy groups specified
	...1	PDBFLCM	Copy modification ID specified
	.... 1...	PDBFLX1	Character arrangement table 1 specified
	.... .1..	PDBFLX2	Character arrangement table 2 specified
	.... ..1.	PDBFLX3	Character arrangement table 3 specified
	1	PDBFLX4	Character arrangement table 4 specified

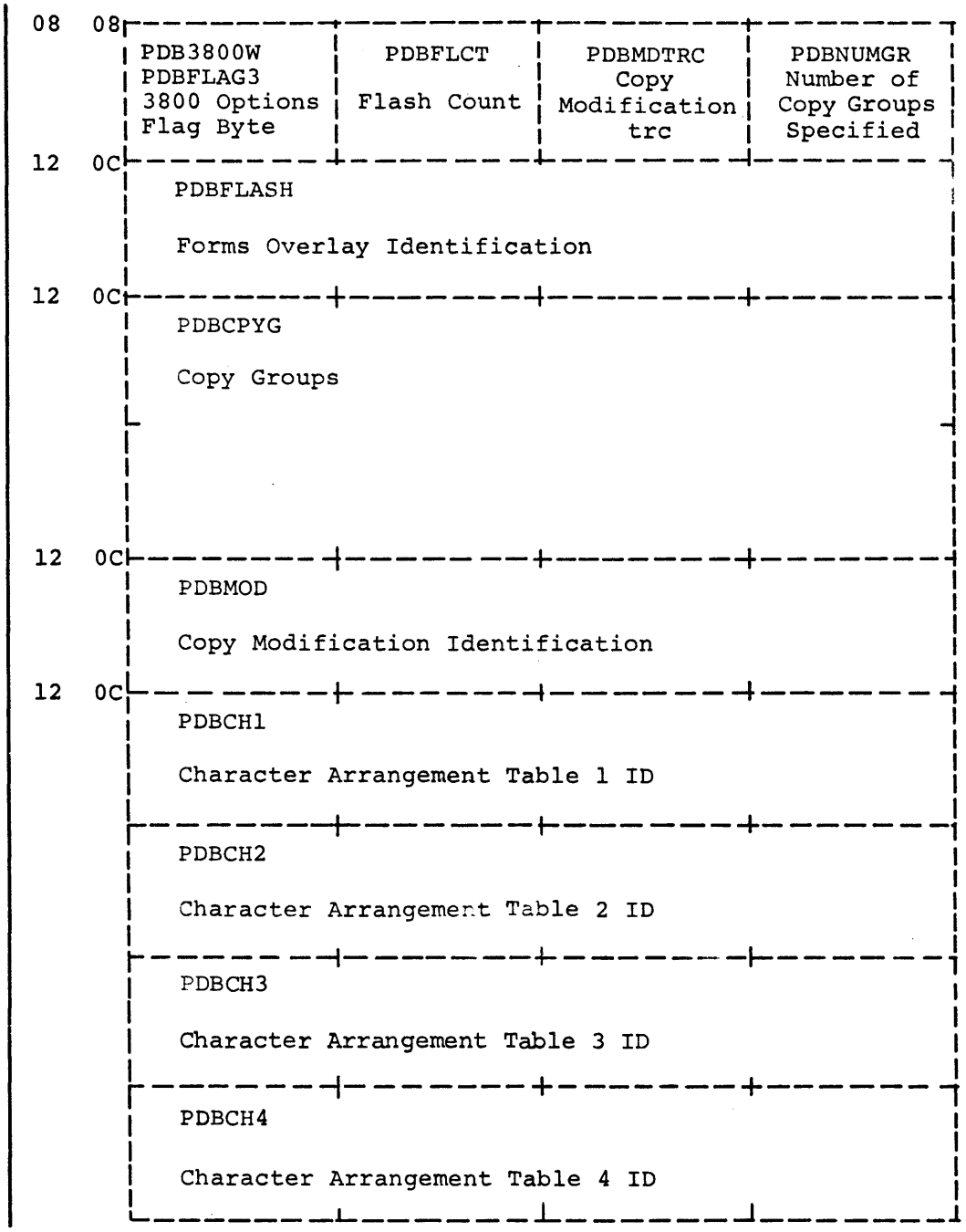
The following fields are present only if the corresponding flag in PDBFLAG3 is one.

09 (09)	1	PDBFLCT	Flash count
10 (0A)	1	PDBMDTRC	Copy modification trc
11 (0B)	1	PDBNUMGR	Number of copy groups
12 (0C)	4	PDBFLASH	Forms overlay ID
12 (0C)	8	PDBCPYG	Copy groups
12 (0C)	4	PDBMOD	Copy modification ID
12 (0C)	4	PDBCH1	Character arrangement table 1 ID
16 (10)	4	PDBCH2	Character arrangement table 2 ID
20 (14)	4	PDBCH3	Character arrangement table 3 ID
24 (18)	4	PDBCH4	Character arrangement table 4 ID

HASP PERIPHERAL DATA DEFINITION BLOCK



HASP PERIPHERAL DATA DEFINITION BLOCK

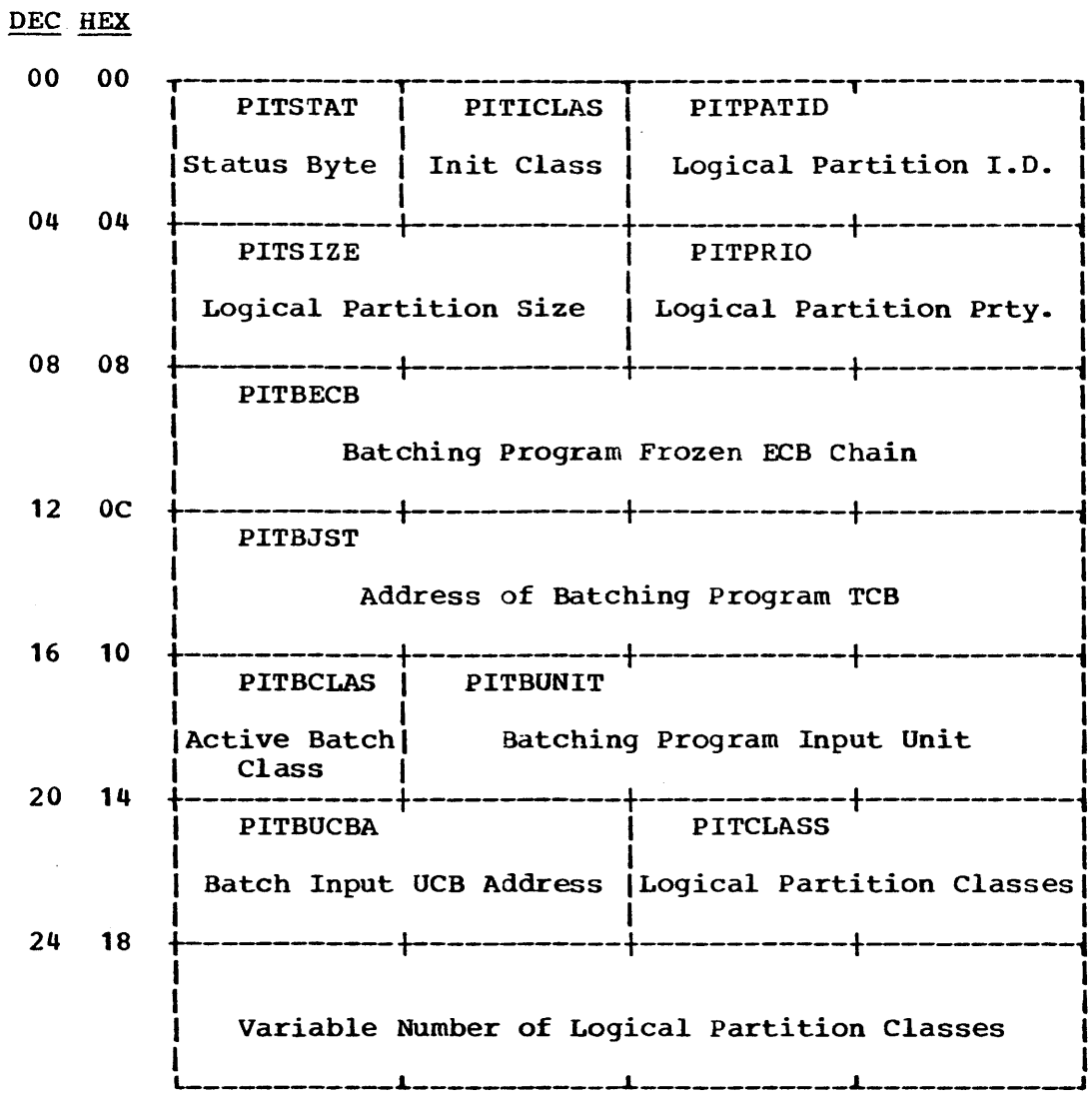


HASP PARTITION INFORMATION TABLE

The Partition Information Table (PIT) completely describes a HASP logical partition, the job classes eligible to run in it, and its current state. The nth PIT is assembled using values &PID(n), &PRI(n), &OSC(n), and &CLS(n). The PIT is used only in module HASPXEQ.

Offset	Bytes and Bits	Field Name	Description
00 (00)	1	PITSTAT	Logical partition status byte
	1...	PITHOLDA	PIT is drained ($P I)
	.1..	PITHOLD1	PIT is drained ($P IN)
	..1.	PITBUSY	Partition busy indicator
	...1	PITIDLE	PIT IDLE message switch
	1	PITLAST	Last PIT indicator
01 (01)	1	PITICLASS	Logical partition initiator class
02 (02)	2	PITPATID	Logical partition identification
04 (04)	2	PITSIZE	Logical partition size
06 (06)	2	PITPRIO	Logical partition priority
08 (08)	4	PITBECB	Batching program frozen ECB chain
12 (0C)	4	PITBJST	Address of batching program TCB
16 (10)	1	PITBCLAS	Active batching class
17 (11)	3	PITBUNIT	Batching program input unit
20 (14)	2	PITBUCBA	Batching program input UCB address
22 (16)	10	PITCLASS	Logical partition classes
32 (20)	8	PITLNGTH	Length of PIT

HASP PARTITION INFORMATION TABLE



HASP PRINT/PUNCH PCE WORK AREA

The next figure shows the format and describes each field in the Print/Punch Processor Work Area (PPPWORK). In order for one module to handle many output devices simultaneously, the following work area is defined to store job related data for each print/punch device.

Offset	Bytes and Bits	Field Name	Description
96 (60)	1	PPFLAG	Print/punch sync flags
96 (60)	4	PDCT	Address of print/punch/remote DCT
100 (64)	1	PDCTFLAG	Print/punch/remote DCT flags
100 (64)	4	PDADCT	Address of direct-access DCT
104 (68)	4	PJOB	Address of Job Queue Entry
108 (6C)	4	PRCHKPTE	Address of checkpoint element
112 (70)	4	PUERRPT	Address of punch error CCW
116 (74)	8	PTIMEON	Print/punch sign-on time
124 (7C)	4	PBUFSAVE	Address of next print/punch buffer
128 (80)	4	PCCWPT	Address of last CCW
132 (84)	4	PCCWEND	Address of last possible CCW
136 (88)	40	PMESSAGE	Message work area
176 (B0)	2	PDDBSKIP	Count of pages to skip
178 (B2)	2		Unused
180 (B4)	1	PPRCFLAG	Checkpoint flags
181 (B5)	1	PPRCPYCT	Copy count
182 (B6)	2	PCEEJRCB	Last eject RCB displacement
184 (B8)	2	PDDBDISP	Current PDDB displacement
186 (BA)	2	PDDBPGCT	Current PDDB page count

HASP PRINT/PUNCH PCE WORK AREA

Offset	Bytes and Bits	Field Name	Description
188 (BC)	4	PPLNCDCT	Current line or card count
192 (C0)	4	PRPAGECT	Current page count
196 (C4)	4	PCEJMTTR	Last eject buffer MTTR
200 (C8)	4	PCEIOTTR	Current IOT MTTR
204 (CC)	4	PSMF6PGE	Current pages this SMF type-6
208 (D0)	4	PSMF6NLR	Current lines this SMF type-6
212 (D4)	1	PPJNDS	Number of data sets
213 (D5)	1	PPSTCNR	Starting copy number
214 (D6)	2	PPRRSV	Reserved
216 (D8)	1	PBUFOPT	Print/punch buffering option
216 (D8)	4	PDEVTYPE	Print/punch device type
220 (DC)	4	PLSAVE	Link register save word
224 (E0)	4	PLSAVE2	Save Word
228 (E4)	4	PLSAVE3	Save Word
232 (E8)	4	PRLINECT	Maximum lines per page
236 (EC)	4	PWKJOE	C'class', AL3 (work JOE)
240 (F0)	4	PCHJOE	A(characteristics JOE)
244 (F4)	4	PCEFORM	Forms for current PDDB
248 (F8)	4	PCEFCB	FCB for current PDDB
252 (FC)	4	PCEUCSB	UCSB for current PDDB
256 (100)	8	PPCPYG	Copy groups
264 (108)	4	PPFLASH	Flash ID
268 (10C)	1	PP3800FL	3000 Options flag
	1...	PPBURST	Bursting
	.1..	PDATATRC	TRC in the input stream
	..1.	PSEPPG	Setting up a separator
	...1	PPREXMIT	This is a rexmission

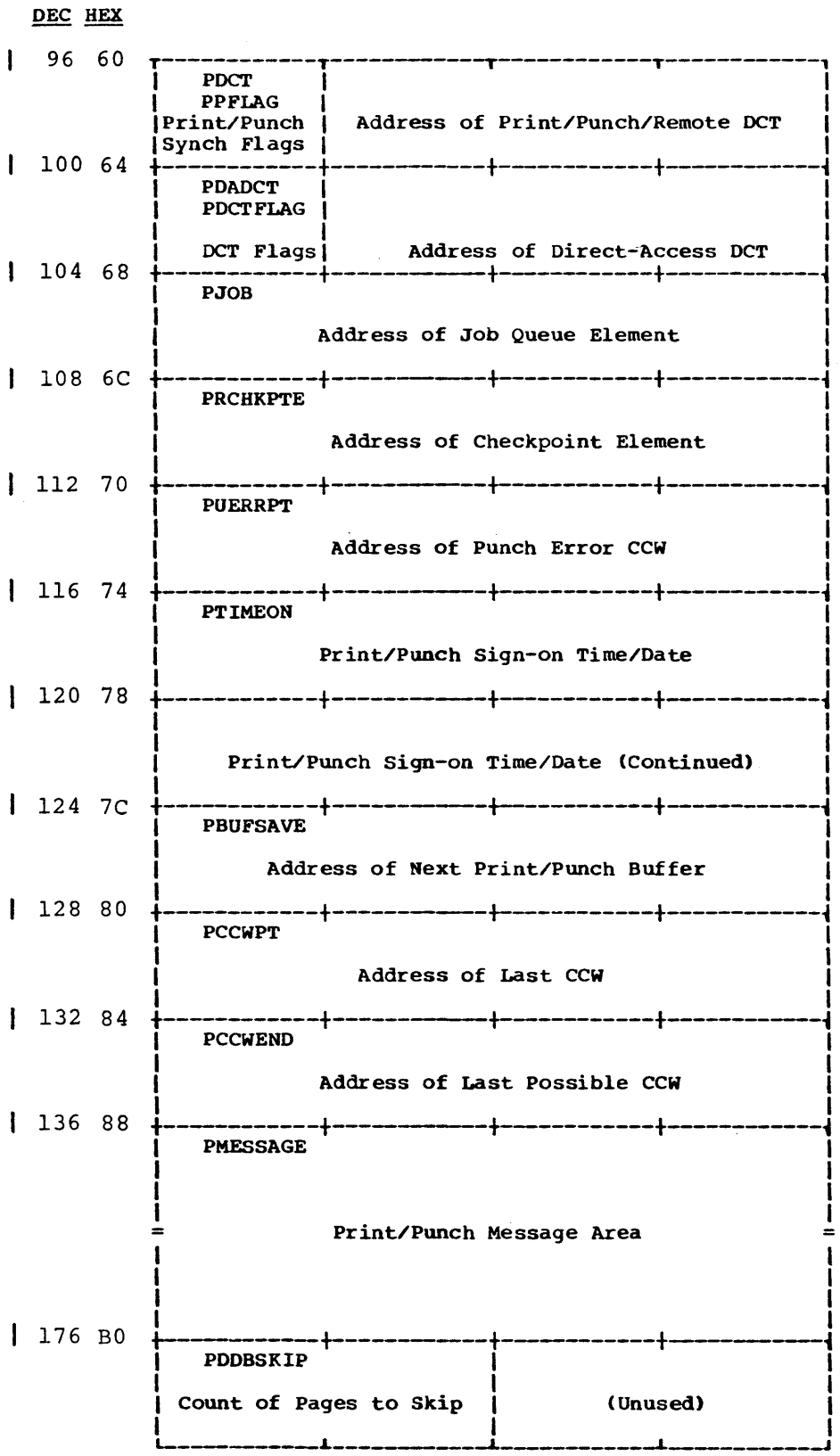
HASP PRINT/PUNCH PCE WORK AREA

Offset	Bytes and Bits	Field Name	Description
	.... 1...	PSETPRT	Force a SETPRT
	.... .1..	PJUSTSEP	Separator EOT Just issued
	.... ..1.	PPCLRP	Clear print not required
269 (10D)	1	PPTRC	TRC from last line
270 (10E)	1	PPFLCT	Flash count
271 (10F)	1	PP38RS	Work area
272 (110)	4	PPXLAT1	Character arrangement table 1
276 (114)	4	PPXLAT2	Character arrangement table 2
280 (118)	4	PPXLAT3	Character arrangement table 3
284 (11C)	4	PPXLAT4	Character arrangement table 4
288 (120)	4	PPMODPT	Copy modification module name
292 (124)	4	PLSAVE4	Save area 3800 setup
296 (128)	16	PSMF6JBN	Job name+ from the JMR
312 (138)	8	PSMF6UIF	User ID
320 (140)	8	PSMF6WST	PRPU signon time
328 (148)	4	PPJOBFRM	Job default forms ID
332 (14C)	4	PPRECCT	Maximum record count
336 (150)	4	PPJCARDS	Maximum cards read
340 (154)	4	PPJLINES	Number of SYSOUT print records
344 (158)	4	PPJPUNCH	Number of SYSOUT punch records
348 (15C)	4	PPJXEQOF	Time off execution
352 (160)	4	PPJXEQON	Time on execution
356 (164)	4	PPJDSKEY	Data set buffer validity key
360 (168)	4	PPJJOBEB	Job number

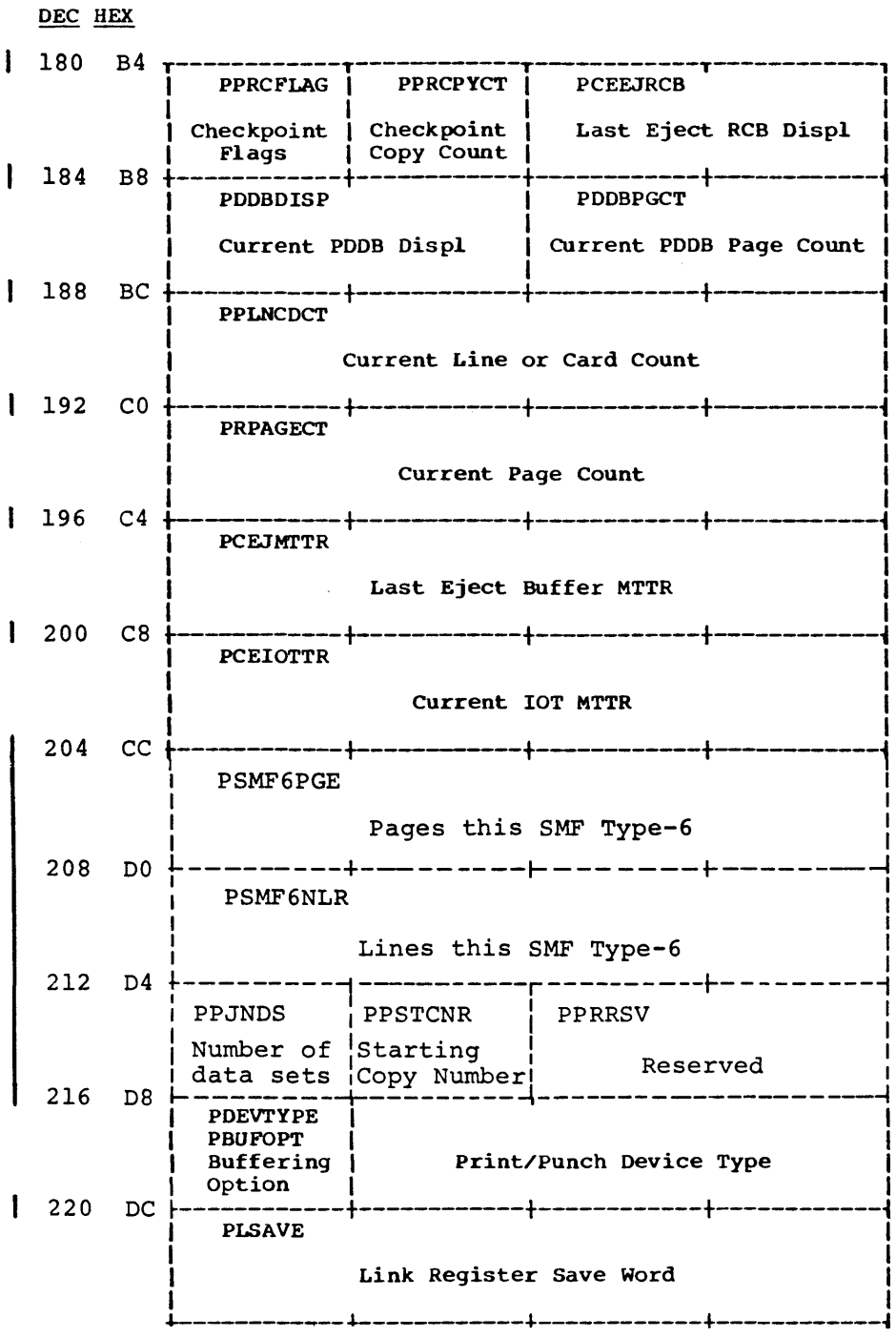
HASP PRINT/PUNCH PCE WORK AREA

Offset	Bytes and Bits	Field Name	Description
364 (16C)	4	PPJROOMN	Room number
368 (170)	8	PPJJNAME	Job Name
376 (178)	20	PPJPNAME	Programmer name
396 (18C)	1	PRINDEX	3211 index value
397 (180)	1	PPMSGCLS	Job message class
398 (18E)	1	PPDSCPY	Data set copy count
399 (18F)	1	PSMF6BID	SMF Work Byte
400 (190)	2	PBSPGCT	Backspace page count
402 (192)	28	PBSPTBL	Backspace table
430 (IAE)	2		Unused
	0	PRRPCEWS	Length of remote print PCE work area
	0	PURPCEWS	Length of remote punch PCE work area
432 (IBO)	n	PCCWCHN	Variable length print/punch CCW chain
	0	PRTPCEWS	Length of local print PCE work area
	0	PUNPCEWS	Length of local punch PCE work area

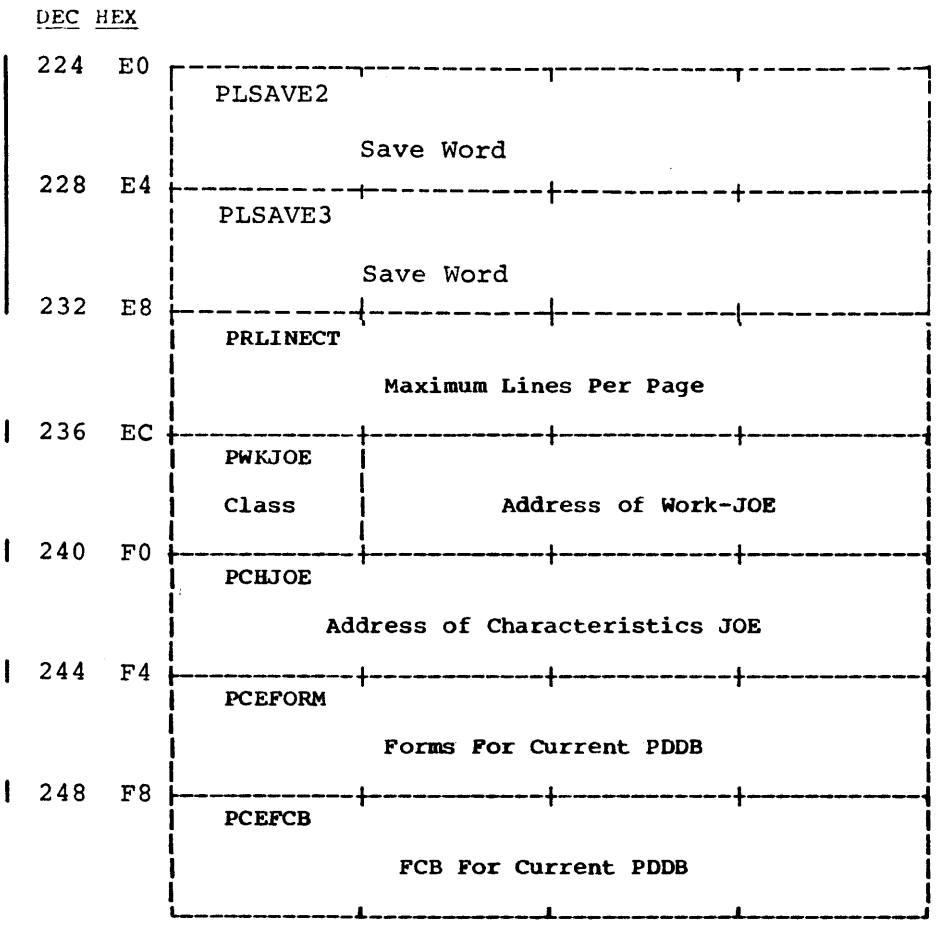
HASP PRINT/PUNCH PCE WORK AREA



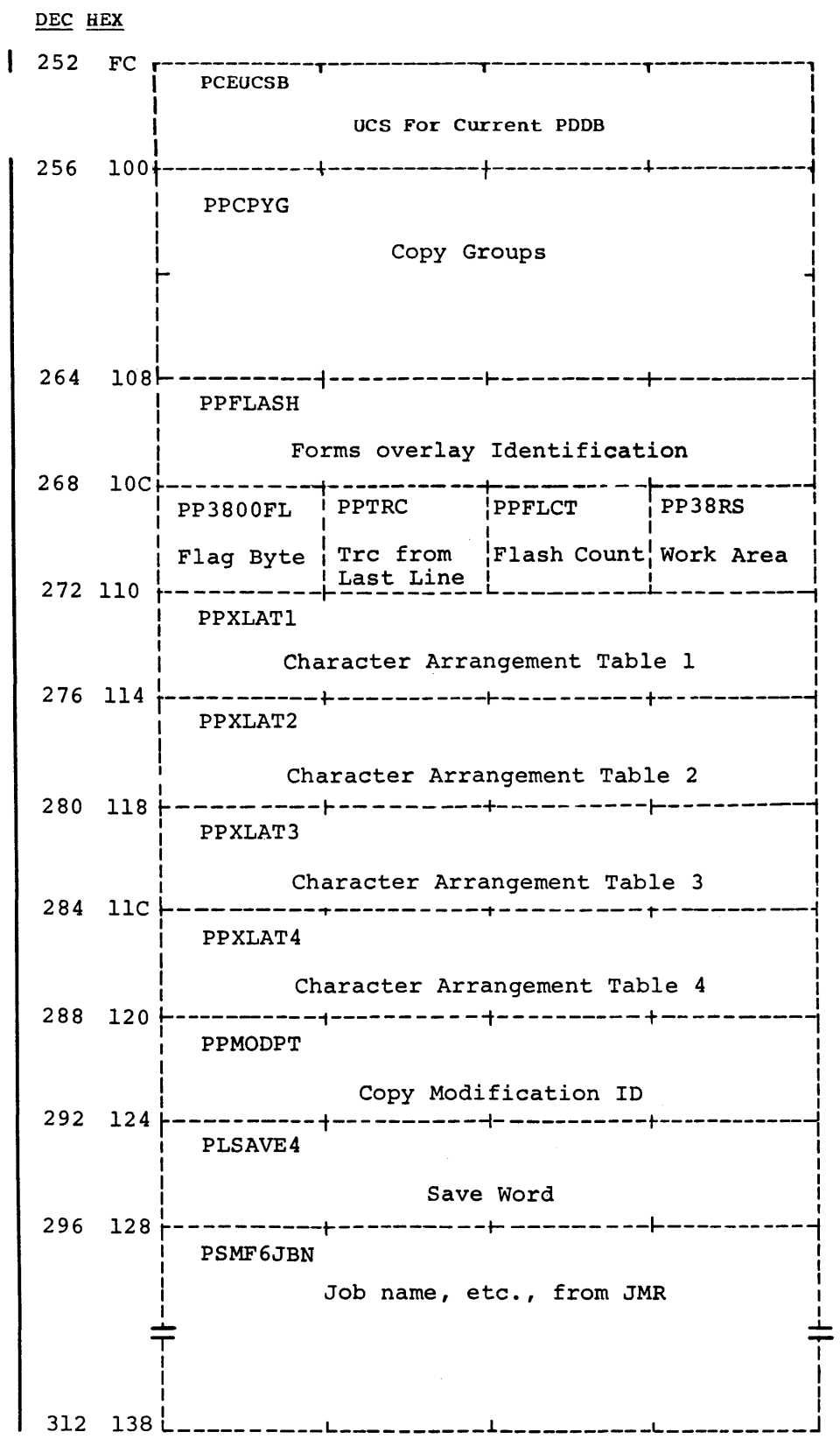
HASP PRINT/PUNCH PCE WORK AREA



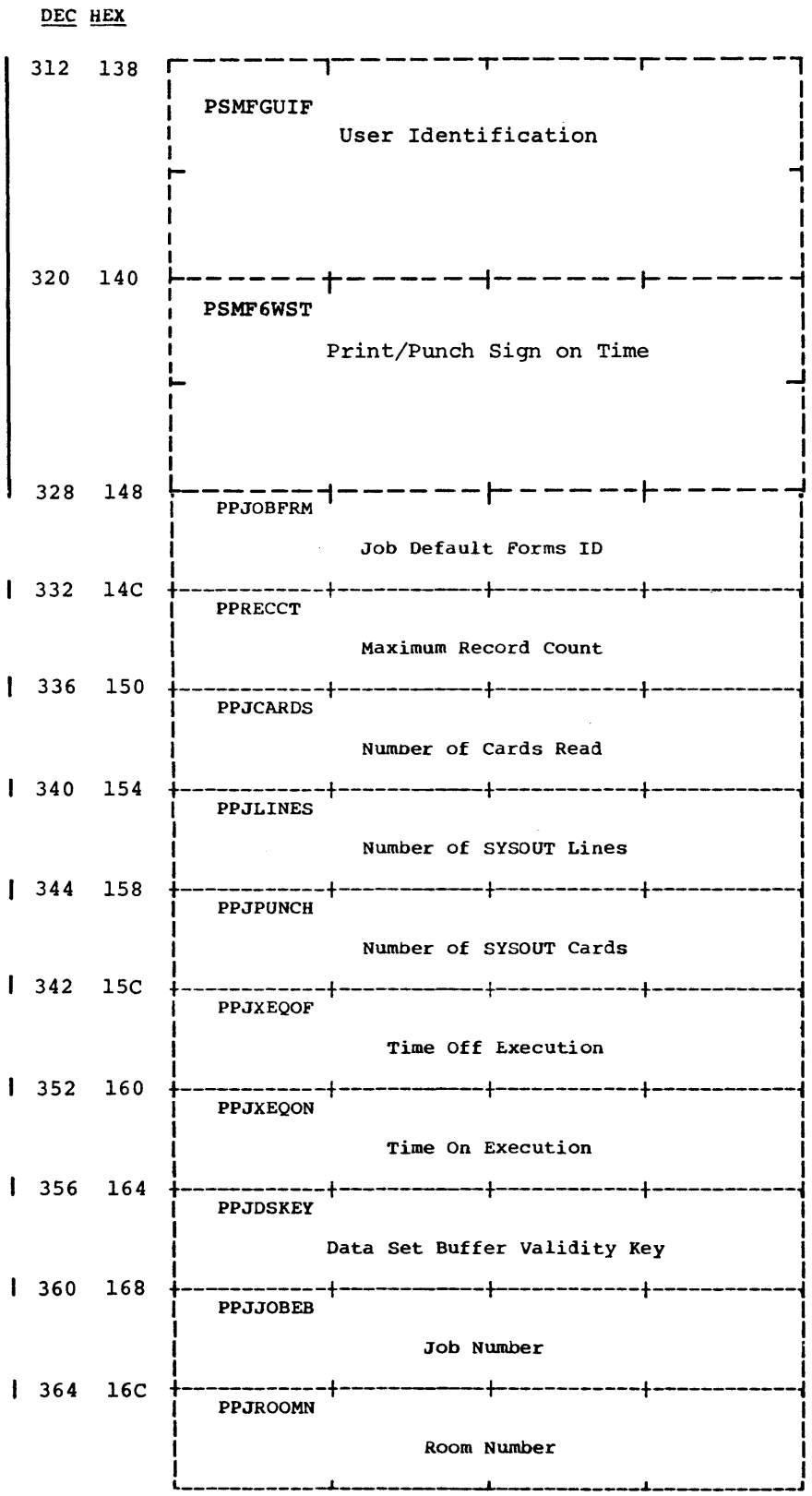
HASP PRINT/PUNCH PCE WORK AREA



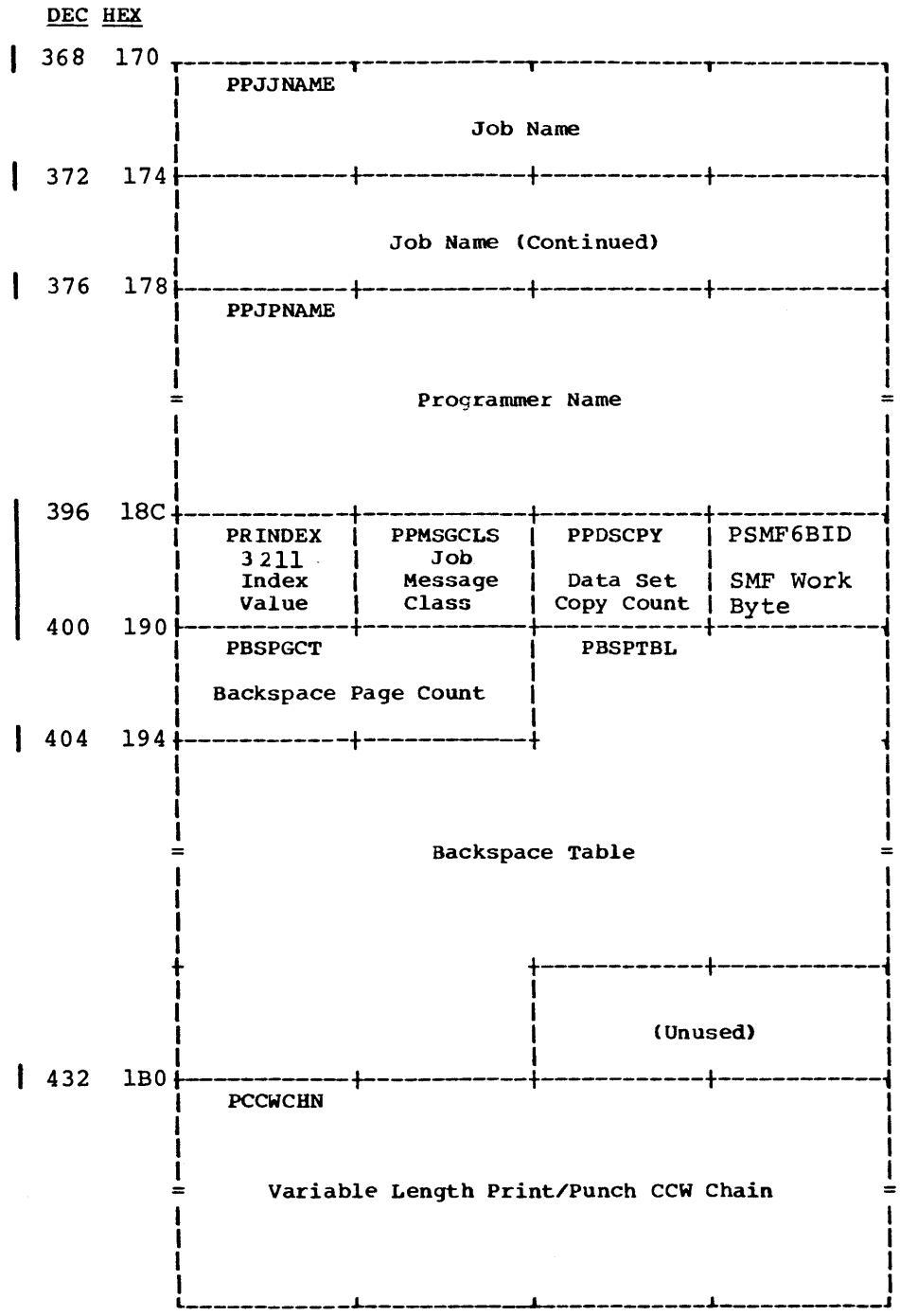
HASP PRINT/PUNCH PCE WORK AREA



HASP PRINT/PUNCH PCE WORK AREA



HASP PRINT/PUNCH PCE WORK AREA

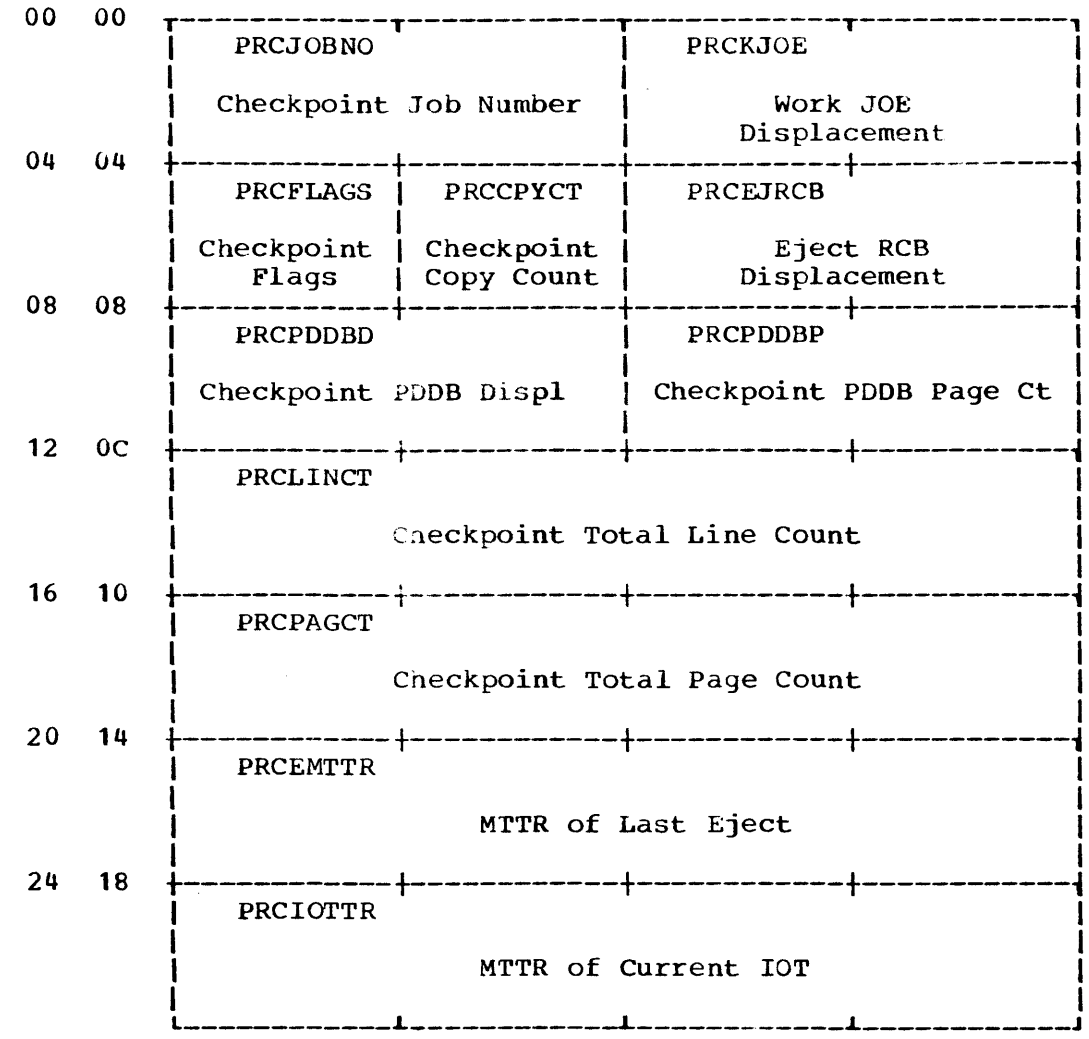


HASP PRINT CHECKPOINT ELEMENT

The next figure shows the format and describes each field in a Print/Punch Checkpoint Element (PRC). After a Print/Punch processor has been given work from the Job Output Table (JOT), a PRC element is used to checkpoint device status during output for purposes of warmstart.

Offset	Bytes and Bits	Field Name	Description
00 (00)	2	PRCJOBNO	CKPT - H - Job number
02 (02)	2	PRCKJOE	CKPT - H - CKPT JOE displacement
04 (04)	1	PRCFLAGS	CKPT - C - Flags
	1...	PRCHKUSE	Checkpoint entry is in use
	.1..	PRCHKJOB	Job active indicator
05 (05)	1	PRCCPYCT	CKPT - C - Current copy count
06 (06)	2	PRCEJRCB	CKPT - H - Eject RCB displacement
08 (08)	2	PRCPDDBD	CKPT - H - PDDB displacement
10 (0A)	2	PRCPDDBP	CKPT - H - PDDB page count
12 (0C)	4	PRCLINCT	CKPT - F - Total line count
16 (10)	4	PRCPAGCT	CKPT - F - Total page count
20 (14)	4	PRCEMTTR	CKPT - F - MTTR of last eject
24 (18)	4	PRCIOTTR	CKPT - F - MTTR of current IOT
28 (1C)	0	PRCSIZE	Length of a PRC element (EQU)

HASP PRINT CHECKPOINT ELEMENT

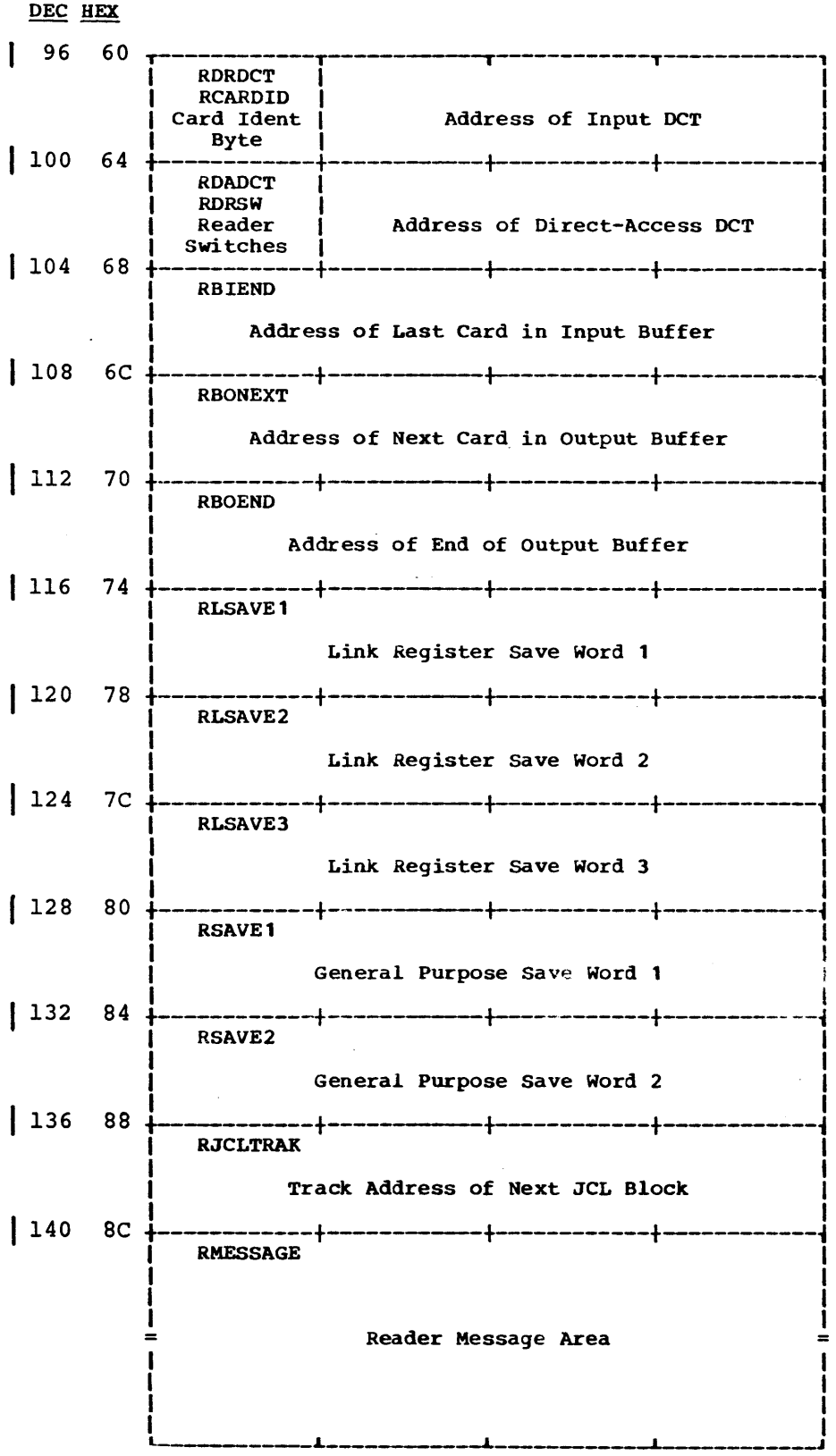


HASP READER PCE WORK AREA

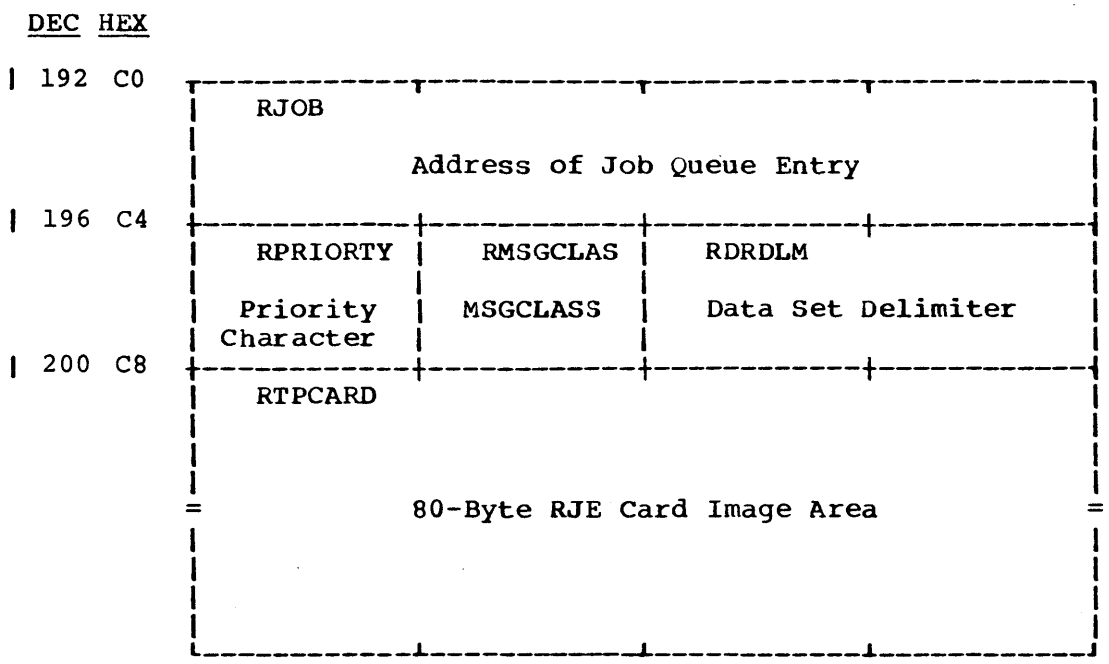
The Input Service PCE Work Area (RDRWORK) is an extension of the Input Service Processor Control Element which is used for reenterable storage by the Input Service Processor.

Offset	Bytes and Bits	Field Name	Description
96 (60)	0	RCARDID	Card identification byte
96 (60)	4	RDRDCT	Address of input DCT
100 (64)	1	RDRSW	Reader switches
100 (64)	4	RDADCT	Address of direct-access DCT
104 (68)	4	RBIEND	Address of last card in input buffer
108 (6C)	4	RBONEXT	Address of next card in output buffer
112 (70)	4	RBOEND	Address of end of output buffer
116 (74)	4	RLSAVE1	Link register save word 1
120 (78)	4	RLSAVE2	Link register save word 2
124 (7C)	4	RLSAVE3	Link register save word 3
128 (80)	4	RSAVE1	General purpose save word 1
132 (84)	4	RSAVE2	General purpose save word 2
136 (88)	4	RJCLTRAK	Track address of next JCL block
140 (8C)	52	RMESSAGE	Reader message area
192 (C0)	4	RJOB	Address of job queue entry
196 (C4)	1	RPRIORTY	Character from /*PRIORITY card
197 (C5)	1	RMSGCLAS	MSGCLASS from JOB card
198 (C6)	2	RDRDLM	Input data set delimiter
	0	RDRPCEWS	Length of normal input PCE WORK AREA
200 (C8)	80	RTPCARD	RJE input card image
	0	RJEPCEWS	Length of RJE input PCE work area

HASP READER PCE WORK AREA



HASP READER PCE WORK AREA



HASP SMF BUFFER

The System Management Facilities buffer (SMF) is used as a storage area within HASP to hold SMF records which are ready to be written to the SMF data set and for copies of common exit parameter areas. The first two words are used for control purposes. Following the two control words are either a common exit parameter area or an SMF record. The header section is identical for all SMF records. A subsystem header follows the header for subsystem records. There are six SMF record types produced by HASP: 6, 26, 43, 45, 47, and 48.

Offset	Bytes and Bits	Field Name	Description
			Buffer Control Area
00 (00)	4	SMFCHAIN	SMF buffer chain to next buffer
04 (04)	1	SMFTYPE	Type of buffer
	1111 1111	SMFJMRTP	JMR buffer
	0000 0000	SMFRECTP	SMF record buffer
05 (05)	3	SMFPARM	Reserved
08 (08)	4	JMRCHAIN	Pointer to purge record buffer
08 (08)	4	SMFRDW	SMF record descriptor word
			Beginning of JMR or HASP SMF record
12 (0C)	56	SMFJMR	JMR data area
			SMF Record Header Area
12 (0C)	1	SMFHDFLG	Header flag byte
13 (0D)	1	SMFHDRTY	Record type
	0000 0110	SMFOUTTP	HASP output processor SMF record type
	0001 1010	SMFPRGTP	HASP purge record type
	0010 1011	SMFSSSTP	HASP start subsystem record type
	0010 1101	SMFPSSTP	HASP stop subsystem record type
	0010 1111	SMFSSETP	HASP start subsystem event record type
	0011 0000	SMFPSETP	HASP stop subsystem event record type
14 (0E)	4	SMFHDTME	TOD, using format from time macro
18 (12)	4	SMFHDDTE	Date in packed decimal form: 00YYDDDF
22 (16)	2	SMFHDSID	System identification

HASP SMF BUFFER

Offset	Bytes and Bits	Field Name	Description
24 (18)	2	SMFHDMOD	System model identifier
26 (1A)	0	SMFRSTRT	Define start of record after header
			SMF type 6 Output Processor record
26 (1A)	8	SMF6JBN	Job name
34 (22)	4	SMF6RST	Reader start time
38 (26)	4	SMF6RSD	Reader start date
42 (2A)	8	SMF6UIF	User identification field
50 (32)	1	SMF6OWC	Output writer class
51 (33)	4	SMF6WST	Writer start time
55 (37)	4	SMF6WSD	Writer start date
59 (3B)	4	SMF6NLR	Number of logical records for writer
63 (3F)	1	SMF6IOE	I/O error indicator
	.... .1..	SMFDSER	Input error on jobs SYSOUT data set
	1	SMFCBER	Input error on HASP control block
64 (40)	1	SMF6NDS	Number of data sets processed by writer
65 (41)	4	SMF6FMN	Form number
69 (45)	7	SMF6RV1	Reserved (first 3 bytes used by HASP)
76 (4C)	4	SMF6JNM	HASP assigned job number
80 (50)	8	SMF6OUT	HASP logical output device name
88 (58)	4	SMF6FCB	FCB identification
92 (5C)	4	SMF6UCS	UCS identification
96 (60)	4	SMF6PGE	Page count
100 (64)	2	SMF6RTE	Output route code
102 (66)	0	SMF6END	End of type 6 record (not 3800)

HASP SMF BUFFER

Offset	Bytes and Bits	Field Name	Description
102 (66)	2	SMF6LN2	Length of first extension
104 (68)	8	SMF6CPS	Copies distribution
112 (70)	16	SMF6CHR	CHARS specification
128 (80)	4	SMF6MID	Copy modification module name
132 (84)	4	SMF6FLI	Forms overlay ID
136 (88)	1	SMF6FLC	Number of copies flashed
137 (89)	1	SMF6BID	Flag byte
	1...		Burster-Trimmer-Stacker used
	.1..		OPTCD=J specified
138 (8A)	0	SMF638E	End of record with 3800 extension

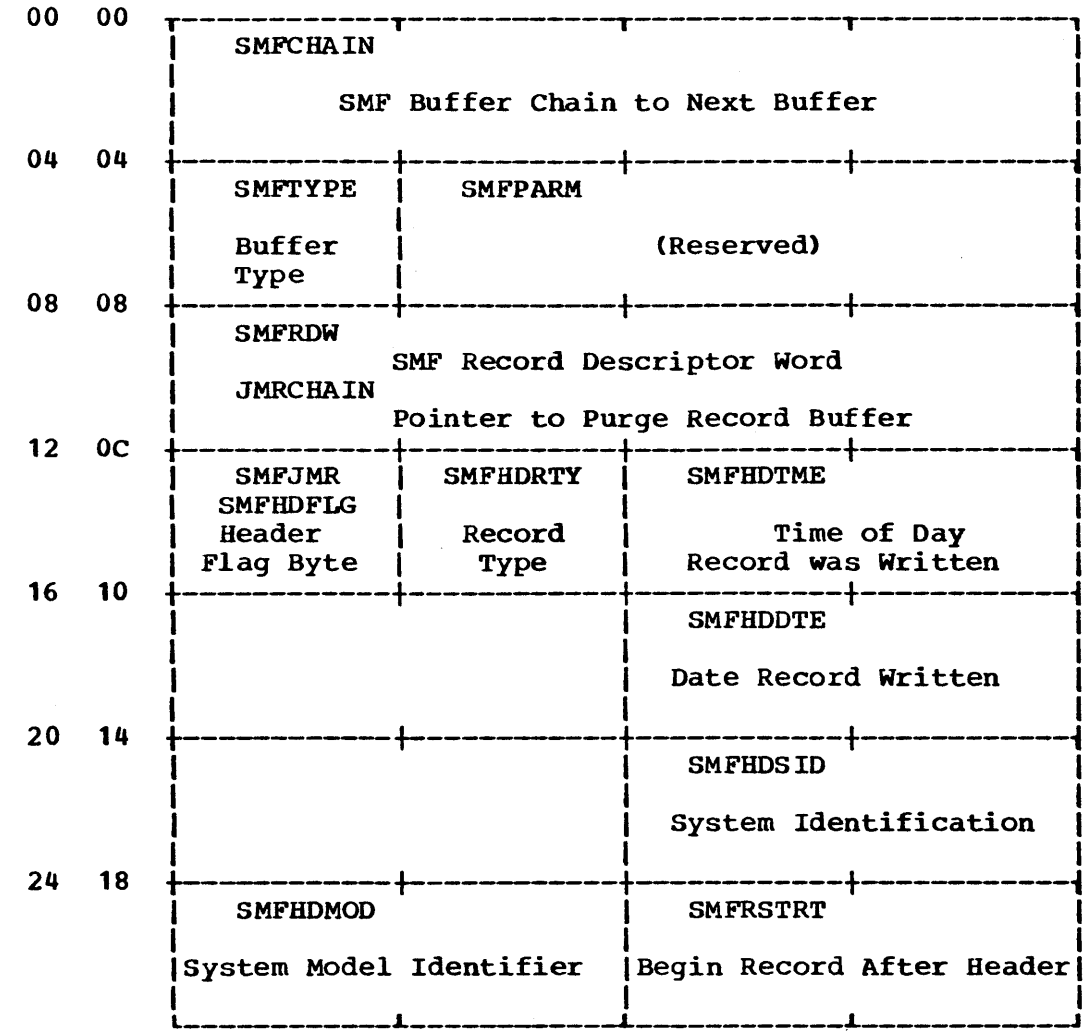
Offset	Bytes and Bits	Field Name	Description
			SMF type 26 purge record
26 (1A)	8	SMF26JBN	Job name (ORG SMFRSTRT)
34 (22)	4	SMF26RST	Reader start time
38 (26)	4	SMF26RSD	Reader start date
42 (2A)	8	SMF26UIF	User identification field
50 (32)	4	SMF26RSV	Reserved
54 (36)	2	SMF26SBS	Subsystem identification
55 (37)	0000 0010	SMFHSPID	HASP subsystem identification
56 (38)	2	SMF26IND	Section existence indicator =X'E000'
56 (38)	1...		Descriptor section indicator
56 (38)	.1..		Events section indicator
56 (38)	..1.		Actuals section indicator
58 (3A)	2	SMF26LN1	Descriptor section length, including this field
60 (3C)	3	SMF36RV1	Reserved
63 (3F)	1	SMF26INF	Job information
	1...	SMFPRICD	/*PRIORITY card present
	.1..	SMFSETUP	/*SETUP card(s) present
	..1.	SMFTHOLD	TYPRUN=HOLD was specified
	...1	SMFNOLOG	No job log option
	.... 1...	SMFEXBCH	Execution batching job
	.... .1..	SMFJBOP5	Reserved
	.... ..1.	SMFJBOP6	Reserved
	1	SMFOPCAN	Job canceled by $C or $P
64 (40)	4	SMF26JNM	HASP assigned job number
68 (44)	8	SMF26RV0	Job name from JOB card
76 (4C)	20	SMF26NAM	Programmer's name from JOB card
96 (60)	1	SMF26MSG	MSGCLASS from JOB card
97 (61)	1	SMF26CLS	Job class from JOB card
98 (62)	2	SMF26XPR	HASP execution selection priority
100 (64)	2	SMF26OPR	HASP output selection priority

Offset	Bytes and Bits	Field Name	Description
102 (66)	2	SMF26LOC	Input route code
104 (68)	8	SMF26DEV	HASP logical device name
112 (70)	4	SMF26ACT	Programmer's accounting number
116 (74)	4	SMF26ROM	Programmer's room number
120 (78)	4	SMF26XTM	Estimated execution time
124 (7C)	4	SMF26ELN	Estimated output lines
128 (80)	4	SMF26EPU	Estimated punched output
132 (84)	4	SMF26FRM	Default output form number
136 (88)	2	SMF26CYP	Print copy count
138 (8A)	2	SMF26LIN	Lines per page
140 (8C)	2	SMF26PRR	Print route code
142 (8E)	2	SMF26PUR	Punch route code
144 (90)	2	SMF26LN2	Events section length, including this field
146 (92)	2	SMF26RV2	Reserved
148 (94)	4	SMF26RPT	Reader stop time
152 (98)	4	SMF26RPD	Reader stop date
156 (9C)	16	SMF26RV3	Reserved
172 (AC)	4	SMF26XST	Execution start time
176 (B0)	4	SMF26XSD	Execution start date
180 (B4)	4	SMF26XPT	Execution stop time
184 (B8)	4	SMF26XPD	Execution stop date
188 (BC)	4	SMF26OST	Output Processor start time
192 (C0)	4	SMF26OSD	Output Processor start date
196 (C4)	4	SMF26OPT	Output Processor stop time

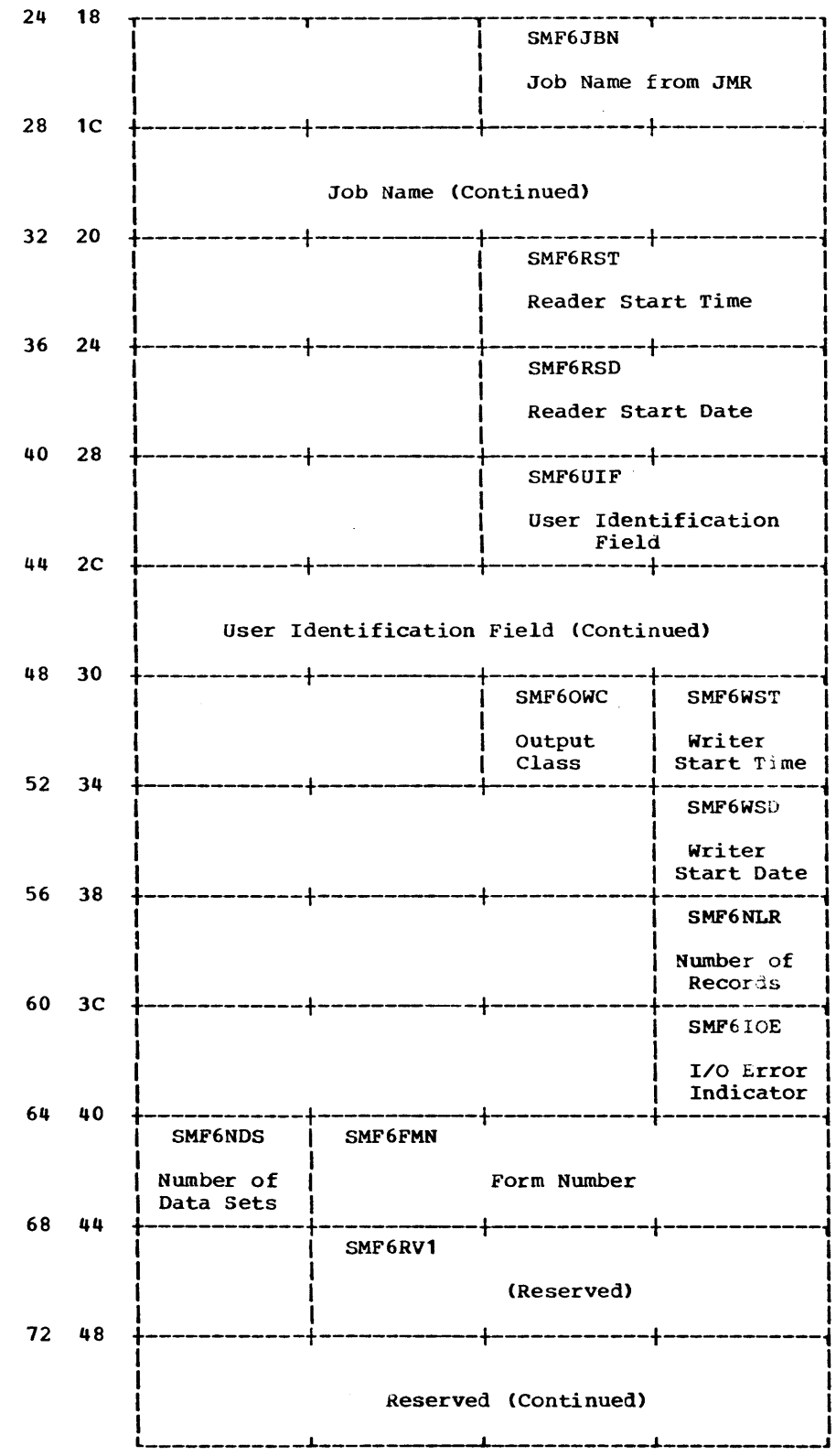
Offset	Bytes and Bits	Field Name	Description
200 (C8)	4	SMF26OPD	Output Processor stop date
204 (CC)	2	SMF26LN3	Actuals section length, including this field
206 (CE)	2	SMF26RV4	Reserved
208 (D0)	4	SMF26ICD	Number of input cards (JCL and data)
212 (D4)	4	SMF26XLN	Generated output lines
216 (D8)	4	SMF26XPU	Generated punched output
220 (DC)	4	SMF26RV5	Reserved
224 (E0)	4	SMF26PLN	Printed lines
228 (E4)	4	SMF26PPG	Printed pages
232 (E8)	4	SMF26PUC	Punched cards
236 (EC)	0	SMF26END	End of type 26 record
			SMF Subsystem Header Area
26 (1A)	2	SMFSSID	HASP subsystem ID (ORG SMFRSTRT)
27 (1B)	0000 0010	SMFHSPID	HASP subsystem identification
28 (1C)	2	SMFSSRSV	Reserved
30 (1E)	2	SMFSSLEN	Length of rest of record, not including this field
32 (20)	0	SMFSSTRT	Define start of subsystem portion
			SMF Type 43 Start HASP Record
32 (20)	3	SMF43RV1	Reserved
35 (23)	1	SMF43OPT	Start HASP options
	1...		Force format option
	.1..		Cold start option
	..1.		Request option
	...1		REP option
	.... 1...		REP list option
	.... .1..		Trace option
	11		Reserved
36 (24)	0	SMF43END	End of type 43 record

Offset	Bytes and Bits	Field Name	Description
			SMF type 45 stop HASP record
32 (20)	0	SMF45END	End of type 45 record (ORG SMFSSTRT)
			SMF type 47 start subsystem event
32 (20)	2	SMF47EVT	Event starting (ORG SMFSSTRT)
33 (21)	0000 0001	SMFRMTEV	HASP SIGNON subsystem event
33 (21)	0000 0010	SMFLINEV	HASP start line subsystem event
34 (22)	2	SMF47LN1	ID section length, (field included)
36 (24)	8	SMF47RMT	Remote name
44 (2C)	8	SMF47LIN	Line name
52 (34)	8	SMF47PSW	Password
60 (3C)	2	SMF47LN2	Message section length, including this field
62 (3E)	36	SFM47MSG	Message
98 (62)	0	SMF47END	End of type 47 record
			SMF type 48 stop subsystem event
32 (20)	2	SMF48EVT	Event stopping
33 (21)	0000 0001	SMFRMTEV	HASP SIGNOFF subsystem event
33 (21)	0000 0010	SMFLINEV	HASP stop line subsystem event
34 (22)	2	SMF48RV1	Reserved
36 (24)	8	SMF48RMT	Remote name
44 (2C)	8	SMF48LIN	Line name
52 (34)	8	SMF48PSW	Password
60 (3C)	0	SMF48END	End of type 48 record

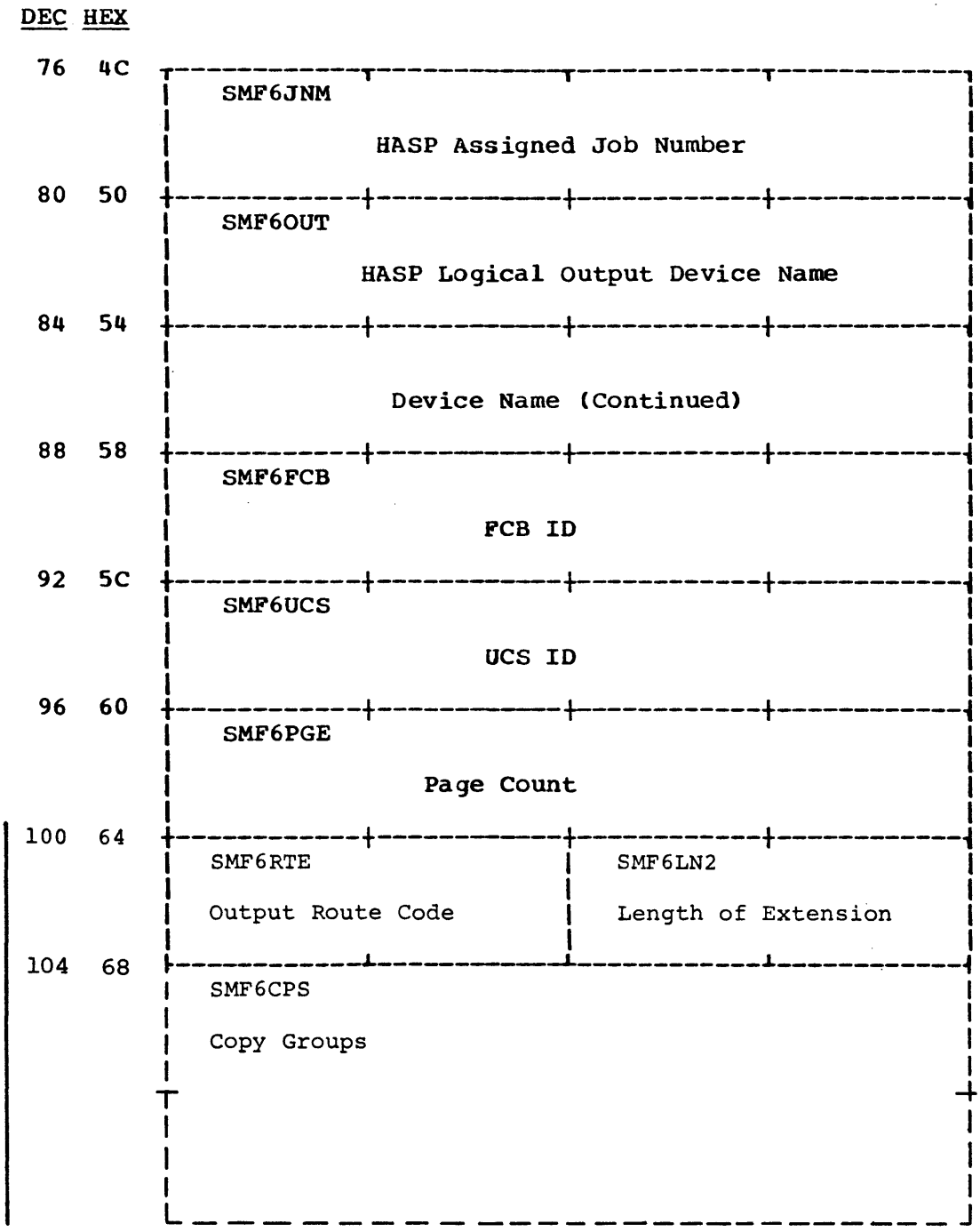
HASP SMF BUFFER



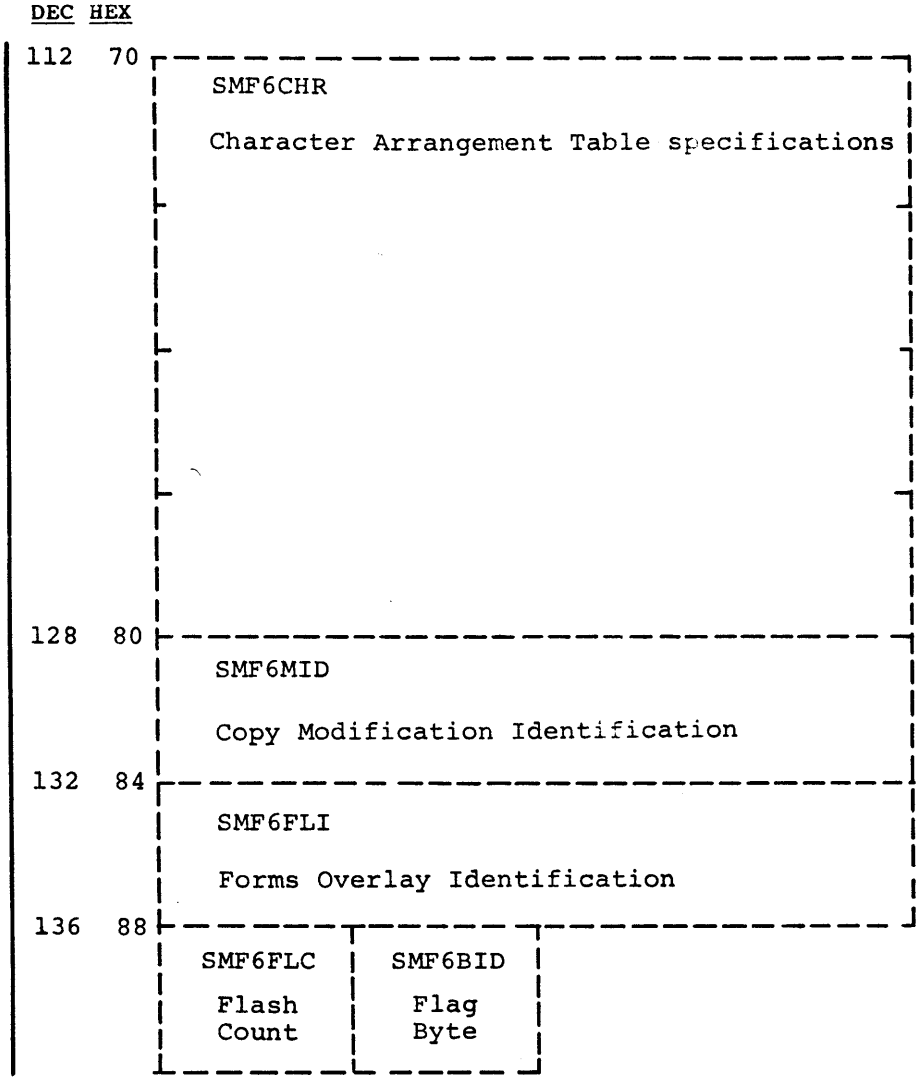
HASP SMF BUFFER



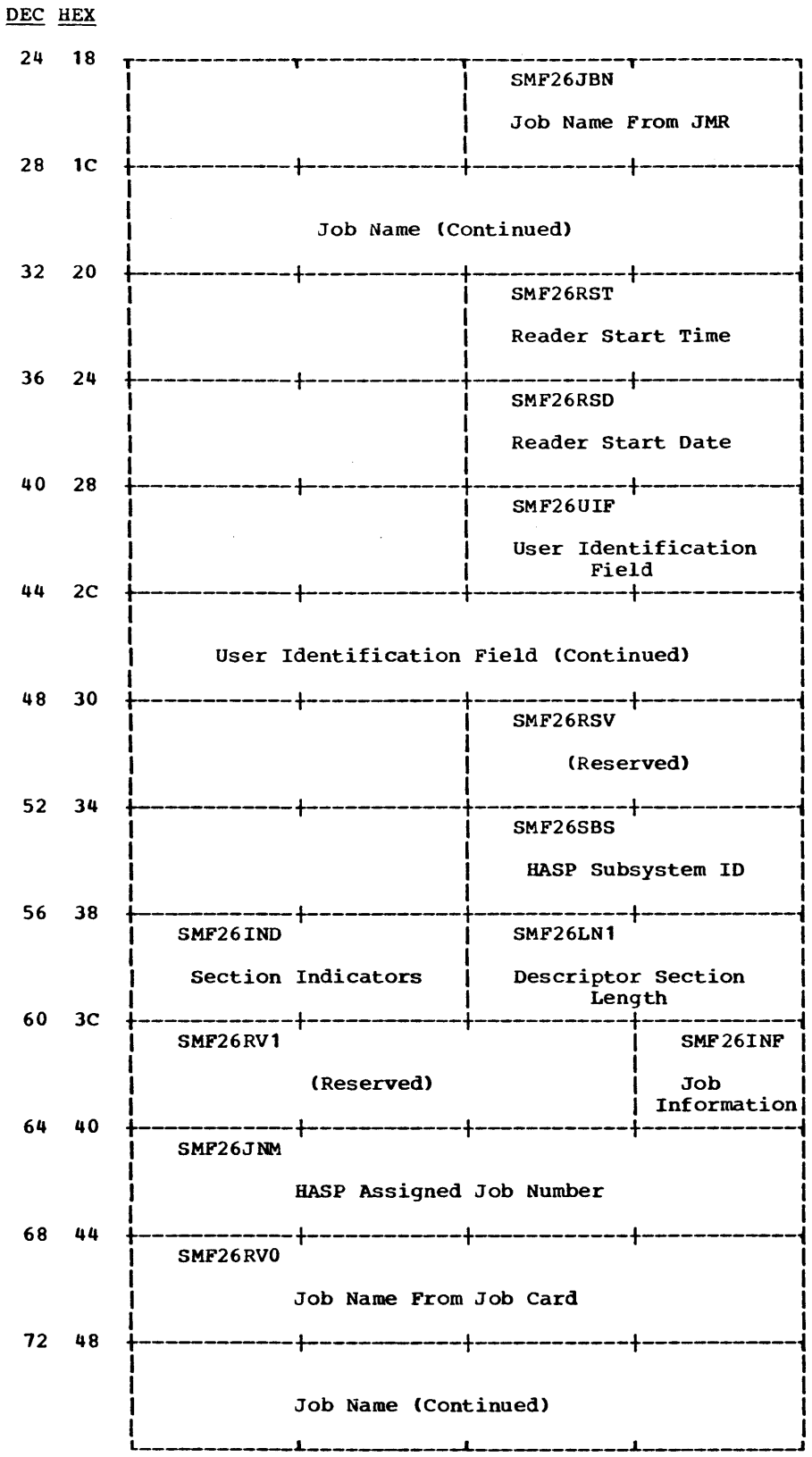
HASP SMF BUFFER



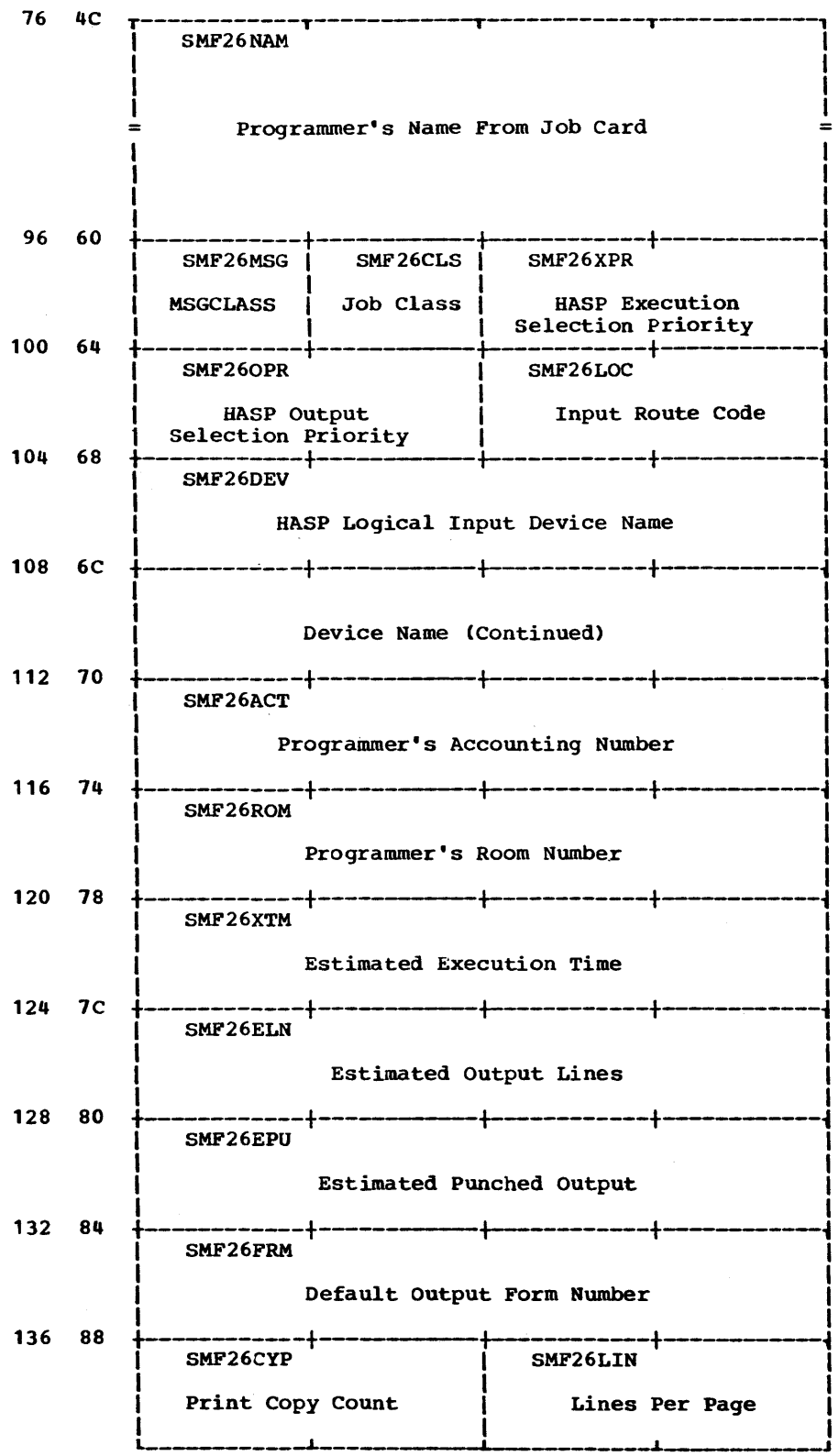
HASP SMF BUFFER



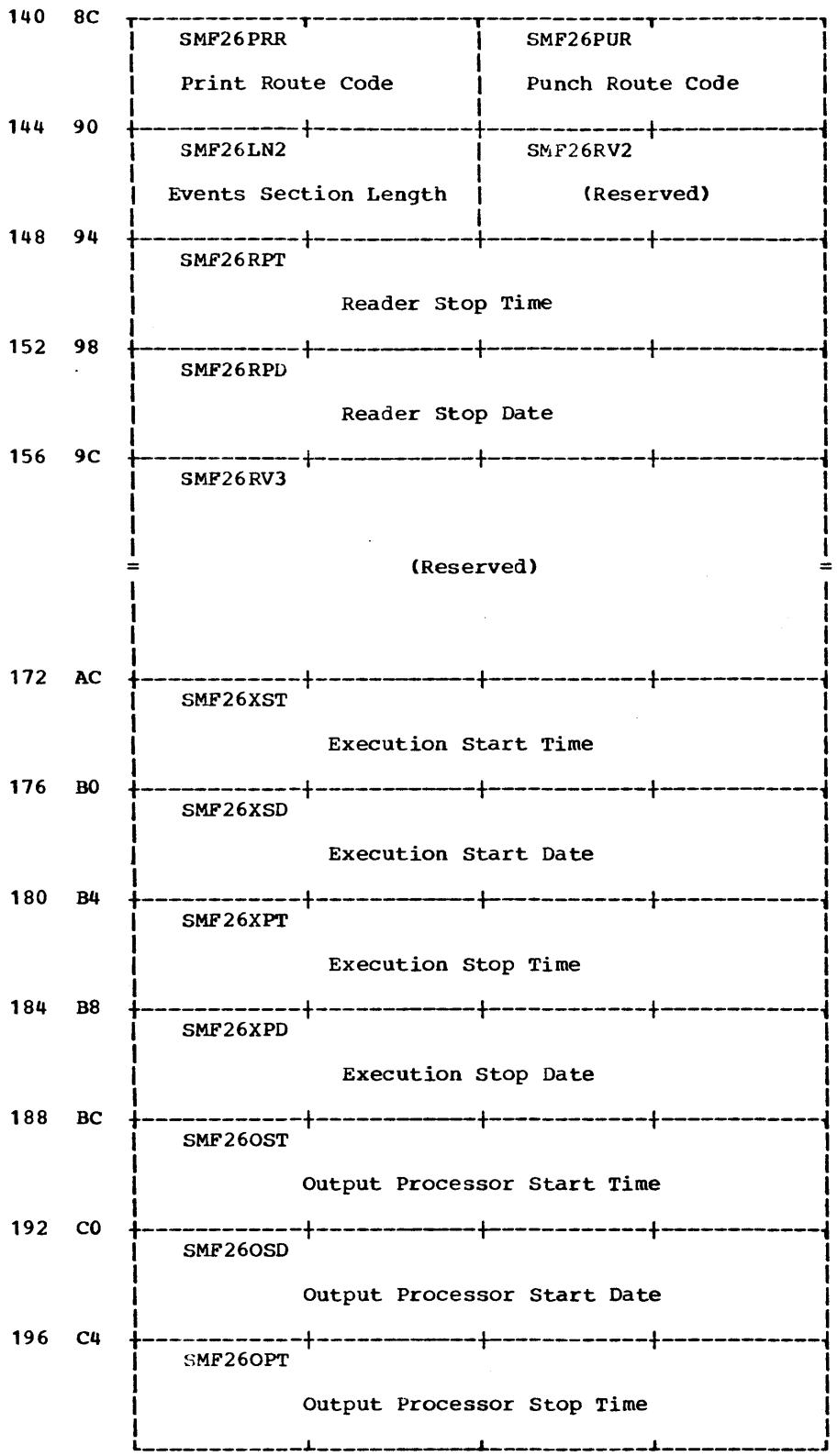
HASP SMF BUFFER



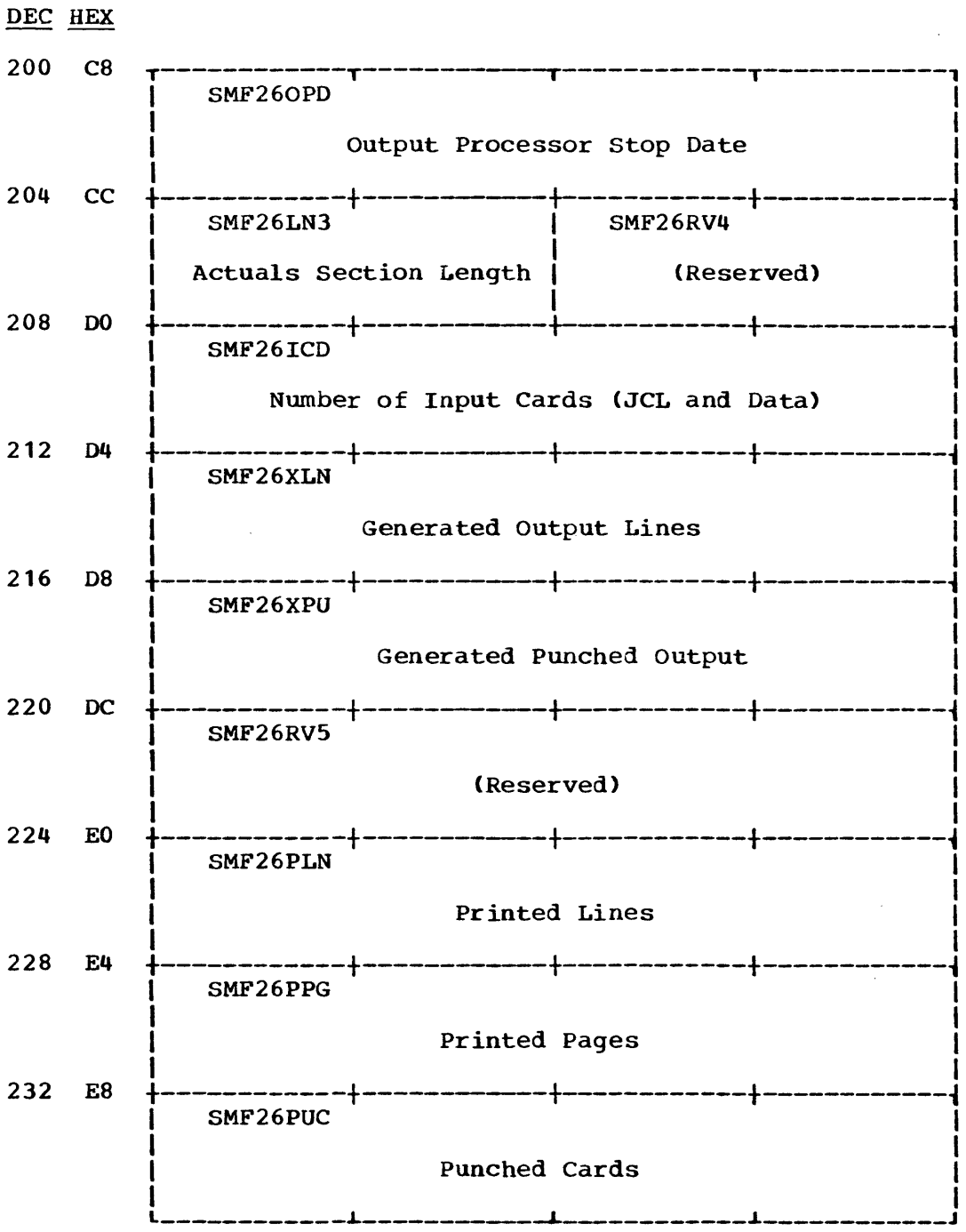
HASP SMF BUFFER



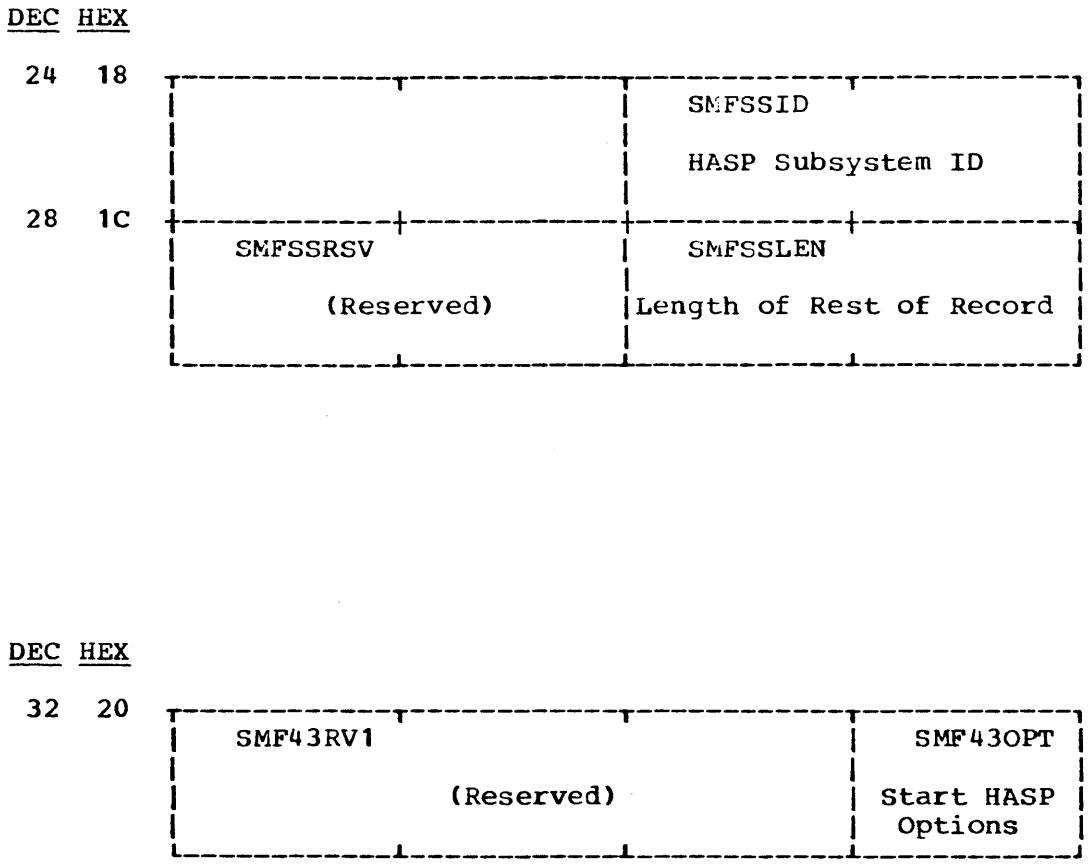
HASP SMF BUFFER



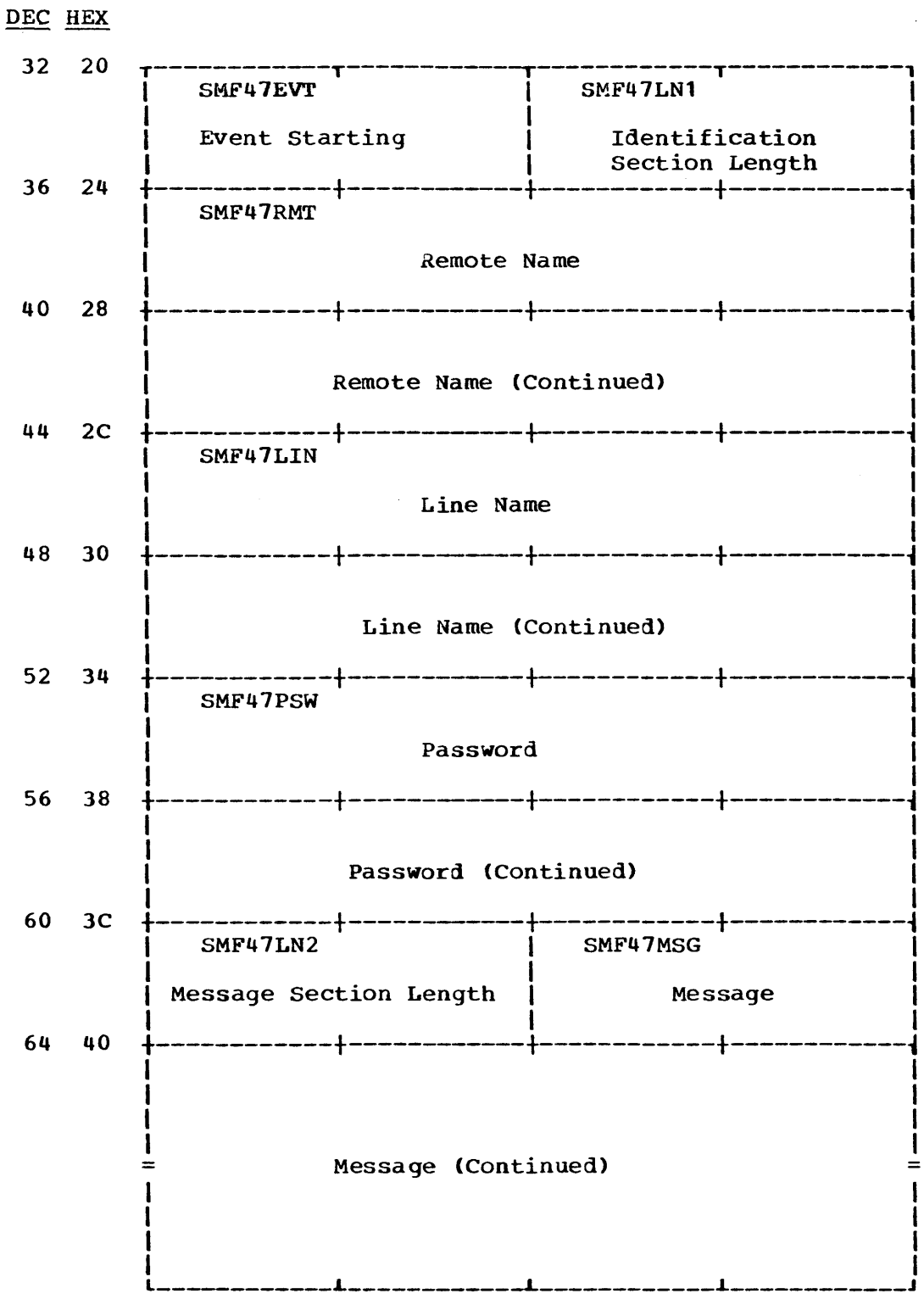
HASP SMF BUFFER



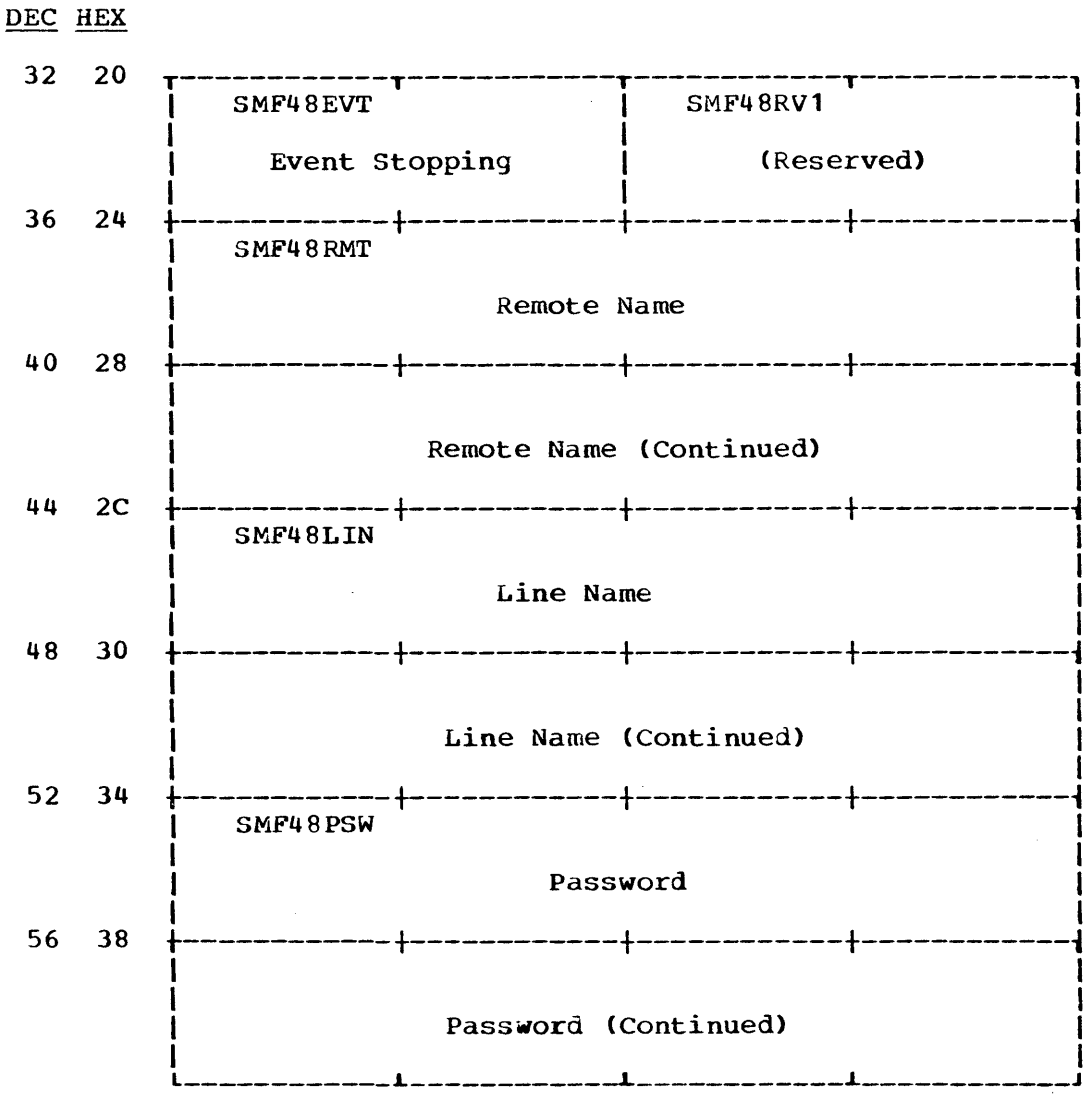
HASP SMF BUFFER



HASP SMF BUFFER



HASP SMF BUFFER

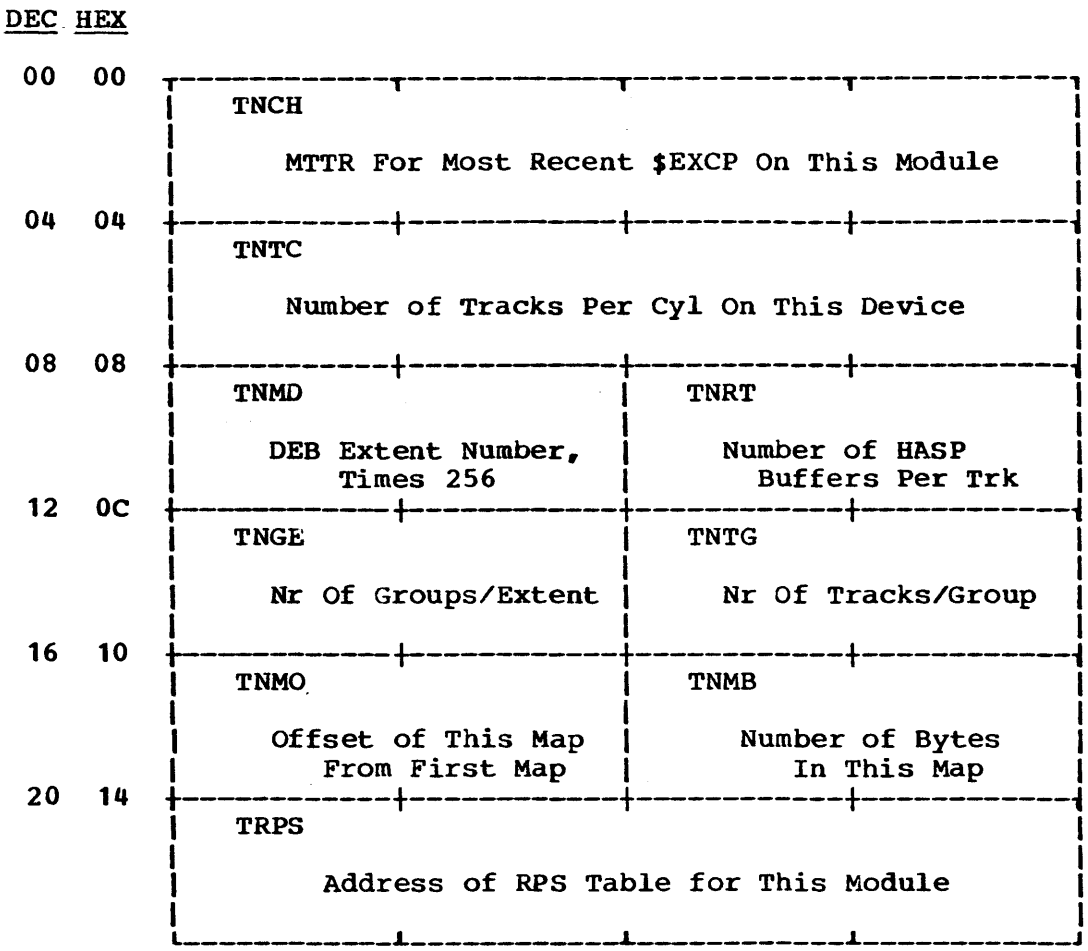


HASP TRACK EXTENT DATA TABLE

The Track Extent Data table (TED) defines for HASP direct-access allocation purposes a single SPOOL volume. It contains information about the physical volume obtained from the OS device characteristics table, about the track groups on the volume, and the position of the device's access mechanism.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	TNCH	Seek address of most recent $EXCP
04 (04)	4	TNTC	Number of tracks/cylinder this device
08 (08)	2	TNMD	Extent number, shifted left eight
10 (0A)	2	TNRT	Maximum record number
12 (0C)	2	TNGE	Number of track groups in extent
14 (0E)	2	TNTG	Number of tracks per group
16 (10)	2	TNMO	Offset of this map from first map
18 (12)	2	TNMB	Number of bytes in this map
20 (14)	0	TEDSIZE	Length of DSECT (EQU)

HASP TRACK EXTENT DATA TABLE



HASP TIMER QUEUE ELEMENT

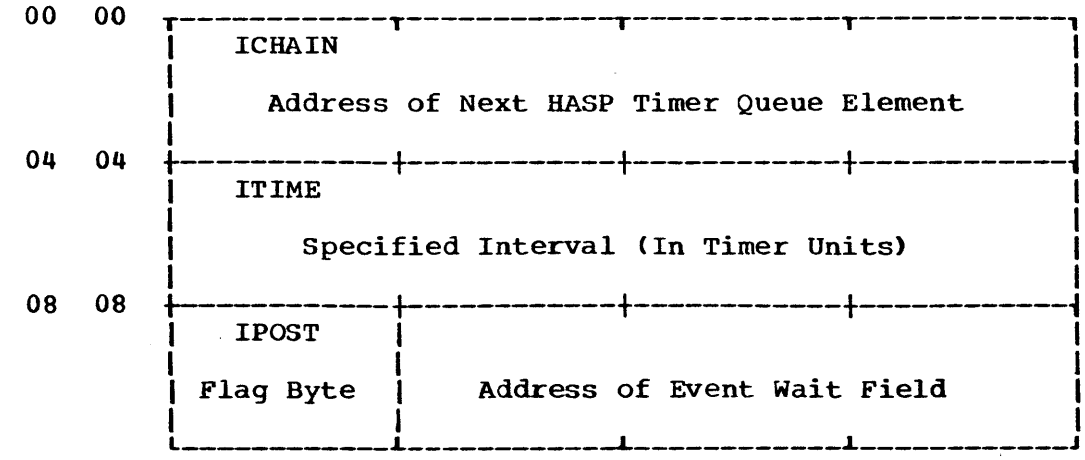
A HASP Timer Queue Element (TQE) is created by any HASP processor which needs to maintain a time interval independent of other intervals.

The processor requests interval time services by using the $STIMER and $TTIMER macro instructions which specify the address of a TQE. Interval Timer Supervisor routines maintain all active TQEs in a chain with the shortest unexpired interval first in the chain.

The TQE specifies the interval and the processor Event Wait Field to be posted for work when the interval expires.

Offset	Bytes and Bits	Field Name	Description
00 (00)	4	ICHAIN	Address of next Timer Queue Element
04 (04)	4	ITIME	Specified interval (in timer units)
08 (08)	4	IPOST	Flag byte and address of EWF to post

HASP TIMER QUEUE ELEMENT



HASP EXECUTION PCE WORK AREA

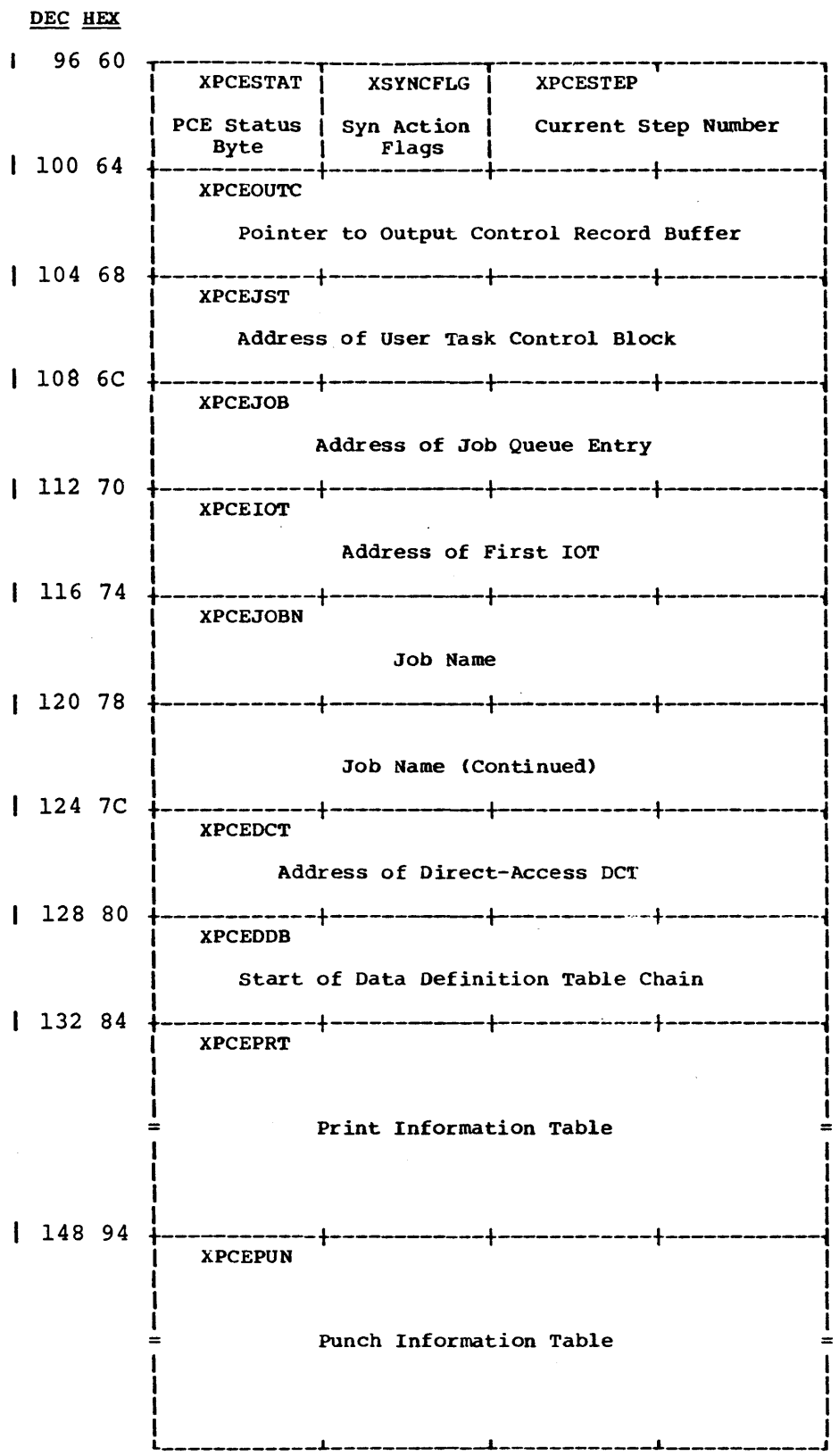
The Execution Work area (XEQWORK) is an extension to each execution PCE to contain information necessary to run a job. It contains such fields as job status flags, job name, JQE address, DDB chain pointer, and output statistics.

Offset	Bytes and Bits	Field Name	Description
96 (60)	1	XPCESTAT	PCE status byte
	1...	XDUPBIT	Job with duplicate name waiting
	.1..	XEOJMES	End-execution message sent
	..1.	XPOSTBIT	XTHAW should thaw XPCEJST
	...1	XREXREQ	Requeue for XEQ at end XEQ
	.... 1...	XSYNCREQ	Synchronous action required
	.... .1..	XOCRMES	OCR-overflow message sent
	.... ..1.	XPCERSV6	Reserved
	1	XPCERSV7	Reserved
97 (61)	1	XSYNCFLG	Synchronous action flags
	1...	XEOJBIT	Terminate job execution
	.1..	XALLOCWT	Write 'AWAITING ALLOCATION'
	..1.	XGETIOT	Get another IOT for reader/interpret
	...1	XIOTWREQ	IOT checkpoint required
	.... 1...	XOUTCDBF	Get a /*OUTPUT card buffer
	.... .1..	XOCRMSG	Write an OCR-overflow message
	.... ..1.	XSYNRSV6	Reserved
	1	XSYNRSV7	Reserved
98 (62)	2	XPCESTEP	Current step number
100 (64)	4	XPCEOUTC	Pointer to /*OUTPUT card buffer
104 (68)	4	XPCEJST	Address of user task control block
108 (6C)	4	XPCEJOB	Address of job queue entry
112 (70)	4	XPCEIOT	Address of first IOT
116 (74)	8	XPCEJOBN	Job name
124 (7C)	4	XPCEDCT	Address of direct-access DCT
128 (80)	4	XPCEDDB	Start of data definition table chain
132 (84)	16	XPCEPRT	Print Information Table
148 (94)	16	XPCEPUN	Punch Information Table

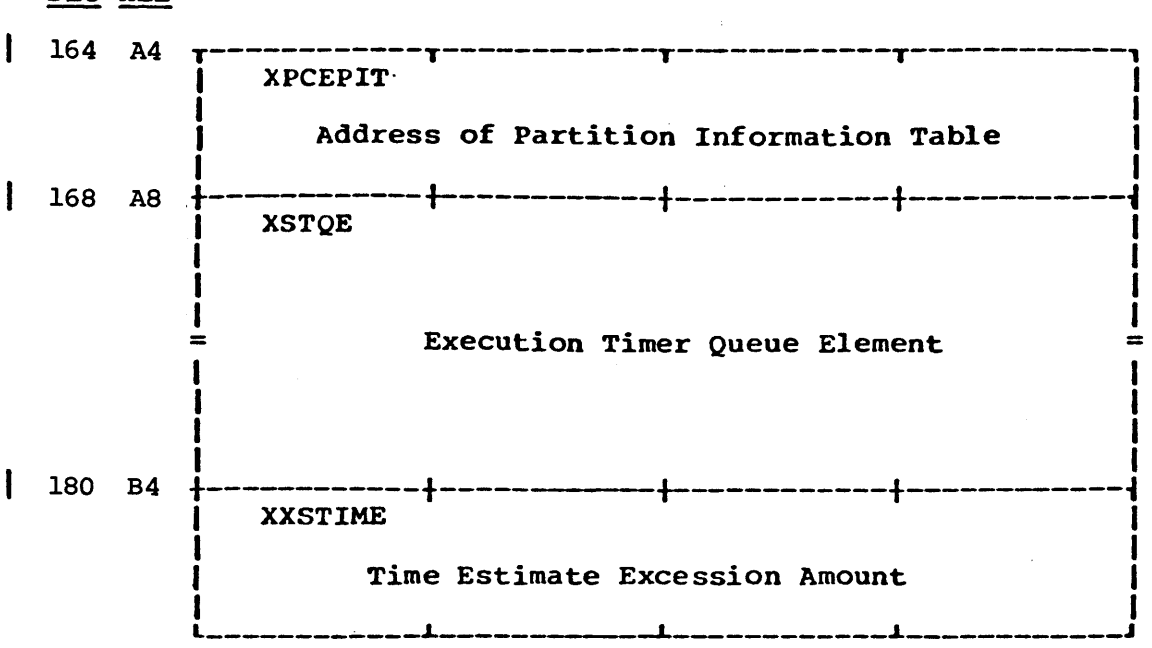
HASP EXECUTION PCE WORK AREA

Offset	Bytes and Bits	Field Name	Description
164 (A4)	4	XPCEPIT	Address of partition information table
168 (A8)	12	XSTQE	Execution Timer Queue Element
180 (B4)	4	XXSTIME	Time estimate excession amount
	0	XEQPCEWS	Length of execution PCE work area (EQU *-PCEWORK)

HASP EXECUTION PCE WORK AREA



HASP EXECUTION PCE WORK AREA



SECTION 6

HASP

DIAGNOSTIC AIDS

REGISTER USAGE

The use of all registers is consistent throughout HASP. The following list indicates for each register the register number, its equated name(s), and its use. Where a register has more than one equated name, the use depends on the equated name. The individual HASP modules can have other equates for work registers. HASP subroutines may use general registers 0, 1, and 15 without saving them.

Register	Name	Use
0	R0	Work and parameter register, contents may be destroyed by HASP subroutines
1	R1	Work and parameter register, contents may be destroyed by HASP subroutines
2	WA R2	Work register A
3	WB R3	Work register B
4	WC R4	Work register C
5	WD R5	Work register D
6	WE R6	Work register E and use by overlay
7	WF R7	Work register F
8	WG BASE3 R8	Work register G Overlay addressability register
9	R9	Generally available for user modifications. Care should be exercised when using this register for modifications.
10	JCT R10	JCT addressability register
11	BASE1 R11	HCT and HASPNUC addressability register
12	BASE2 R12	Processor addressability base register to be modified on entry to a module
13	SAVE R13	PCE addressability register

REGISTER USAGE

14	LINK R14	Link register
15	R15	Work register, contents may be destroyed by HASP subroutines.

STORAGE DUMP CONTAINING HASP

If the dump has OS control blocks formatted, e.g., as printed by IMPRDMP, the multiple task structure of HASP will be apparent. The main task is the load module named HASP. There will always be three subtasks present, HASPWTR, HASPWTO, and HASPACCT.

The first part of CSECT HASPNUC (which is first in the load module HASP) is the HASP Communication table (HCT). The HASP Vector Table ($HVT) is at the beginning of HCT. The majority of the formal interfaces with VS2 are exits from VS2 Release 1 to HASP coding. The addresses into HASP exist in $HVT, pointed to by CVTHJES, when HASP is active. The HCT contains a series of branches to HASP Control Service subprograms and contains most of the globally used status bytes, counters, control block chain pointers, and queue pointers.

The Processor Control Elements (PCEs) are in HASPNUC and can be found by looking in cell $PCEORG in the HCT or by following the OS save area chain from the system-provided first save area. PCEs are chained together exactly like OS save areas: forward chain in word 3, back chain in word 2. HASP uses the forward chain as its PCENEXT or dispatching chain.

It may be desirable to analyze the overlay status of HASP when working on an undetermined problem. The overlay areas should first be inspected.

A pointer to the first area can be found in cell $OACEADR in HASPNUC. All area control fields are defined in BUFDSECT. Subsequent areas are chained from OACECHN of the previous area.

The CSECT in each area can be determined by looking at OACENAME (which is the first four bytes read from disk) for the last three or four characters of the CSECT name. OACEOCON also identifies the routine assigned to the area. OCON can be interpreted by looking at the listing produced by HASPOBLD.

If any PCEs are using the routine, OACEPCE will be nonzero and will point to the first of a chain of PCEs. If BUFECBCC is zero, an uncompleted read operation was in progress to load the routine from disk.

Once the status of overlay areas is known, PCEs should be inspected for possible use of overlay. A PCE which is using a routine currently in memory should be on a chain which begins with OACEPCE of an area and continues through PCEOPCE of one or more PCEs. PCEOCON of each should match OACEOCON.

If a PCE is not on an area chain but does have the $EWFOLAY bit on in its PCEEWF, it should be on a queue beginning at cell $WAITACE in HASPNUC. This queue continues through PCEBASE3 of subsequent PCEs. Side chains of other PCEs from PCEOPCE may be present if several PCEs have equal PCEOCONS (requesting the same routine).

A PCE which has the $EWFOROL bit on in its PCEEWF is not on any chain (PCEOPCE is not meaningful), but its PCEOCON indicates the requested

routine. PCER15 may be a very small value which then represents the relative displacement into the routine at which execution will later resume.

PCEORTRN contains the return address from the original $LINK which invoked overlay. A PCE not in one of the three above states is not using overlay, and its PCEOCON is not meaningful.

HASP REPing at initialization time is a standard feature. &OREPSIZ must be set to reserve a small amount of storage if REPing of overlay CSECTs is desired. REP cards may specify absolute storage locations or locations in any CSECT, as taken from assembly listings.

For ZAPing overlay CSECTS, the SYSLIB card should reference the sequential overlay data set. The address for the CCHHR is taken from the listing produced by HASPOBLD. BASE adjusts for nonzero assembly origin of the CSECT. Most overlay CSECTs are shorter than the overlay record size of 1280 bytes. The space in the record following the last assembled location of the CSECT may be used as patch area. ABSDUMP cards may be used to dump the entire overlay data set or single CSECTs.

DEBUGGING TOOLS WITHIN HASP

HASP provides several debugging tools to help diagnose problems, as follows:

1. A trace routine can be invoked.

2. A patch space can be reserved by using the HASP macro $PATCHSP.

3. The REP facility can be used when HASP is started to make dynamic changes to HASP code.

4. The Catastrophic Error Handler routine can be used to take a dump when necessary.

5. The HASP $COUNT macro can be used to trace problems.

For further explanation of the debugging tools within HASP, refer to Section 3 HASP Trace Services, HASP Error Services, and the HASP REP Routine. In the appendix refer to the sections HASP Debug Services, HASP Error Services, and HASP Coding Aid Services.

SECTION 7

HASP APPENDIXES

APPENDIX A

HASP PROGRAMMER MACROS

HASP MACROS OVERVIEW

The HASP Control Service Programs provide a comprehensive set of services which aid the HASP Processors in performing their respective tasks in an efficient manner without burdening the processor programmer down with endless detail. These services are requested by the processor through the use of HASP macro instructions. The services are subdivided in this publication, as follows:

1. Buffer Services, which provide for the acquisition and release of HASP buffers.

2. Unit Services, which provide for the acquisition and release of HASP input/output units.

3. Job Queue Services, which provide the processors with an interface with the HASP job queue.

4. Direct-Access Space Services, which provide for the allocation and deallocation of HASP direct-access storage space.

5. Input/Output Services, which provide all communication with the Operating System Input/Output Supervisor.

6. Time Services, which provide for the setting and interrogation of the interval timer.

7. Overlay Services, which provide the capability to define and utilize sections of HASP that may optionally be made resident on direct-access storage and fetched into a dynamic area within HASP whenever required.

8. Synchronization Services, which provide synchronization and communication between HASP processors, the HASP Dispatcher, and the Operating System.

9. System Management Facilities Services, which provide the processors with an interface with the SMF services.

10. Debug Services, which provide facilities for aid in debugging HASP.

11. Error Services, which provide a uniform way of processing detected errors.

12. Coding Aid Services, which provide the HASP programmer with coding aids not usually available in the Operating System, but useful in coding HASP routines.

Some of the above services are provided by "in-line" code expansion wherever the macro instruction is used. The remainder of the services are provided by routines which are integral parts of the Control Service programs. For more information about these routines refer to Section 3. These routines are "linked to" by code generated wherever the macro instruction is used. At execution time, the macro expansion passes information to the control program routine to specify the exact nature

of the service to be performed. This information is broken down into parameters and, in general, is passed to the routine through general purpose registers called parameter registers.

The macro expansion can contain load instructions (LA,L,LH,etc.) that form parameters in parameter registers, and/or it can contain instructions which load parameter registers from registers loaded by the processor. The processor can also load parameters directly. Registers R1 and R0 are generally used as parameter registers.

Each parameter resulting from the expansion of a macro instruction is either an address or a value.

1. Address Parameter

   An address parameter is a standard 24-bit address. It is always located in the three low-order bytes of a parameter register. The high-order byte in the parameter register should contain all zeros. Any exception to this rule will be stated in the individual macro instruction description.

   An address parameter is always an effective address. The Control Service program is never given a 16-bit or 20-bit explicit address of the form D(B) or D(B,X) and then required to form an effective address. When an effective address is to be resolved, it is formed either by the macro expansion or before the macro instruction is issued.

2. Value Parameter

   A value parameter is a field of data other than an address. It is of variable length and is usually in the low-order bits of a parameter register. The value parameter will always have a binary format. The high-order unused bits in the parameter register should contain all zeros. An exception to this rule will be stated in the individual macro instruction description.

   Certain value parameters can be placed in a register along with another parameter, which can either be an address or a value parameter. In this case, a value parameter will be in other than the low-order bits. Two or more parameters in the same register are called packed parameters.

3. Operands

   Parameters are specified by operands in the macro instruction. An address parameter can result from a relocatable expression or, in certain macro-instructions, from an implied or explicit address. A value parameter can result from an absolute expression or a specific character string. Address and value parameters can both be specified by operand written as an absolute expression enclosed in parentheses. This operand form is called register notation. The value of the expression designates a register into which the specified parameter must

be loaded by the processor before the macro instruction is issued. The contents of this register are then placed in a parameter register by the macro expansion.

The processor programmer writes an operand in a HASP macro instruction to specify the exact nature of the service to be performed. Operands are of two types:

1. Positional Operands

   A positional operand is written as a string of characters. This character string can be an expression, an implied or explicit address, or some special operand form allowed in a particular macro instruction.

   Positional operands must be written in a specific order. If a positional operand is omitted and another positional operand is written to the right of it, the comma that would normally have preceded the omitted operand must be written. This comma should be written only if followed by a positional operand; it need not be written if it would be followed by a keyword operand or a blank.

   In the following examples, EX1 has three positional operands. In EX2, the second of three positional operands is omitted, but must still be delimited by commas. In EX3, the first and third operands are omitted; no comma need be written to the right of the second operand.

```
EX1     $EXAMP      A,B,C

EX2     $EXAMP      A,,C

EX3     $EXAMP      ,B
```

2. Keyword Operand

   A keyword operand is written as a keyword immediately followed by an equal sign and an optional value.

   A keyword consists of one through seven letters and digits, the first of which must be a letter. It must be written exactly as shown in the macro instruction description.

   An optional value is written as a character string in the same way as a positional operand.

   Keyword operands can be written in any order, but they must be written to the right of any positional operands in the macro instruction.

   In the following examples, EX1 shows two keyword operands. EX2 shows the keyword operands written in a different order and to the right of any positional operands. In EX3, the second and

third positional operands are omitted; they need not be delimited by commas, because they are not followed by any positional operands.

```
EX1     $EXAMP      KW1=X,KW2=Y

EX2     $EXAMP      A,B,C,KW2=Y,KW1=X

EX3     $EXAMP      A,KW1=X,KW2=Y
```

Certain operands are required in a macro instruction, if the macro instruction is to make a meaningful request for a HASP executor service. Other operands are optional, and can be omitted. Whether an operand is required or optional is indicated in the macro instruction descriptions.

BASIC NOTATION USED TO DESCRIBE MACRO INSTRUCTIONS

HASP macro instructions are presented in this section by means of macro instruction descriptions, each of which contains an illustration of the macro instruction format. This illustration is called a format description. An example of a format description is as follows:

```
[symbol]    $EXAMP      name1-value mnemonic,name2-CODED VALUE,

                        KEYWD1=value mnemonic,KEYWD2=CODED VALUE
```

Operand representations in format descriptions contain the following elements:

1. An operand name, which is a single mnemonic word used to refer to the operand. In the case of a keyword operand, the keyword is the name. In the case of a positional operand, the name is merely a reference. In the above format description, name1, name2, KEYWD1, and KEYWD2 are operand names.

2. A value mnemonic, which is a mnemonic used to indicate how the operand should be written if it is not written as a coded value. For example, "addr" is a value mnemonic that specified that an operand or optional value is to be written as either a relocatable expression or register notation.

3. A coded value, which is a character string that is to be written exactly as it is shown. For example, RDR is a coded value.

The format description also specifies when single operands and combinations of operands should be written. This information is indicated by notational elements called metasymbols. For example, in the preceding format description, the brackets around "symbol" indicate that a symbol in this field is optional.

Operand Representation

Positional operands are represented in format descriptions in one of two ways:

1. By a 3-part structure consisting of an operand name, a hyphen, and a value mnemonic. For example: name1-addr.

2. By a 3-part structure consisting of an operand name, a hyphen, and a coded value. For example: name1-RDR.

Keyword operands are represented in format descriptions in one of two ways:

1. By a 3-part structure consisting of a keyword, an equal sign, and a value mnemonic. For example: KEYWD1=addr.

2. By a 3-part structure consisting of a keyword, an equal sign, and a coded value. For example: KEYWD1=RDR.

The most significant characteristic of an operand representation is whether a value mnemonic or coded value is used; these two cases are discussed below.

Operands With Value Mnemonics

When a keyword operand is represented by:

KEYWORD=value mnemonic

the programmer first writes the keyword and the equal sign and then a value of one of the forms specified by the value mnemonic.

When a positional operand is represented by:

name-value mnemonic

the programmer writes only a value of one of the forms specified by the value mnemonic. The operand name is merely a means of referring to the operand in the format description; the hyphen simply separates the name from the value mnemonic. Neither is written.

The following general rule applies to the interpretation of operand representations in a format description; anything shown in upper-case letters must be written exactly as shown; anything shown in lower-case letters is to be replaced with a value provided by the programmer. Thus, in the case of a keyword operand, the keyword and equal sign are written as shown, and the value mnemonic is replaced. In the case of a positional operand, the entire representation is replaced.

The value mnemonics listed below specify most of the allowable operand forms that can be written in HASP macro instructions. Other value mnemonics, which are rarely used, are defined in individual macro instruction descriptions.

1. Symbol - the operand can be written as a symbol.

2. Relexp - the operand can be written as a relocatable expression.

3. Addr - the operand can be written as (1) a relocatable expression or (2) register notation designating a register that contains an address in its three low-order bytes. The designated register must be one of the registers 2 through 12, unless special register notation is used.

4. Addrx - the operand can be written as (1) an indexed or nonindexed implied or explicit address or (2) register notation designating a register that contains an address in its three low-order bytes. An explicit address must be written as in the RX form of an assembler language instruction.

5. Adval - the operand can be written as (1) an indexed or nonindexed implied or explicit address or (2) register notation designating a register that contains a value. An explicit address must be written as in the RX form of an assembler language instruction.

6. Absexp - the operand can be written as an absolute expression.

7. Value - the operand can be written as (1) an absolute expression or (2) register notation designating a register that contains a value in its three low-order bytes.

8. Text - the operand can be written as a character constant as in a DC data definition instruction. The format description shows explicitly if the character constant is to be enclosed in single quotation marks.

9. Code - the operand can be written as one of a large set of coded values; these values are defined in the macro instruction description.

Coded Value Operands

Some operands are not represented in format descriptions by value mnemonics. Instead, they are represented by one or more upper-case character strings that show exactly how the operand should be written. These character strings are called coded values, and the operands for which they are written are called coded value operands.

A coded value operand results in either a specific value parameter or a specific sequence of executable instructions.

If a positional operand can be written as any one of two or more coded values, all possible coded values are listed and are separated by vertical stroke indicating that only one of the values is to be used.

Metasymbols

Metasymbols are symbols that convey information to the programmer, but are not written by him. They assist in showing the programmer how and when an operand should be written. The metasymbols used in this section are:

1. | This is a vertical stroke and means "or". For example, A|B means either the character A or the character B.

2. {} These are braces and denote grouping. They are used most often to indicate alternative operands. For example:

   {YES|NO}

   In the example above, either YES or NO must be written.

3. [] These are brackets and denote options. Anything enclosed in brackets can either be omitted or written once in the macro instruction. For example:

   [YES|NO]

   In the example above, YES, or NO, or neither can be written. The underlining indicates that, if neither is written, YES is assumed.

SPECIAL REGISTER NOTATION

If an operand of a HASP macro instruction is written using register notation, the resulting macro expansion loads the parameter contained in the designated register into either parameter register R1 or parameter register R0.

For example, if an operand is written as (R15), and if the corresponding parameter is to be passed to the control program in register R1, the macro expansion could contain the instruction:

```
LR   R1,R15
```

The processor can load parameter registers directly, before the execution of the macro expansion; this is called preloading. The programmer specifies that preloading will occur by writing an operand as either "(R1)" or "(R0)" this is called special register notation. This notation is special for two reasons:

1. The register notation designation of registers R1 and R0 is generally not allowed.

2. The designation must be made by the specific four characters "(R1)" or "(R0)", rather than by the general form of an absolute expression enclosed in parentheses. For example, even though the absolute symbol RONE could be equated to R1,

"(RONE)" must not be written instead of "(R1)" if special register notation is intended. If this were done, the macro expression would contain a useless instruction:

```
LR   R1,RONE
```

The format description shows whether special register notation can be used, and for which operands. This is demonstrated by the following example:

```
[symbol]   $EXAMP   {abc-addrx | (R1)} , {def-addrx | (R0)}
```

Both operands can be written in the addrx form, and therefore can be written using register notation. Ordinary register notation indicates that the parameter register should be loaded from the designated register by the macro expansion. The format description also shows that the abc operand can be written as "(R1)", and the def operand can be written as "(R0)". If either of these special register notations is used, the processor must have loaded the designated parameter register before the execution of the macro instruction.

REGISTER TRANSPARENCY

In general, the following registers cannot be considered transparent across a HASP macro expansion and the associated link to the Control Service program:

1. LINK
2. R14
3. R15
4. R0
5. R1.

All other registers will be transparent unless specifically stated in the individual macro instruction description.

$GETBUF - Acquire A HASP Buffer From The HASP Buffer Pool or RJE Buffer From The RJE Buffer Pool

The $GETBUF macro instruction obtains a buffer from the HASP or RJE buffer pool and returns the address of this buffer in register R1.

Format Description:

```
[symbol]   $GETBUF  [none-relexp] [,TYPE=TP] [,OLAY=YES]
```

none
specifies a location to which control will be returned if there are no buffers available.

If this operand is omitted, the condition code will be set to reflect the availability of a buffer as follows:

1. CC=0 - no buffer is available.

2. CC≠0 - R1 contains the address of the buffer.

TYPE=TP
specifies that the buffer is to be obtained from the RJE buffer pool rather than the HASP buffer pool.

OLAY=YES
must be specified if the $GETBUF macro instruction is coded physically within an overlay segment.

$FREEBUF - Return A HASP Buffer To The HASP Buffer Pool Or RJE Buffer To The RJE Buffer Pool

The $FREEBUF macro instruction is used to return a HASP buffer to the HASP buffer pool or RJE buffer to the RJE buffer pool.

Format Description:

```
[symbol]    $FREEBUF    {buffer-addrx | (R1)} [,OLAY=YES]
```

buffer
specifies either a pointer to a buffer or the address of a buffer to be returned to the buffer pool as follows:

1. If "buffer" is written as an address, then it represents the address of a full word which contains the address of the buffer to be returned in its three low-order bytes.

2. If "buffer" is written using register notation (either regular or special register notation), then it represents the address of the buffer to be returned.

3. If register notation is used, the address must have been loaded into the designated register before the execution

of this macro instruction.

OLAY=YES
must be specified if the $FREEBUF macro instruction is coded physically within an overlay segment.

Caution: The specified buffer must have been obtained by a $GETBUF macro instruction. The action of the macro instruction as well as future $GETBUF and $FREEBUF macro instructions is unpredictable in other cases.

JOB QUEUE SERVICES

The HASP Job Queue consists of a chain of Job Queue Elements and can be divided into five logical queues. These five logical queues are represented by the following symbolic names:

Symbolic Name	Logical Job Queue
$INPUT	Queue of jobs in input processing
$XEQ	Queue of jobs awaiting O/S Execution Phase
$OUTPUT	Queue of jobs awaiting Output Phase
$HARDCPY	Queue of jobs that are in the Job Output Table
$PURGE	Queue of jobs awaiting Purge Phase

For more information concerning the formats of the HASP Job Queue Element and the HASP Job Information Table Element, refer to section 5 under JQE and JIT.

$QADD - Add Job Queue Element To The HASP Job Queue

The $QADD macro instruction adds an element to the HASP Job Queue, placing it in the specified logical queue. The address of the associated Job Information Table Entry is returned in register "R0".

Format Description:

```
[symbol]   $QADD   {element-addrx | (R1)}, {queue-value | (R0)}
                   [,full-relexp] [,OLAY=YES]
```

element
specifies the address of an element which is to be added to the HASP Job Queue.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

queue
specifies the logical queue in which the Job Queue Element is to be placed. This value must always be one of the five logical queue types.

If register notation is used, one of these values must have been

loaded into the designated register before the execution of this macro instruction.

full
specifies a location to which control will be returned if the HASP Job Queue is full.

If this operand is omitted, the condition code will be set to reflect the status of the HASP Job Queue as follows:

1. CC=0 - the queue is full and the element cannot be accepted.

2. CC≠0 - the element was successfully added to the queue. "R0" contains the address of the associated JIT Entry.

OLAY=YES
must be specified if the $QADD macro instruction is coded physically within an overlay segment.

$QGET - Obtain Job Queue Element From The HASP Job Queue

The $QGET macro instruction obtains a Job Queue Element from the specified logical queue of the HASP Job Queue and returns the address of this element in register "R1". The address of the associated Job Information Table Entry is returned in register "R0".

Format Description:

```
[symbol]    $QGET    {queue-value | (R1)} [,none-relexp]
                     [,PRROUTE=YES] [,PUROUTE=YES] [,OLAY=YES]
```

queue
specifies the logical queue from which the Job Queue Element is to be obtained. This value must always be one of the five logical queue types.

If register notation is used, one of these values must have been loaded into the designated register before the execution of this macro instruction.

none
specifies a location to which control will be returned if the specified logical queue is empty.

If this operand is omitted, the condition code will be set as follows:

1. CC=0 - the specified logical queue is empty.

2. CC≠0 - "R1" contains the address of a Queue Element from the specified logical queue and "R0" contains the address

of the associated JIT Entry.

PRROUTE=YES
specifies that bits 0-7 of register "R0" contain a route code which must match the route code (QUEPRTRT) of the Job Queue Element obtained.

PUROUTE=YES
specifies that bits 8-15 of register "R0" contain a route code which must match the route code (QUEPUNRT) of the Job Que Element obtained.

OLAY=YES
must be specified if the $QGET macro instruction is coded physically within an overlay segment.

&QPUT - Return Job Queue Element To The HASP Job Queue

The $QPUT macro instruction returns a Job Queue Element to the HASP Job Queue, placing it in the specified logical queue. The address of the associated Job Information Table Entry is returned in register "R0".

Format Description:

```
[symbol]   $QPUT   {element-addrx | (R1)}, [queue-value | (R0)}

                   [,OLAY=YES]
```

element
specifies the address of an element which is to be returned to the HASP Job Queue.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

queue
specifies the logical queue in which the Job Queue Element is to be placed. This value must always be one of the five logical queue types.

If register notation is used, one of these values must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
must be specified if the $QPUT macro instruction is coded physically within an overlay segment.

CAUTION: The specified Job Queue Element must have been previously obtained with a $QGET macro instruction or the action of the $QPUT macro instruction is unpredictable.

PROGRAMMING NOTE: The $QPUT macro instruction cannot be used to change the priority of a Job Queue Element. If a change of priority is desired, the $QREM and $QADD macro instructions must be used.

$QREM - Remove Job Queue Element From The HASP Job Queue

The $QREM macro instruction removes a specified Job Queue Element from the HASP Job Queue.

Format Description:

```
[symbol)    $QREM    {element-addrx | (R1)} [,OLAY=YES]
```

element
specifies the address of an element which is to be removed from the HASP Job Queue.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
must be specified if the $QREM macro instruction is coded physically within an overlay segment.

CAUTION: The specified Job Queue Element must have been previously obtained with a $QGET macro instruction or the action of the $QREM macro instruction is unpredictable.

$QSIZ - Determine Number Of Elements In A Logical Queue

The $QSIZ macro instruction determines the number of Job Queue Elements in a specified logical queue of the HASP Job Queue and returns this value in register "R1".

Format Description:

```
[symvol]    $QSIZ    {queue-value | (R1)} [,none-relexp]

                     [,PRROUTE=YES] [,PUROUTE=YES] [,OLAY=YES]
```

queue
specifies the logical queue which is to be counted. This value must always be one of the five logical queue types.

If register notation is used, one of these values must have been loaded into the designated register before the execution of this macro instruction.

none
specifies a location to which control will be returned if the

specified logical queue is empty.

If this operand is omitted, the condition code will be set to reflect the status of the specified logical queue as follows:

1. CC=0 - the specified queue is empty (R1=0).

2. CC≠0 - the specified queue contains at least one Job Queue Element (R1 = number of elements in queue).

PRROUTE=YES
specifies that bits 0-7 of register "R0" contain a route code which must match the route code (QUEPRTRT) of all jobs counted.

PUROUTE=YES
specifies that bits 8-15 of register "R0" contain a route code which must match the route code (QUEPUNRT) of all jobs counted.

OLAY=YES
must be specified if the $OSIZ macro instruction is coded physically within an overlay segment.

$QLOC - Locate Job Queue Element For Specific Job

The $QLOC macro instruction locates the Job Queue Element associated with the job with the specified job number and returns the address of this element in register "R1". The address of the associated Job Information Table Entry is returned in register "R0".

Format Description:

```
[symbol]    $QLOC    {jobno-adval | (R1)} [,none-relexp]
                     [,OLAY=YES]
```

jobno
specifies the binary job number associated with the job for which the Job Queue Element is being searched.

If an address is used it specifies the address of a half-word that contains the binary job number. This half-word must be located on a half-word boundary.

If register notation is used, the binary job number must have been loaded into the designated register before the execution of this macro instruction.

none
specifies a location to which control will be returned if the specified job number is not locatable in the HASP Job Queue.

If this operand is omitted, the condition code will be set to reflect the status of register "R1" as follows:

1. CC=0 - the specified job is not locatable.

2. CC≠0 - the specified job is locatable and "R1" contains the address of the associated Job Queue Element, and "R0" contains the address of the associated JIT Entry.

OLAY=YES
must be specified if the $QLOC macro instruction is coded physically within an overlay segment.

$GETUNIT - Acquire A Unit Device Control Table (DCT)

The $GETUNIT macro instruction obtains a Device Control Table (DCT) for a specified type of unit, and returns the address of this DCT in register R1.

Format Description:

```
[symbol]   $GETUNIT   type-code [,none-relexp] [,OLAY=YES]
```

type
specifies the type of unit for which a DCT is to be obtained. The values for this operand and their meanings are:

- DA - Direct-Access DCT
- LNE - Line DCT
- RDR - Card Reader DCT
- RJR - Remote Reader DCT
- INR - Internal Reader DCT
- PRT - Printer DCT
- RPR - Remote Printer DCT
- PUN - Punch DCT
- RPU - Remote Punch DCT

none
specifies a location to which control will be returned if there are no available Device Control Tables for the specified device. If this operand is omitted, the condition code will be set to reflect the availability of a DCT as follows:

1. CC=0 - no DCT is available.
2. CC≠0 - R1 contains the address of a DCT of the specified type.

OLAY=YES
must be specified if the $GETUNIT macro instruction is coded physically within an overlay segment.

$FREUNIT - Release A Unit Device Control Table (DCT)

The $FREUNIT macro instruction is used to release a Device Control Table (DCT).

Format Description:

```
[symbol]   $FREUNIT     {dct-addrx | (R1)} [,OLAY=YES]
```

dct
specifies either a pointer to a DCT or the address of a DCT to be released as follows:

If "dct" is written as an address, then it represents the address of a full word which contains the address of the DCT to be released in its three low-order bytes. This word must be located on a full-word boundary in core.

If "dct" is written using register notation (either regular or special register notation), then it represents the address of the DCT to be released.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
must be specified if the $FREUNIT macro instruction is coded physically within an overlay segment.

Caution: The specified DCT must have been obtained by a $GETUNIT macro instruction. The action of the macro instruction is unpredictable in other cases.

$TRACK - Acquire A Direct-Access Track Address

The $TRACK macro instruction obtains a track address on a HASP committed direct-access device and returns this track address in register R1.

Format Description:

```
[symbol]   $TRACK   {allocmap-addrx | (R1)} [,OLAY=YES]
```

allocmap
: Specifies the address of a track allocation map from which the direct-access space is to be allocated. This allocation map includes a 2-word header in front of the actual allocation bit map.

  If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
: must be specified if the $TRACK macro instruction is coded physically within an overlay segment.

$PURGE - Return Direct-Access Space

The $PURGE macro instruction is used to return the direct-access space which has been allocated for a given job.

Format Description:

```
[symbol]   $PURGE   {bitmap-addrx | (R1)} [,OLAY=YES]
```

bitmap
: specifies the address of a track allocation map containing the direct-access space to be returned. This allocation map does not include the 2-word header used for track allocation.

  If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
: must be specified if the $PURGE macro instruction is coded physically within an overlay segment.

$EXCP - Execute HASP Channel Program

The $EXCP macro instruction initiates HASP input/output activity.

Format Description:

```
[symbol]   $EXCP    {dct-addrx | (R1)} [,OLAY=YES]
```

dct
specifies either a pointer to a Device Control Table (DCT) or the address of a DCT which represents a device upon which input/output activity is to be initiated.

If "dct" is written as an address, it represents the address of a full word which contains the address of the DCT in its three low-order bytes. This word must be located on a full-word boundary.

If "dct" is written using register notation (either regular or special register notation), it represents the address of the DCT.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES
must be specified if the $EXCP macro instruction is coded physically within an overlay segment.

$EXTP - Initiate Remote Terminal Input/Output Operation

The $EXTP macro instruction initiates a remote terminal input/output action or operation.

Format Description:

```
[symbol]   $EXTP   type-code, {dct-addrx | (R1)}
                   [,{loc-addrx | (R0)}] [,OLAY=YES]
```

type
specifies the type of operation as follows:

1. OPEN - Initiate remote terminal processing.
2. GET - Receive one record from the remote terminal.
3. PUT - Send one record to the remote terminal.
4. CLOSE - Terminate remote terminal processing.

dct
specifies either a pointer to a DCT or the address of a DCT

which represents the remote terminal device.

If "dct" is written as an address, it represents the address of a full word which contains the address of the remote terminal device DCT in its three low-order bytes. This word must be located on a full-word boundary in core.

If "dct" is written using register notation (either regular or special register notation), it represents the address of the remote terminal device DCT.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

loc

If "type" specifies either "OPEN" or "CLOSE", this parameter should not be specified.

If "type" specifies "GET", this parameter specifies the address of an area into which the input record will be placed. The input area must be defined large enough to contain the largest record to be received.

If "type" specifies "PUT", this parameter specifies the address of a CCW which contains the carriage control (or stacker select), address, and length of the record to be written.

If register notation is used, the appropriate address must have been loaded into the designated register before the execution of this macro instruction.

OLAY=YES

must be specified if the $EXTP macro instruction is coded physically within an overlay segment.

$WTO - HASP Write To Operator

The $WTO macro instruction initiates output activity on one or more of the devices designated as operator consoles.

Format Description - Standard Form:

```
[symbol]   $WTO     {message-addrx | (R1)} , {length-value | (R0)}

                    [,JOB={YES|NO}] [,WAIT={YES|NO}]

                    [,CONVERT={YES|NO}] [,ROUTE=code] [,CLASS=code]

                    [,PRI=code]
```

Format Description - Execute Form:

```
[symbol]    $WTO      {message-addrx | (R1)} [,{length-value | (R0)}]
                      , MF=(E,name)
```

Format Description - List Form:

```
[name]    $WTO    [length-value,] MF=L [,JOB={YES|NO}]

                  [,WAIT={YES|NO}] [,CONVERT={YES|NO}]

                  [,ROUTE=code] [,CLASS=code] [,PRI=code]
```

message
specifies the address of a message which is to be written on the designated console(s).

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

length
specifies the length of the above message.

If register notation is used, the value must have been loaded into the low-order byte of the designated register before the execution of the macro instruction. The rest of the register must be zero unless the message is being sent to a remote terminal (see below).

Note: When using the execute and list forms of the macro instruction, the length can be specified on either form but must not be specified on both.

JOB
specifies whether the characters "JOB nnnn" will be appended to the start of the messages as follows:

1. YES - The job number will be appended to the start of the message.

2. NO - The job number will not be appended to the message.

If this operand is omitted, JOB=YES will be assumed.

Caution: Unless JOB=NO is specified, the JCT register must be loaded with the address of the Job Control Table before the execution of this macro instruction, or the job number printed will be unpredictable.

WAIT
specifies the action to be taken in the event that no Console Message Buffers are available as follows:

1. YES - Return will not be made until a Console Message Buffer has become available and the message has been queued.

2. NO - An immediate return will always be made with the condition code set as follows:

   a. CC=0 - No Console Message Buffers were available. The message was not accepted and the macro instruction must be reissued.

   b. CC≠0 - The message was accepted.

If this operand is omitted, WAIT=YES will be assumed.

Note: Unless WAIT=NO is specified, the message to be issued must be constructed in a reenterable area of storage.

Caution: WAIT=NO must be specified if the $WTO macro instruction is physically coded within an overlay segment. However, if the message is not accepted because no console message buffers are available, you should issue the $WAIT macro instruction with the CMB event-code before reissuing the $WTO request.

CONVERT
specifies the type of consoles indicated as follows:

1. YES - Logical consoles have been specified (e.g., $LOG) and these must be converted to physical consoles by the Control Service program.

2. NO - Physical consoles have been specified and no conversion is necessary. (This specification is reserved for Command Processor responses to commands)

If this operand is omitted, CONVERT=YES will be assumed.

ROUTE
specifies the console or consoles on which the above message is to be written. The code consists of the absolute sum of one or more of the logical console designations in the following list:

Designation	Console Specified
$LOG	System log console(s)
$ERR	Error console(s)
$UR	Unit record operations area
$TP	Teleprocessing operations area
$TAPE	Tape operations area
$MAIN	Chief operator's area
$ALL	All of the above consoles
$REMOTE	Remote terminal console

Note: If "$REMOTE" is specified, no other consoles should be specified, the register form of "length" must be specified, and the remote terminal number must be loaded into bits 16-23 of the register used to specify the length before the execution of the macro instruction. Bits 0-15 of this register must be zero.

If no ROUTE is specified, the "$LOG" console will be assumed.

Caution: The designation "$ALL" should not be used in conjunction with any other console but should be specified alone. Failure to observe this rule will give unpredictable results.

CLASS
specifies the class of the message as one of the following:

1. $DOMACT - The message requires immediate action and is always written.
2. $ALWAYS - The message is essential and should always be written.
3. $ACTION - The message requires eventual operator action.
4. $NORMAL - The message is considered important to normal computer operations.
5. $TRIVIA - The message is considered unimportant to normal computer operations.

If no CLASS is specified, $NORMAL will be assumed.

The $DOMACT specification is reserved for $WTOs issued to logical consoles. On return from $WTO processing, R1 will contain the address of the Console Message Buffer (CMB) containing the message. The CMB will be retained in the system until a corresponding $DOM is executed using the returned pointer.

PRI
specifies the priority of the message as one of the following:

1. $HI - High priority
2. $ST - Standard priority
3. $LO - Low priority.

If no PRI is specified, $ST priority will be assumed.

$DOM - HASP Delete Operator Message

The $DOM macro instruction releases the Console Message Buffer (CMB) containing an immediate action message given to the Operating System via SVC35. $DOM also causes the Operating System copy of the message to be deleted.

Format Description:

```
[symbol]    $DOM    [CMB={(R1)| cmb-addrx}]
```

CMB
specifies the address of the CMB returned by the execution of a "$WTO CLASS=$DOMACT" statement.

Caution: The $DOM macro must be executed once and only once for each CMB retained in the system awaiting the execution of the macro.

$TIME - Request Time Of Day

The $TIME macro instruction obtains the time of day and returns this time in register R0. The time is returned as an unsigned 32-bit binary number in which the least significant bit has a value of 0.01 second.

Format Description:

```
[symbol]   $TIME   [OLAY=YES]
```

OLAY=YES
must be specified if the $TIME macro instruction is coded physically within an overlay segment.

The time returned is the time of day based on a 24-hour clock.

$STIMER - Set Interval Timer

The $STIMER macro instruction sets an interval into a programmed interval timer.

Format Description:

```
[symbol]   $STIMER {loc-addrx | (R1)} [,OLAY=YES]
```

loc
specifies the address of a HASP Timer Queue Element. Before this macro instruction is executed, the Timer Queue Element must be initialized as follows:

1. ITIME must be initialized with the interval to be set in the following manner:

   a. If "x" seconds are desired, the ITIME should be set to "x"

   b. If "y" hundredth-seconds (0.01 seconds) are desired, then ITIME should be set to the two's complement of "y".

2. IPOST must be initialized with the address of the Event Wait Field to be posted.

If register notation is used, the address must have been loaded into the designated register before the execution of this macro instruction.

Refer to Section 5 under TQE for more information about the HASP Timer Queue Element.

OLAY=YES
must be specified if the $STIMER macro instruction is coded physically within an overlay segment.

Programming Note: An unlimited number of independent $STIMER time intervals can be active at any time provided that each has been furnished with a unique HASP Timer Queue Element.

$TTIMER - Test Interval Timer

The $TTIMER macro instruction obtains the time remaining in the associated time interval that was previously set with a $STIMER macro instruction. The value of the time interval remainder is returned in register R0 in seconds (rounded to the nearest second). The $TTIMER macro instruction can also be used to cancel the associated time interval.

Format Description:

```
[symbol]    $TTIMER {loc-addrx | (R1)} [,CANCEL]
                    [,OLAY=YES]
```

loc
specifies the address of the Timer Queue Element.

If register notation is to be used, the address must have been loaded into the designated register before the execution of this macro instruction.

CANCEL
specifies that the interval in effect should be cancelled.

If this operand is omitted, processing continues with the unexpired portion of the interval still in effect.

If the interval expired before the $TTIMER macro instruction was executed, the CANCEL operand has no effect.

OLAY=YES
must be specified if the $TTIMER macro instruction is coded physically within an overlay segment.

HASP OVERLAY PROGRAMMING RULES

The following comments summarize the rules for coding and using "overlayable code" in HASP. All rules apply to use of any control sections created by use of the $OVERLAY macro, even if the code so produced is optionally made permanently resident as part of the overlay build process. HASP Overlay does not use any overlay facility defined elsewhere in OS documentation. More precise details of overlay macro syntax are given later in this section. The Overlay Build process, Overlay Service, and Overlay Roll internal logic are described in Section 3.

Creating Overlay Control Sections

The beginning of a portion of HASP executable coding or tables to be made overlayable is indicated by the $OVERLAY macro. By convention, the name field begins with "HASP" and continues with up to four more characters. The fifth character (first after "HASP") usually indicates the processor of which the overlayable code is a part; e.g., R for read, X for execution, P for print/punch, etc. A specific example is "HASPXJI1", the name of the first of three overlays used by the HASP Execution Processor for job initiation actions. The name coded with $OVERLAY will be defined at the first location coded by the programmer after the $OVERLAY and will be used to derive a name for the control section created.

The operands of $OVERLAY specify the priority for use of overlay resources and, in conjunction with the HASPGEN parameter &OLAYLEV, whether the code created is to be actually disk or main storage resident during HASP operation.

The $OVERLAY macro is a functional replacement for CSECT, USING, and BALR or L when creating a HASP overlayable control section. $OVERLAY creates an actual assembly control section and indicates local addressability in register BASE3. Overlay Service and Roll functions ensure that the proper base value is loaded into BASE3 when an overlay section is being used.

An overlay control section's coding may be terminated and all effects of a previous $OVERLAY cancelled in one of two ways. Another overlay may be begun by a new $OVERLAY macro. Nonoverlay coding may be resumed by DROPing register BASE3 and reestablishing an appropriate CSECT.

If it is desired to add more coding to a previously terminated overlay section, the actions in the following example must be performed. &xyz is a properly declared variable symbol. HASPabcd is the overlay name chosen by the programmer. Other symbols are defined in standard HASP assemblies. The second statement must be placed after the $OVERLAY defining the overlay section to be resumed, before another $OVERLAY is used.

```
HASPabcd  $OVERLAY  12,0     (original definition)
&xyz      SETC '&OSECT'
            .
            .
            .
&xyz      CSECT             (later additional code)
          USING  HASPabcd-OACEPROG+BUFDSECT,BASE3
```

Calling Overlay Routines

The three executable macros $LINK, $XCTL, and $LOAD cause an overlay routine to be made available for use in addressable storage. The single operand of each of these macros gives the name of the overlay to be used, either directly or by providing (in register form) the address of a $OCON macro which gives the name. The name referenced is that used with a $OVERLAY macro to create the overlay routine. The overlay control section ($OVERLAY and following code) may be either in the same or in a different HASP assembly as a macro which calls it.

The $LINK and $LOAD macros must be physically placed in nonoverlay CSECTs and executed only when no other overlay routine is being used, i.e., nested calling of overlays is not defined. With $LINK, program control is eventually passed to the first instruction after $OVERLAY of the called routine. The address of the caller's next instruction is saved for later return. $LOAD returns control to the next instruction after $LOAD when the routine is available in storage.

$XCTL relinquishes use of an overlay routine, previously called by $LINK or $XCTL, and calls a new overlay routine which is entered as if called by $LINK. The return address saved by the original $LINK is not altered. $XCTL must always be executed when an overlay is in use, but may physically be in an overlay routine or in nonoverlay coding, subject to the requirements given later under "Coding While Using Overlay Routines."

$RETURN and $DELETE both relinquish use of an overlay routine, which must be in use when they are executed. These macros have no operands; the routine released is the only one in use at the time. $RETURN causes control to pass to the next instruction after the $LINK previously executed by the processor from nonoverlay code. $RETURN, like $XCTL, may physically reside anywhere. $DELETE must physically reside in nonoverlay code and is valid only after a routine was previously called by $LOAD. Control continues following $DELETE, after use of the overlay routine has been released.

Overlay routines may be called only by HASP processors operating under the primary HASP TCB, HASP Dispatcher, and PCE control. They may not be called from HASP subtasks. Overlay routines may not be called in exits from the Asynchronous Input/Output Processor.

Coding While Using Overlay Routines

On entry to an executable overlay by $LINK or $XCTL, or after loading an overlay with $LOAD, the caller's registers R0-R7 and R9-R13 are preserved. However, registers BASE3 (same as R8 or WG in unmodified HASP), LINK, R15, and the condition code are destroyed and are not later restored. While an overlay routine is being used (after the execution of $LINK or $LOAD, but before the execution of $RETURN or $DELETE), the program must not alter the value of register BASE3.

Coding in an overlay routine is "covered" by local addressability provided by $OVERLAY. Coding physically outside an overlay but referring to it (usual case after a $LOAD) must be covered by a USING like that in the example under "Creating Overlay Control Sections." Other addressability (e.g., BASE1, BASE2) remains in effect if not dropped and may be used.

Program control may be transferred out of or into an overlay routine and its storage may be retrieved, as long as overlay control of that routine is in effect (has not been released by $RETURN, $DELETE, or $XCTL to a new routine) and proper addressability is maintained. References to locations in an overlay routine from physically outside the overlay at any other time are illegal.

Relocatable valued A- or V-type constants must not be physically coded in overlay routines. Such constants may be coded in nonoverlay CSECTs and referenced from overlay routines. Relocatable A- or V-type literals may be coded if the literal pool containing them is not physically in an overlay routine. An A or V constant or literal containing an "unpaired" (see Assembly Language SRL) reference to a symbol defined in an overlay routine is always illegal, regardless of location.

When use of an overlay routine is released by $RETURN or $DELETE, only the LINK and BASE3 registers are destroyed. All other registers and the condition code are preserved as set prior to the execution of these macros.

Total size of all coding in an overlay routine must not exceed the value of the internal assembly variable &OLAYSIZ, currently set at 1280 bytes in unmodified HASP. An error message will be produced during the Overlay Build process for each routine which violates this restriction. Routines which are too long because of modification, etc., may be temporarily tested by forcing them to be resident in the HASP load module, as described in the HASP System Programmer's Guide. Such routines should, of course, be reduced to within the maximum size by recoding or splitting into multiple overlays.

Overlay Location Independent Coding

Whenever a HASP processor which is using an overlay routine executes $WAIT, regardless of the physical location of the $WAIT, the Overlay Roll Processor may preempt the overlay area for other use. When control is returned to the processor following the $WAIT, the overlay routine may have been reread from direct access, destroying all self-modification or temporary storage in the overlay, and may be in a

different overlay area, making all address values relative to the overlay routine's location invalid (in registers or elsewhere).

The first effect above (destruction of temporary storage) is similar to the effect on single (nonreentrant) temporary storage in nonoverlay coding used by multiple processors when $WAIT is executed. The effect on overlay storage may take place when only one PCE is using an overlay routine. Reentrant temporary storage (e.g., in a PCE work area) or reconstruction from known values after $WAIT will avoid errors due to this possible "refreshing" of overlay routines.

The second effect (changing overlay location) is, or course, peculiar to use of overlay routines. System Overlay Service and Roll logic automatically makes proper adjustments to registers BASE3 (overlay routine base value) and R15 ($WAIT reentry address) if the $WAIT is physically in the overlay routine.

Other address values relative to an overlay routine are usually created in registers by use of instructions such as LA (with BASE3 as base), BAL, or BALR (the last two if physically in an overlay routine). These registers should be made relative prior to $WAIT by "SLR n,BASE3" instruction(s) and made absolute after $WAIT by "ALR n,BASE3" instruction(s). Equivalent techniques may be created for other coding situations.

Certain HASP macros which call services subroutines represent a "hidden" possible $WAIT. They must be treated as equivalent to $WAIT in all cases previously described. Specifically, any macro for which the keyword parameter OLAY=YES is defined (see other appendixes for macro descriptions) represents a hidden $WAIT, regardless of physical location. The OLAY=YES is coded only if the macro physically exists in an overlay routine. Macro expansion and service subroutine exit coding handle possible adjustment of the LINK register. The services subroutines assume that all parameter address values (in R0, R1, or R15) are not relative to an overlay routine. Other addresses relative to an overlay routine must be adjusted before and after the service macro call by the caller.

The $WTO macro is a special case. It represents a hidden $WAIT unless WAIT=NO is coded. If coded physically in an overlay routine, WAIT=NO must be coded. It may be coded physically outside an overlay routine without WAIT=NO, but then registers must be treated as for macros which have OLAY=YES defined.

HASP OVERLAY MACROS

$OVERLAY - Define Overlay Segment

The $OVERLAY macro instruction defines the instructions which follow it as an overlay segment and defines the name, priority, and residence susceptibility factor of this overlay segment.

Format Description:

HASPname-symbol $OVERLAY prio-value [,resfact-value]

HASPname
specifies the name to be assigned to the overlay segment. The first four characters must be the characters "HASP". The last four characters can be any unique combination of alphameric characters.

prio
specifies the priority of the overlay segment as follows:

1. 0 - Lowest priority
2. &LOW - Low priority
3. &MED - Medium priority
4. &HIGH - High priority.

resfact
specifies the residence susceptibility factor of the overlay segment as follows:

1. 0 - Never resident
2. &LOW - Resident only if &OLAYLEV<4
3. &MED - Resident only if &OLAYLEV<8
4. &HIGH - Resident only if &OLAYLEV<12.

If this parameter is omitted, a resident factor of 0 will be used.

Note: This parameter may be overridden at the time that the overlay library is built.

$OCON - Define Overlay Constant

The $OCON macro instruction defines an overlay constant (OCON) for use in conjunction with other overlay macro-instructions.

Format Description:

[symbol] $OCON HASPname-symbol

HASPname
specifies the name of an overlay segment.

$LINK - Link To An Overlay Segment

The $LINK macro instruction is used to link to an overlay segment from a nonoverlay segment.

Format Description:

```
[symbol]    $LINK    {HASPname-symbol | (register)}
```

HASPname
specifies the name of the overlay segment to which control is to be transferred.

If register notation is used, the register specified must be loaded with the address of an overlay constant (OCON) which represents the overlay segment to which control is to be transferred.

$XCTL - Transfer Control To Another Overlay Segment

The $XCTL macro instruction is used to transfer control from one overlay segment to another.

Format Description:

```
[symbol]    $XCTL    {HASPname-symbol | (register)}
```

HASPname
specifies the name of the overlay segment to which control is to be transferred.

If register notation is used, the register specified must be loaded with the address of an overlay constant (OCON) which represents the overlay segment to which control is to be transferred.

$RETURN - Return From An Overlay Segment

The $RETURN macro instruction is used to return control from an overlay segment to a nonoverlay segment.

Format Description:

```
[symbol]    $RETURN
```

$LOAD - Load An Overlay Segment

The $LOAD macro instruction is used to load an overlay segment from a nonoverlay segment. The address of the overlay area into which the overlay segment has been loaded is returned in register BASE3.

Format Description:

```
[symbol]    $LOAD    {HASPname-symbol | (register)}
```

HASPname
specifies the name of the overlay segment to be loaded.

If register notation is used, the register specified must be loaded with the address of an overlay constant (OCON) which represents the overlay segment to be loaded.

$DELETE - Delete A Loaded Overlay Segment

The $DELETE macro instruction is used to delete an overlay segment which has been loaded with a $LOAD macro instruction.

Format Description:

```
[symbol]    $DELETE
```

$ACTIVE - Specify Processor Is Active

The $ACTIVE macro instruction indicates to the HASP Dispatcher that the associated processor is processing a job or task.

Format Description:

```
[symbol]   $ACTIVE   [R=register]
```

R
specifies the register which is to be used by the $ACTIVE macro instruction.

If R is omitted, register R1 will be used.

$DORMANT - Specify Processor Is Inactive

The $DORMANT macro instruction indicates to the HASP Dispatcher that the associated processor has completed the processing of a job or task and is now going into a "dormant" state.

Format Description:

```
[symbol]   $DORMANT   [R=register]
```

R
specifies the register which is to be used by the $DORMANT macro instruction.

If R is omitted, register R1 will be used.

Caution: The $DORMANT macro instruction should never be executed unless a corresponding $ACTIVE has been executed for the same processor.

$WAIT - Wait For A HASP Event

The $WAIT macro instruction places the associated processor in a HASP wait condition and specifies the event for which the processor is waiting in the Processor Control Element Event Wait Field.

Format Description:

```
[symbol]   $WAIT   event-code [,ENABLE] [,OLAY=YES]
```

event
specifies the event upon which the processor is waiting as one of the following:

1. BUF - waiting for a HASP Buffer.
2. TRAK - waiting for a direct-access track address.

3. JOB - waiting for a job.
4. UNIT - waiting for a Device Control Table.
5. CKPT - waiting for the completion of a HASP checkpoint.
6. CMB - waiting for a Console Message Buffer.
7. SMF - waiting for an SMF Buffer.
8. JOT - waiting for Job Output Table service.
9. OPER - waiting for an operator response.
10. IO - waiting for the completion of an input/output operation.
11. WORK - waiting to be redirected.
12. HOLD - waiting for a $S operator command.
13. DDB - waiting for a Data Definition Table.

ENABLE
specifies that the system mask in the PSW should be set to all ones prior to returning to the HASP Dispatcher.

OLAY=YES
need not be specified.

$POST - Post A HASP Event Complete

The $POST macro instruction indicates a HASP event is complete by turning off the specified bit in the indicated Event Wait Field.

Format Description:

```
[symbol]   $POST   ewf-relexp,event-code
```

ewf
specifies the address of the Event Wait Field which is to be posted. This operand can also be written in the form D(B).

event
specifies the event which is to be posted as one of the following:

1. BUF - a HASP Buffer has been returned.
2. TRAK - direct-access space has been released.
3. JOB - a HASP Job Queue Element has changed status.
4. UNIT - a Device Control Table has been released.

5. CKPT - a HASP checkpoint has completed.
6. CMB - a Console Message Buffer has been returned.
7. SMF - an SMF Buffer has been returned.
8. JOT - the HASP Job Output Table has changed status.
9. OPER - an operator has responded.
10. IO - an input/output operation has completed.
11. WORK - a processor has been redirected.
12. HOLD - an operator has entered a $S command.
13. DDB - a Data Definition Table has been released.

Caution: The $POST macro instruction should not be executed unless addressability to the HASP Communication Table (HCT) has been established.

$ENABLE - Enable Interrupts

The $ENABLE macro instruction causes the specified interrupts to be enabled.

Format Description:

```
[symbol]    $ENABLE    mask-code [,OLAY=YES]
```

mask
specifies the interrupts to be enabled as follows:

ALL - Enable all interrupts.

OLAY=YES
need not be specified.

$DISABLE - Disable Interrupts

The $DISABLE macro instruction causes the specified interrupts to be disabled.

Format Description:

```
[symbol]    $DISABLE    mask-code [,OLAY=YES]
```

mask
specifies the interrupts to be disabled as follows:

1. ALL - Disable all interrupts.

2. INT - Disable external interrupts.

OLAY=YES
need not be specified.

$GETSMFB - Acquire A HASP SMF Buffer From The HASP SMF Buffer Pool

The $GETSMFB macro instruction obtains a buffer from the HASP SMF buffer pool and returns the address of this buffer in register R1 or returns a zero in register R1 if no buffers were available and WAIT=NO was specified in the macro.

Format Description:

```
[symbol]    $GETSMFB    [WAIT={YES|NO}] [,OLAY={YES|NO}]
```

WAIT
specifies the action to be taken in the event no HASP SMF buffers are available as follows:

1. YES - Control will not be returned to the caller until a HASP SMF buffer has become available.
2. NO - An immediate return will be made. If no buffers are available, register R1 will contain a zero upon return to the calling routine.

OLAY=YES
must be specified if the $GETSMFB macro instruction is coded physically within an overlay segment.

$QUESMFB - Queue A HASP SMF Buffer On A Busy Queue

The $QUESMFB macro instruction places the HASP SMF buffer address, pointed to by register R1, on a busy queue and POSTs HASPACCT.

Format Description:

```
[symbol]    $QUESMFB    [OLAY={YES|NO}]
```

OLAY=YES
must be specified if the $QUESMFB macro instruction is coded physically within an overlay segment.

Caution: The HASP SMF Services should not be used if the HASPGEN parameter &NUMSMFB is set to less than 2.

$TRACE - Make Entry In The HASP Trace Table

The $TRACE macro instruction makes an entry in the HASP Trace Table if the &TRACE option is set nonzero. If the &TRACE option is set to zero, this macro instruction does not generate any code.

Format Description:

```
[symbol]    $TRACE
```

Programming Note: The $TRACE macro expansion and associated Control Service Program preserve all registers and the condition code. For more information concerning the HASP Trace Table, refer to Section 3 under Trace Effector.

$COUNT - Count Selected Occurrences

The $COUNT macro instruction increments a counter every time the macro instruction is executed and can be used to determine the number of times a particular event occurs or a particular section of code is entered. The counter is a halfword (modulo 65,536) which is located 14 bytes deep in the macro expansion (symbol+14).

Format Description:

```
[symbol]    $COUNT    [R=register]
```

R
specifies a register to be used in performing the counting operation. If this parameter is omitted, register "R1" will be used.

$ERROR - Indicate Catastrophic Error

The $ERROR macro instruction is used to indicate that a catastrophic error has occurred, one that prevents any further processing by HASP. The macro instruction causes the following message to be printed out on the console using an ordinary OS WTO:

```
$ HASP SYSTEM CATASTROPHIC ERROR.  CODE = symbol
```

HASP then executes a 1-instruction loop.

Format Description:

```
symbol    $ERROR
```

symbol
consists of a 4-character symbol indicating the type of error which occurred.

This operand usually consists of a letter, two digits, and trailing blanks, and will be printed as the error code in the message which is printed.

Note: This operand must be present.

$DISTERR - Indicate Disastrous Error

The $DISTERR macro instruction is used to indicate that a disastrous error has occurred. The macro instruction causes the following message to be printed out on the $ERR and $LOG consoles:

```
DISASTROUS ERROR - COLD START SYSTEM ASAP
```

Format Description:

```
[symbol]    $DISTERR    [OLAY=YES]
```

OLAY=YES
must be specified if the $DISTERR macro instruction is coded physically within an overlay segment.

$IOERROR - Log Input/Output Error

The $IOERROR macro instruction is used to log an input/output error on the operator's console.

Format Description:

```
[symbol]     $IOERROR     {buffer-addrx | (R1)} [,OLAY=YES]
```

buffer
specifies either a pointer to a HASP buffer or the address of a

buffer which has been associated with a HASP input/output error.

If "buffer" is written as an address, it represents the address of a full word which contains the address of the buffer in error in its three low-order bytes.

If "buffer" is written using register notation (either regular or special register notation), it represents the address of the buffer in error.

If register notation is used, the address must have been loaded into the designated register before the execution of the macro instruction.

OLAY=YES
must be specified if the $IOERROR macro instruction is coded physically within an overlay segment.

$GLOBAL - Define GLOBAL Symbols

The $GLOBAL argument on a COPY instruction causes all HASP GLOBAL Symbols to be defined. This COPY instruction must be given before the COPY $HASPGEN instruction described below to function correctly.

Format Description:

```
COPY    $GLOBAL
```

$HASPGEN - Define HASPGEN Parameters

The $HASPGEN argument on a COPY instruction causes all general HASPGEN parameter values to be defined. This COPY instruction may be placed anywhere in an assembly but must follow the COPY $GLOBAL instruction.

Format Description:

```
COPY    $HASPGEN
```

NULL - Define A Symbol

The NULL macro instruction defines the symbol in the name field, if any, as having the current value of the location counter rounded up, if necessary, to a half-word boundary.

Format Description:

```
[symbol]    NULL
```

$HASPCB - Generate HASP Control Blocks

The $HASPCB macro instruction causes the specified HASP Control Block definitions and, optionally, documentation for those control blocks to be generated.

Format Description:

```
$HASPCB    cb1-code [,cb2-code]...[,cb32-code] [,DOC=YES]
```

cb1-cb24
specifies the control block definitions to be generated as follows:

1. HCT - HASP Communication Table DSECT (or CSECT)
2. PCE - HASP Processor Control Element DSECT
3. BUFFER - HASP Buffer DSECT
4. CMB - HASP Console Message Buffer DSECT

5. SMF - HASP SMF Buffer DSECT
6. DCT - HASP Device Control Table DSECT
7. JQE - HASP Job Queue Element Definitions
8. JIT - HASP Job Information Table Definitions
9. JCT - HASP Job Control Table DSECT
10. IOT - HASP Input/Output Table DSECT
11. TED - HASP Track Extent Data Table DSECT
12. TQE - HASP Timer Queue Element Definitions
13. OTB - HASP Overlay Table DSECT
14. DDT - HASP Data Definition Table DSECT
15. OCR - HASP Output Control Record DSECT
16. PDDB - HASP Peripheral Data Definition Block DSECT
17. PIT - HASP Partition Information Table Definitions
18. JOE - HASP Job Output Element DSECT
19. JOT - HASP Job Output Table DSECT
20. PRC - HASP Print Checkpoint Element Definitions
21. MSA - HASP Message Allocation Control Block DSECT
22. CVT - OS Communication Vector Table DSECT
23. TCB - OS Task Control Block DSECT
24. RB - OS Request Block DSECT
25. JSCB - OS Job Step Control Block DSECT
26. DCB - OS Data Control Block DSECT
27. DEB - OS Data Extent Block DSECT
28. UCB - OS Unit Control Block DSECT
29. RDRWORK - HASP Input Processor PCE Work Area DSECT
30. XEQWORK - HASP Execution Processor PCE Work Area DSECT
31. OUTWORK - HASP Output Processor PCE Work Area DSECT

32. PPPWORK - HASP Print/Punch Processor PCE Work Area DSECT

These arguments can be specified in any combination with the following exceptions:

1. If JCT or IOT is specified, BUFFER must be specified as a prior argument.

2. If JOT is specified, JOE must be specified as a prior argument.

3. If RDRWORK, XEQWORK, OUTWORK, or PPPWORK is specified, PCE must be specified as a prior argument.

4. If OUTWORK is specified, JOE must be specified as a prior argument.

5. If PPPWORK is specified, JCT and BUFFER must be specified as prior arguments.

DOC=YES
specifies that documentation of the control blocks is desired.

$XXC - Variable Core-To-Core Operation

The $XXC macro instruction generates a variable number of core-to-core operations such that there is effectively no restriction on the length of such an operation. The $XXC is especially useful when the length of a core-to-core operation is dependent upon the value of an assembly parameter which may cause the number of operations needed to vary.

Format Description:

```
[symbol]   $XXC   op-code,to-relexp,from-relexp
                  [,length-integer]
```

op
specifies the core-to-core operation as one of the following:

1. NC - AND

2. XC - Exclusive OR

3. MVC - Move

4. MVN - Move Numerics

5. MVZ - Move Zones

6. OC - OR

7. TR - Translate

to
specifies the address of the first field.

This operand may optionally be written as two absolute expressions separated by a comma and enclosed in parentheses. The first expression will be interpreted as a displacement and the second as a base register.

from
specifies the address of the second field.

This operand may optionally be written as two absolute expressions separated by a comma and enclosed in parentheses. The first expression will be interpreted as a displacement and the second as a base register.

length
specifies the total number of bytes in the field.

If this operand is omitted, the length attribute of the first field will be used.

$PATCHSP - Generate Patch Space

The $PATCHSP macro instruction causes a specified number of bytes of patch space to be generated. This patch space will be divided into halfwords and listed in the assembly in such a way that both the assembly location (for REPing and SUPERZAPing) and the base displacement (in the form BDDD) will be printed for each halfword.

Format Description:

```
[symbol]   $PATCHSP   length-number
```

length
specifies the length of the patch space in bytes.

Caution: Local addressability is required for this macro instruction to assemble correctly.

$DLENGTH - Compute Decimal Length

The $DLENGTH macro instruction causes the length of a CSECT (or DSECT) to be computed and that length to be printed in decimal.

Format Description:

```
symbol   $DLENGTH   [HEADER=character]
```

symbol
specifies a name to which the decimal length of the CSECT (or DSECT)

will be assigned. This must be unique for each use of the $DLENGTH macro instruction within a given assembly.

HEADER
specifies a 1-character header which will ensure unique internally-generated symbols. This must be specified differently for each use of the $DLENGTH macro instruction within a given assembly.

If this operand is omitted, the character "L" will be used.

$RTAMDEF - Remote Terminal Access Method Definitions

The $RTAMDEF argument on a COPY instruction causes certain Remote Terminal Access Method symbols to be defined.

Format Description:

```
COPY    $RTAMDEF
```

$FCB - Define 3211 Forms Control Buffer Load

The $FCB macro instruction causes the creation of an overlay CSECT containing a forms control buffer load for the 3211. It may be used in CSECT HASPPRPU to create a new FCB load or, in a stand-alone assembly to be included later in HASP via the Overlay Build process, to replace an existing FCB load.

Format Description:

```
FCBx    $FCB    inch,page,chan-line[,line...]
                [,chan-line][,line...]...,[INDEX=value]
```

x specifies the 1- to 4-character ID by which the FCB load will be referenced by the operator, using the command $TPRTn,C=x and by the programmer using JCL parameters. The ID must be alphanumeric and cannot be "1" or "V".

inch specifies lines per inch. It must be 6 or 8.

page specifies lines per page. It must be 180 or less.

chan specifies carriage channel number. It must be greater than 0, not greater than 12, and followed by a hyphen.

line specifies line number at which the carriage channel punch is to appear. It must not be greater than lines per page.

INDEX specifies a physical print position (1-31) which will be defined as a logical print position 1 for program output. If omitted, a value of 1 is assumed.

$# ADD - Add a Work/Characteristics JOE Pair to the JOT

The $#ADD macro instruction adds a WORK JOE to the specified JOT class queue and conditionally adds the characteristics JOE to the characteristics queue.

Format Description:

```
[symbol]   $#ADD   {WORK=addrx | (RO)}, {CHAR=addrx | (Rl)}
                   [,CLAS=addrx] [,OLAY={YES|NO}]
```

WORK
specifies the address of a prototype work JOE which is to be added to the JOT.

CHAR
specifies the address of a prototype characteristics JOE which is to be merged into the characteristics queue.

CLAS
specifies the address of a byte containing one of the values A-Z, 0-9 representing the class queue to which the work JOE is to be added.

CLAS can be omitted only if WORK=(R0) and the high order byte of (R0) contains the class queue value.

OLAY=YES
must be specified if the $# macro instruction is coded physically within an overlay segment.

NOTE:
The condition code upon exit from the $#ADD macro instruction is set to reflect the status of the request.

cc=0 the service was successfully performed.
cc≠0 the JOT is full - request must be retried later.

$#REM - Remove a Work/Characteristics JOE from the JOT

The $#REM macro instruction is used to remove a work and characteristics JOE pair from the JOT after the output requirement they represent has been satisfied.

Format Description:

```
[symbol]   $#REM  {WORK=addrx | (Rl)} [,OLAY={YES|NO}]
```

WORK
specifies the address of a work JOE which is to be returned to the queue of free JOEs in the JOT. If the related characteristics JOE is not being shared by another work JOE it is also returned to the free queue. The related checkpoint JOE is also freed.

OLAY=YES
must be specified if the $#REM macro instruction is coded physically within an overlay segment.

NOTE:
The related job will be purged from the system if all of its output requirements are removed and its current queue position is $HARDCPY.

$#GET - Search the JOT Class Queues for an output Item which Matches the Requesting Specification

The $#GET macro instruction is used by the Print/Punch Processors to search the JOT for output work. If work is found a checkpoint JOE is assigned to support warm start.

Format Description:

```
[symbol]   $#GET  {WTRID=addrx | (R0)}  {DCT=addrx | (R1)}
           [,HAVE={YES|NO}] [,OLAY={YES|NO}]
```

WTRID
specifies a special SYSOUT writer.

DCT
specifies the address of the JES2 Device Control Table for the requesting processor. The device setup fields in the DCT are used in the process of selecting work. The route code field of the DCT determines which work items are available for selection.

HAVE=NO
specifies that if work is found it is NOT to be assigned to the requester.

OLAY=YES
must be specified if the $#GET macro instruction is coded physically within an overlay segment.

NOTE:
The condition code upon exit from the $#GET macro instruction is set to reflect the status of the request.

cc=0 no work which matches the request is currently available.
cc≠0 work is available and the following registers have been set:

(R0) address of characteristics JOE
(R1) address of the related JOB Queue Element
(R15) address of the selected WORK JOE (the high order byte of R15 contains the class ID).

$#PUT - Return an Unfinished Work Item to the JOT for Later Processing

The $#PUT macro instruction allows a processor to return a work item to the JOT for later processing. Optionally, the status of the work item is maintained for warm start of the system or restart of the work.

Format Description:

```
[symbol]   $#PUT  {WORK=addrx | (R1)} [,PRC=addrx | (R0)]
           [,OLAY={YES|NO}]
```

WORK
specifies the address of a work JOE which is to be returned to the JOT class queues for future selection.

PRC
specifies the address of a checkpoint JOE if the current status of the work item is to be remembered. If PRC= is not specified, the work item is reset to reflect its initial entry status.

OLAY=YES
must be specified if the $#PUT macro instruction is coded physically within an overlay segment.

$#CAN - Cancel All Work Items Not Currently Being Processed for a Specific JOB

The $#CAN macro instruction is used to remove from the JOT all available work items for a job. Any work items thus removed will not be processed by any output processor.

Format Description:

```
[symbol]   $#CAN  {JQE=addrx | (R1)} [,OLAY={YES|NO}]
```

JQE
specifies the address of the JOB Queue Element for which all JOT entries are to be purged.

OLAY=YES
must be specified if the $#CAN macro instruction is coded physically within an overlay segment.

NOTE:
The specified job will be purged from the system if all of its output requirements are removed and its current queue position is $HARDCPY.

APPENDIX B

MULTI-LEAVING

MULTI-LEAVING

"MULTI-LEAVING" is a term which describes a computer-to-computer communication technique developed for use by HASP. In a gross sense, MULTI-LEAVING can be defined as the fully synchronized, pseudo-simultaneous, bi-directional transmission of variable number of data streams between two or more computers utilizing binary synchronous communications facilities. The following section describes, in general terms, the basic structure of MULTI-LEAVING.

MULTI-LEAVING Philosophy

The basic element for MULTI-LEAVING transmission is the character string. One or more character strings are formed from the smallest external element of transmission - the physical record. These physical records are input to MULTI-LEAVING and may be any of the classic record types (card images, printed lines, tape records, etc). For efficiency in transmission, each of these data records is reduced to a series of character strings of two basic types. These two types are: (1) a variable length nonidentical series of characters and, (2) a variable number of identical characters. Because of the high frequency occurrence of blank characters, a special case is made in 2 (noted previously) when the duplicate character is a blank. An eight bit control field, termed a String Control Byte (SCB), precedes each character string to identify the type and length of the string. Thus, a string as in 1 (noted previously) is represented by an SCB followed by the nonduplicate characters. A string of consecutive, duplicate, nonblank characters can be represented by an SCB and a single character (the SCB indicates the duplication count and the character following indicates the character to be duplicated). In the case of an all blank character string, only an SCB is required to indicate both the type and number of blank characters. A data record to be transmitted is, therefore, segmented into the optimum number of character strings (to take full advantage of the identical character compression) by the transmitting program. A special SCB is utilized to indicate the grouping of character strings which compose the original physical record. The receiving program can then reconstruct the original record for processing.

In order to allow multiple physical records of various types to be grouped together in a single transmission block, an additional eight bit control field precedes the group of character strings representing the original physical record. This field, the Record Control Byte (RCB), identifies the general type and function of the physical record (input stream, print stream, data set, etc.). A particular RCB type has been designated to allow the passage of control information between the various systems. Also, to provide for simultaneous transmission of similar functions (i.e., multiple input streams, etc.) a stream identification code is included in the RCB. A second 8-bit control field, the Sub-Record Control Byte (SRCB) is also included immediately following the RCB. This field is utilized to supply additional information concerning the record to the receiving program. For example, in the transmission of data to be printed, the SRCB can be utilized for carriage control information.

MULTI-LEAVING

For actual MULTI-LEAVING transmission, a variable number of records may be combined into a variable block size, as indicated previously (i.e., RCB,SRCB,SCB1,SCB2,...SCBn,RCB,SRCB,SCB1,...etc.). The MULTI-LEAVING design provides for two (or more) computers to exchange transmission blocks, containing multiple data streams as described above, in an interleaved fashion. To allow optimum use of this capability, however, a system must have the capability to control the flow of a particular data stream while continuing normal transmission of all others. This requirement becomes obvious if one considers the case of the simultaneous transmission of two data streams to a system for immediate transcription to physical I/O devices of different speeds (such as two print streams). To provide for the metering of the flow of individual data streams, a Function Control Sequence (FCS) is added to each transmission block. The FCS is a sequence of bits, each of which represent a particular transmission stream. The receiver of several data streams can temporarily stop the transmission of a particular stream by setting the corresponding FCS bit off in the next transmission to the sender of that stream. The stream can subsequently be resumed by setting the bit on.

Finally, for error detection and correction purposes, a Block Control Byte (BCB), is added as the first character of each block transmitted. The BCB, in addition to control information, contains a modulo 16 block sequence count. This count is maintained and verified by both the sending and receiving systems to exercise a positive control over lost or duplicated transmission blocks.

In addition to the normal binary synchronous text control characters (STX, ETB, etc.), MULTI-LEAVING utilizes two of the BSC control characters - ACK0 and NAK. ACK0 is utilized as a "filler" by all systems to maintain communications when data is not available for transmission. NAK is used as the only negative response and indicates that the previous transmission was not successfully received. The following figure indicates the format of a typical MULTI-LEAVING transmission block.

MULTI-LEAVING

bytes		
DLE	-	BSC Leader (SOH if no transparency feature)
STX	-	BSC START-OF-TEXT
BCB	-	Block Control Byte
FCS	-	Function Control Sequence
FCS	-	Function Control Sequence
RCB	-	Record Control Byte for record 1
SRCB	-	Sub-Record Control Byte for record 1
SCB	-	String Control Byte for record 1
DATA	-	Character String
SCB	-	String Control Byte for record 1
DATA	-	Character String
SCB	-	Terminating SCB for record 1
RCB	-	RCB for record 2
SRCB	-	SRCB for record 2
SCB	-	SCB for record 2
DATA	-	Character String
SCB	-	Terminating SCB for record 2
RCB	-	Transmission Block Terminator
DLE	-	BSC Leader - (SYN if no transparency feature)
ETB	-	BSC Ending Sequence

Typical MULTI-LEAVING Transmission

MULTI-LEAVING Control Specification

The following pages indicate the bit-by-bit definitions of the various MULTI-LEAVING control fields and notes concerning their utilization.

String Control Byte (SCB)

Usage: Control field for data character strings

Bit Meanings: O = 0 = End of record (K L J J J J J = 0)
O = 1 = All Other SCBs

K = 0 = Duplicate Character String
L = 0 = Duplicate Character is blank
L = 1 = Duplicate Character is nonblank (and follows SCB)
JJJJJ = Duplication count

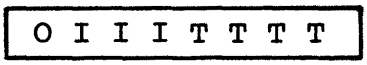
K = 1 = Nonduplicate Character String
LJJJJJ = Character String Length

NOTES:

1. If KLJJJJJ = 0 and O = 1, SCB indicates record is continued in next transmission block.

2. Count units are normally 1 but may be in any other units. The units utilized may be indicated as function control sign-on or dynamically in the SRCB.

Record Control Byte (RCB)



Usage: To identify each record type within a transmission block

Bit Meanings: O = 0 = End of transmission block (I I I T T T T = 0)
O = 1 = All others RCBs

III = Stream identifier - used to identify streams of multiple identical functions (i.e., multiple print streams to a multiple printer terminal, etc.)

III = Control information if TTTT = 0 (control record)
= 000 = Reserved for future expansion
= 001 = Request to initiate a function transmission (Prototype RCB for function in SRCB)
= 010 = Permission to initiate a function transmission (RCB for function contained in SRCB).
= 011 = Reserved
= 100 = Reserved
= 101 = Available for local modification
= 110 = Available for local modification
= 111 = General Control Record (type indicated in SRCB)

TTTT = Record type identifier
= 0000 = Control record
= 0001 = Operator message display request
= 0010 = Operator command
= 0011 = Normal input record
= 0100 = Print record
= 0101 = Punch record
= 0110 = Data set record
= 0111 = Terminal message routing request
= 1000 - 1100 = Reserved for future expansion
= 1101 - 1111 = Available for local modifications

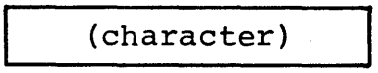
Sub-Record Control Byte (SRCB)

Usage: To provide supplemental information about a record

Bit Meanings: O = 1 (Must always be on)

SSSSSSS = Additional record information - actual content is dependent on record type. Several examples are listed below:

SRCB for General Control Record



Usage: To identify the type of generalized control record

Bit Meanings: character = A = Initial terminal SIGN-ON
= B = Final terminal SIGN-OFF
= C = Print initialization record
= D = Punch initialization record
= E = Input initialization record
= F = Data set transmission initialization
= G = System configuration status
= H = Diagnostic control record
= I - R = Reserved
= S - Z = Available for local modification

SRCB for Print Records

MULTI-LEAVING

Usage: To provide carriage control information for print records

Bit Meanings: O = 1 (Must always be on)

M = 0 = Normal carriage control
= 1 = Reserved for future use

CCCCCC = Carriage control information
= 1000NN = Space immediately NN spaces
= 11NNNN = Skip immediately to channel NNNN
= 0000NN = Space NN lines after print
= 001100 = Load 3211 FCB image
= 01NNNN = Skip to channel NNNN after print
= 000000 = Suppress space

SRCB for Punch Records

Usage: To provide additional information for punch records

Bit Meanings: O = 1 (Must always be on)

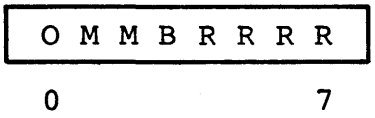
SS = Punch stacker select information

B = 0 = Normal EBCDIC card image
= 1 = Column Binary card image

M = 00 = SCB count units = 1
= 01 = SCB count units = 2
= 10 = SCB count units = 4
= 11 = Reserved

RR = Reserved for future expansion

SRCB for Input Record



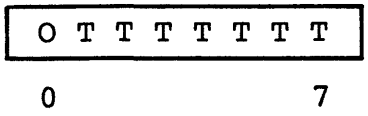
Usage: To provide additional information for input records

Bit Meanings: O = 1 (Must always be on)

M = 00 = SCB count units = 1
= 01 = SCB count units = 2
= 10 = SCB count units = 4
= 11 = Reserved

RRRR = Reserved

SRCB for Terminal Message Routing Record

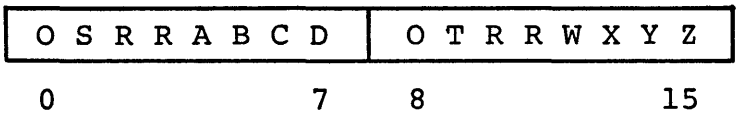


Usage: To indicate the destination of a terminal message

Bit Meanings: O = 1 (Must always be on)

TTTTTTT = Remote system number (1 ≤ T ≤ 127)
TTTTTTT = 0 = Broadcast to all remote systems)

Function Control Sequence (FCS)



Usage: To control the flow of individual function streams

Bit Meanings: O = 1 (Must always be on)

S = 1 = Suspend all stream transmission (WAIT-A-BIT)
= 0 = Normal state

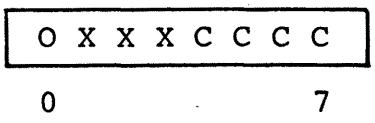
T = Remote console stream identifier

R = Reserved for future expansion

ABCD...WXYZ = Various function stream identifiers
(oriented only to recipient)
- Normal print (or input) = A,B,C,...
- Normal punch streams = Z,Y,X,...

NOTE - a bit on = continue function transmission
- a bit off = suspend function transmission

Block Control Byte (BCB)



Usage: Transmission block status and sequence count

Bit Meanings: O = 1 (Must always be on)

CCCC = Modulo 16 block sequence count

XXX = Control information as follows --
= 000 = Normal Block
= 001 = Bypass sequence count validation
= 010 = Reset expected block sequence count to CCCC
= 011 = Reserved
= 100 = Reserved
= 101 = Available for user modification
= 110 = Available for user modification
= 111 = Reserved for future expansion

MULTI-LEAVING In BSC/RJE

The previous sections have grossly outlined the specifications of a comprehensive, MULTI-LEAVING communications system. While the HASP support for programmable BSC work stations is completely consistent with the MULTI-LEAVING design, it does not utilize certain of the features provided in MULTI-LEAVING. These feature limitations include:

1. The transmission of record types other than print, punch, input, console and control is not supported.

2. The only general control record types utilized are the terminal SIGN-ON.

3. Only SCB count units of 1 are utilized.

4. No support is included for column binary cards.

GLOSSARY

HASP

TERMS AND ABBREVIATIONS

HASP GLOSSARY

This glossary defines HASP terms and abbreviations as they are used in this manual. The VS2 abbreviations used in this manual are also listed. For a further explanation of the VS2 abbreviations see VS2 documentation or the IBM Data Processing Glossary (GC20-1699).

APG: Automatic priority group

asynchronous: Without regular time relationship; unexpected or unpredictable with respect to execution of a program's instructions

BRDR: Background Reader of TSO

BUFFER: HASP buffer

CCW: Channel Command Word

CEA: Channel end appendage

CKPT: Checkpoint

CMB: HASP Console Message Buffer

cold start: HASP initialization option; any jobs HASP has from a previous IPL are ignored, and any SPOOL volume space used by those jobs is released.

COMWORK: HASP Command Processor PCE work area

CR: Carriage return

CVT: Communications Vector Table

DASD: Direct access storage device

DAT: Dynamic address translation

DCB: Data Control Block

DCT: HASP Device Control Table

DEB: Data Extent Block

DDT: HASP Data Definition Table

DOM: Delete operator message

ECB: Event Control Block

EOF: End-of-file

EWF: HASP Event Wait Field

HASP GLOSSARY

Execution Batch Scheduling: A job batching feature within HASP activated by use of the &XBATCHC HASPGEN parameter

FCB: Forms Control Buffer

FIFO: First in, first out

HASPGEN: Generation of a HASP System

HCT: HASP Communication Table

HDB: HASP Data Block

Internal Reader: HASP Internal Reader; a pseudo-device used to enter jobs directly into HASP job stream from any other nonswappable program operating in the system

IOB: Input/Output Block

IOS: Input/Output supervisor

IOT: HASP Input/Output Table

IPL: Initial program load

JCL: Job control language

JCT: HASP Job Control Table

JFCB: Job File Control Block

JIT: HASP Job Information Table

JMR: Job Management Record

JOE: HASP Job Output Element

JOT: HASP Job Output Table

JQE: HASP Job Queue Element

JSCB: Job Step Control Block

LIFO: Last in, first out

LPA: Link pack area

LPAQ: Link pack area queue

LSQA: Local system queue area

MSA: HASP message SPOOLING allocation control block

MULTI-LEAVING: A computer-to-computer communication technique; the fully synchronized, pseudo-simultaneous, bi-directional transmission of a variable number of data streams between two or more computers utilizing binary synchronous communications facilities

NIP: Nucleus initialization procedure

OCR: HASP Output Control Record

OTB: HASP Overlay Table

OUTWORK: HASP Output Processor PCE work area

PCE: HASP Processor Control Element

PDDB: HASP Peripheral Data Definition Block

PGFX: Page fix

PIT: HASP Partition Information Table

PPPWORK: HASP Print/Punch Processor PCE work area

PRC: HASP print checkpoint element

pseudo device: A physically non-existant device represented by a Unit Control Block in the Operating System Nucleus which is used by HASP to interface with the sequential access methods for SPOOLed data sets. This UCB is generated at SYSGEN time by a unique specification on the IODEVICE macro-instruction.

RB: Request Block

RDRWORK: HASP Input Processo.

RDW: Record descriptor word

REP: HASP replacement card

REQ: HASP initialization option that will cause HASP to wait for a $S command before beginning job processing

RJE: HASP Remote Job Entry

RMTGEN: A procedure for generating HASP remote terminal programs, the load module which when executed causes HASP remote terminal programs to be generated.

RPS: Rotational Position Sensing

RQE: Request Queue Element

RTAM: HASP Remote Terminal Access Method

RTP: Remote Terminal Processor

SIO: Start Input/Output

SMB: System Message Block

SMF: HASP System Management Facilities

SPOOL: A DASD volume(s), or portion thereof, used by HASP for storing a job's SYSIN and SYSOUT data and CHECKPOINT data

SQA: System Queue Area

SWA: Scheduler Work Area

synchronous: Occurring with a regular or predictable time relationship

TCB: Task Control Block

TED: Track Extent Data

TIOT: Task Input/Output Table

TSO: Time sharing option

TQE: HASP Timer Queue Element

UCB: Unit Control Block

UCS: Universal Character Set

UCSB: Universal Character Set Buffer

UR: Unit record

warm start: HASP initialization option that causes HASP to continue to process all jobs in the system when the system crashed or HASP was stopped

WTL: Write to log

WTO: Write to Operator

WTOR: Write to Operator with Reply

XEQWORK: HASP Execution Processor PCE work area

INDEX

IBM Technical Newsletter

This Newsletter No. SN27-1555
Date Sept. 15, 1976

Base Publication No. GY27-7255-0
File No. S370-36

Previous Newsletters SN25-0122

OS/VS 2
HASP II Version 4 Logic

This technical newsletter, a part of Version 4.1 of HASP II and of Release 1.7 of OS/VS2 SVS, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent HASP versions and SVS releases unless specifically altered. Pages to be inserted and removed are:

Cover, edition notice
1–4
1-5, 1-6
2-13–2-18
3-13.1 (3.13.1 added)
3-29, 3-30
3-33–3-36
3-41–3-44
3-75
3-117, 3.118.1 (3-118.1 added)
4-3–4-26
5-15–5-17
5-21, 5-22
5-25, 5-26
5-31, 5-32
5-39, 5-40
5-45–5-47 (5-46.1 added)
5-59–5-65
5-69, 5-70
5-75, 5-76.1 (5-76.1 added)
5-79–5-81
5-85–5-92 (5-90.1 added)
5-95–5-101 (5-100.1, 5-100.2 added)
5-105–5-110.1 (5-110.1 added)
5-117, 5-118.1 (5-118.1 added)
5-129–5-132
7-25, 7-26
IND-1–IND-7

Summary of Amendments

Support for the IBM 3800 Printing Subsystem has been added. Also, miscellaneous technical and editorial changes have been made throughout the book.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150

Printed in U.S.A.

IBM®

Technical Newsletter

This Newsletter No. SN25-0122
Date May 28, 1975

Base Publication No. GY27-7255-0

Previous Newsletters None

OS/VS 2
HASP II Version 4 Logic

This Technical Newsletter, a part of release 1 of OS/VS2, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent OS/VS2 releases unless specifically altered. Pages to be removed/inserted are:

Cover	5-61, 5-62
5, 6	6-3, 6-4
2-1, 2-2	7-1—7-2
3-43, 3-44	7-39—7-42
3-63, 3-64	7-55—7-68
3-111, 3-112	Ind-3,—Ind-6
5-49, 5-50	

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

Summary of Amendments

Miscellaneous editorial and technical changes have been made throughout the text and a section on MULTI-LEAVING has been added.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Department 63T
Neighborhood Road, Kingston, New York 12401

Printed in U.S.A.

OS/VS2
HASP II Version 4 Logic
GY27-7255-0

Reader's Comment Form

Your comments about this publication will help us to improve it for you. Comment in the space below, giving specific page and paragraph references whenever possible. All comments become the property of IBM.

Please do not use this form to ask technical questions about IBM systems and programs or to request copies of publications. Rather, direct such questions or requests to your local IBM representative.

If you would like a reply, please provide your name and address (including ZIP code).

Fold on two lines, staple, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

GY27-7255-0

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold and Staple

First Class Permit
Number 6090
San Jose, California

Business Reply Mail

No postage necessary if mailed in the U.S.A.

Postage will be paid by:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Staple

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 HASP II Version 4 Logic (File No. S370-36) Printed in U.S.A. GY27-7255-0

GY27-7255-0

OS/VS2 HASP II Version 4 Logic (File No. S370-36) Printed in U.S.A. GY27-7255-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)