IBM

Field Engineering Education
Supplementary Course Material

OS/HASP

Volume 1

PREFACE

This document is intended for the use of IBM FE Programming System Representatives enrolled in course 10191.

PRELIMINARY EDITION (May 1971)
This publication has been printed in a preliminary format so that it would be available to the intended users in time for training on this course. This preliminary manual may contain typographical errors that would normally be corrected before publication. This edition is not eligible for suggestion awards, however, your comments will be appreciated.

Issued to: _____

Branch Office: _____ No: _____

Address: _____

_____

If this manual is mislaid, please return it to the above address.

Address any comments concerning the contents of this publication to: IBM, Field Engineering Education Media Development Center, Dept 927, Rochester, Minnesota 55901.

MAGNETIC TAPE KEY

BASIC

This volume contains two files as described below.

File 1 - Assembled object decks and JCL necessary to perform a HASPGEN (refer to Section 10 of this manual for information concerning use of this tape).
Records - 417, Characters/block - 80, Records/block - 1, Blocks/file - 417.

File 2 - Source decks for HASP-II, Version 3.0.
Records - 50,343, Characters/block - 1600, Records/block - 20, Blocks/file - 2518.

*Optional

System/3 users only - 140 96-column cards. This deck is a "starter system" for the HASP MULTI-LEAVING Remote Job Entry support.

*Optional material will be forwarded only when specifically requested.

TABLE OF CONTENTS

Introduction

The information contained in Volume 1 and 2 of the HASP System Supplementary Course Material was originally distributed as a one-volume document.

Volume 1 contains pages 1 through 590, Section 1 through 8. Volume 2 contains pages 1 through 594, Section 9 through 12.

A Contents has been included in each volume for your convenience.

1.0 INTRODUCTION

The HASP SYSTEM operates as a compatible extension to the MFT or MVT options of the Operating System for System/360 and System/370 to provide specialized supplementary support in the areas of job management, data management, and task management.

HASP appears as a transparent "front-end" processor to OS to, via the SPOOLing functions normally associated with OS input readers and output writers, act as an automatic scheduler and operator of OS. Because of this relationship between HASP and the Operating System, various other functional, performance and operational benefits can be included in HASP.

The use of HASP offers an installation the following advantages:

- IMPROVED PERFORMANCE - In many cases, because of the singular, specialized use of resources by HASP, system performance may be improved. Any improvement is dependent upon the configuration and job mix and can only be determined by actual measurement. (See Section 2 of this manual for additional details.)

- IMPROVED OPERATIONAL PROCEDURES - HASP acts as an automatic interface between the operator and OS, to perform various OS control functions previously done directly by the operator. Readers, Writers and Initiators in OS are started and scheduled automatically by HASP. Also, many additional operator commands for controlling job flow and device operation are provided by HASP. (See Section 11 of this manual for additional details.)

- INCREASED SYSTEM FUNCTION - The use of HASP provides certain functions which are not otherwise available. These include dynamic task ordering based upon CPU - I/O characteristics (see Section 2 for additional details); the inclusion of relevant console messages in each job's output (see Section 7 for additional details); the capability of any job to introduce another job into the HASP queue via an internal reader (see Section 12.10 for additional details); an execution batching facility to pass jobs directly to a processing program such as a one-step monitor (see Section 12.13 for additional details); many additional operational control functions (see Section 11 for additional details); a priority aging technique (see Section 4.22 for additional details); a pre-execution volume fetch facility (see Section 11 for additional details); and various other functional enhancements.

- RESOURCE REDUCTION - Because of the dynamic direct-access allocation techniques utilized by HASP, installations may, in general, reduce the number of direct-access volumes required

for SPOOLing functions as compared with a non-HASP SYSTEM. The size of the OS SYS1.SYSJOBQE data set may also be reduced since all job queueing is performed by HASP.

Certain installations may actually reduce system main storage requirements (increase problem program space available) by adding HASP to their system because of the OS functions replaced by HASP. In any case, the space required for the HASP partition or region will be at least partially compensated for by the elimination of duplicate functions.

- LOW-ENTRY, HIGH-PERFORMANCE REMOTE JOB ENTRY - For a nominal increase in the size of HASP, an installation can utilize the HASP RJE support for a wide variety of workstation devices. Support for Binary-Synchronous, CPU workstations employs an advanced technique called MULTI-LEAVING which provides for simultaneous operation of all devices on a remote workstation. A subset of the HASP operator command language is provided to all remote sites. Workstation programs are supplied for all supported CPU workstations. (See Section 12.11 for additional details.)

- TRANSPARENT OPERATIONS - HASP is, in general, transparent to both the Operating System and to user programs. Although a special SYSGEN is required, no actual modifications to OS are required to utilize HASP. Thus, the same generation of OS may be interchangeably used with or without HASP. Because of this transparency, HASP is generally independent of the OS release level or options selected and can be used as a stable base for local modifications to customize for local operational requirements.

  Most standard jobs which operate under OS can be run with absolutely no change in a HASP environment. Most installations can, therefore, implement HASP with little or no changes to current user programs.

2.0 GENERAL DESCRIPTION

HASP is a specialized program which operates in the same CPU with OS/360 to perform the peripheral functions associated with batch job processing.

HASP is loaded as a normal OS/360 program and upon gaining control enters the supervisor mode via a special SVC routine. Control of all on-line unit record devices is assumed, the designated intermediate storage direct-access device(s) are initialized and job processing begins. The basic interface between HASP and OS/360 is through the Input-Output Supervisor (IOS). The entry point of IOS is modified so that Input-Output requests to unit record devices are diverted to HASP rather than being physically executed by IOS. Jobs which have been previously read from physical input devices by HASP can now be passed to OS by simulating a successful completion of the intercepted I/O request. In a similar manner, print and punch output from jobs being processed by OS/360 can be intercepted and queued on intermediate storage for later transcription to unit record devices.

HASP has four major processing stages which account for its four major external functions. These are:

1. INPUT STAGE - This stage reads jobs simultaneously from an essentially unlimited number of various types of on-line card readers, tapes 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 upon priority and class from the queue established by the Input stage and passes those jobs to OS/360 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/360. At the completion of a job, it is placed in a queue to await processing by the next stage.

3. PRINT STAGE - The 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.

4. PUNCH STAGE - This stage 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 the these processes are controlled by re-entrable code so that 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.

Following are 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 space for intermediate storage on a record basis, within definable track groups, for jobs. The use of this technique offers the following advantages:

  1. Disk-arm motion and interference is minimized by dynamically allocating space based upon 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 selector 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 an 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 memory 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 memory which is allocated as required. This technique allows not only multiple data sets of a job, but multiple jobs to share this area, thereby insuring optimum use of storage.

- EXECUTION TASK MONITOR

A significant item contributing to system performance is the correct ordering of dispatching priorities of jobs in relation to their CPU-I/O utilization ratios. It is obviously straightforward to manually set the dispatching priorities of two jobs, one of which is completely I/O-bound and the other completely CPU-bound. It becomes more difficult to determine relative priorities of multiple jobs with varying degrees of CPU-I/O ratios and impossible to determine priorites for multiple jobs which constantly change from CPU to I/O bound or vice versa.

HASP provides a feature which, at frequent intervals, examines each eligible job and dynamically re-orders the OS dispatching chain based upon the measured CPU-I/O characteristics of the jobs during the previous interval. This capability relieves an installation of the responsibility of attempting to assign job dispatching priorities while insuring the optimum ordering of jobs being processed by the Operating System.

(The remainder of this page intentionally left blank.)

3.0 HASP STRUCTURE

The primary goal in the design of any execution support system such as HASP must be the efficient manipulation of the various resources required for processing. The first design steps must then include the determination of what resources will be required and the careful application of sound programming design techniques to achieve an efficient and consistent solution to the allocation of these resources.

A study would reveal that HASP requires the following resources:

1. Main Storage
2. Direct-Access Space
3. Input/Output Units
4. Central Processing Unit Time
5. Input/Output Channel and Unit Time
6. Programs
7. Jobs
8. Interval Timer

Since these resources are essentially the basic facilities provided by the Operating System, it would at first seem that these facilities would be sufficient to meet the requirements of HASP. Further studies show, however, that the philosophies of the Operating System's services are not always consistent with the design requirements of a system such as HASP.

For instance, the main storage services provided by the Operating System are very flexible and comprehensive but fail to meet the requirements of HASP in the following areas:

- As requests for main storage are serviced, memory becomes fragmented in such a way that eventually a request for storage cannot be serviced for lack of contiguous memory even though the total amount of storage available far exceeds the requested quantity.

- As the amount of available storage decreases, the requestor becomes more susceptible to being placed in an OS WAIT state or being ABENDed. These conditions are both intolerable to HASP.

- The primary use of main storage in HASP is for buffering space for input/output purposes. These input/output purposes require that an Input/Output Block be associated with each segment of main storage which the Operating System Main Storage Supervisor, only naturally, does not provide. This means that HASP would have to construct such a block for each main storage segment it required.

In a similar fashion the Direct-Access Device Space Manager (DADSM) provides flexible and comprehensive services for normal job processing requirements but fails to meet the requirements of HASP in the following areas:

- Because of the data set concept employed by DADSM, the "hashing" or "fragmentation" problem described above also impacts the allocation of direct-access space.

- The data set concept complicates the simultaneous allocation of storage across many volumes (for selector channel overlap).

- The DADSM limit of extents per volume tends to cause volume switching, and the associated time delays are intolerable to HASP.

- DADSM consists of non-resident routines which must be loaded for each direct-access space allocation service. Because of the frequent allocation requirements, the associated overhead involved in the loading of these routines would degrade the performance of HASP to a certain extent.

Since the unit-record input/output units which the scheduler allocates to the jobs being processed in other partitions must be available for use by HASP, HASP must be responsible for the allocation of its own input/output units.

The Operating System Task Supervisor is responsible for the allocation of Central Processing Unit (CPU) time to all tasks in the system. The different functions of HASP (reading cards, printing, punching, etc.) could be defined as individual OS tasks except for the following considerations:

- Defining each function as a separate task would prohibit HASP from being used with anything other than a variable-task system.

- Inter-task communication and synchronization is many times more complex than intra-task communication and synchronization.

The Operating System Input/Output Supervisor is responsible for the allocation of all input/output channel and unit time. It completely meets all requirements and is used by HASP for all input/output scheduling.

The Operating System Interval Timer Supervisor provides complete interval timer management services but limits these services to one user per task. Since HASP has many functions which have simultaneous interval timer requirements, an interface must be provided which will grant unlimited access to the OS Interval Timer Supervisor.

The following sections describe, in detail, the allocation techniques and algorithms used in HASP to provide the allocation of the resources listed above.

3.1 ALLOCATION OF MAIN STORAGE

The main storage requirements of HASP are as follows:

- Storage space for buffering card images and print lines between intermediate direct-access storage devices and unit-record devices.
- Storage space for normally non-resident control tables during times when they are resident in main storage.
- Storage for console messages which have been queued for output to or input from one or more operator consoles.
- Storage for elements which reflect the status of all jobs which are queued for any stage of processing by HASP.
- Storage space for non-resident processing routines (overlays) during times when they are in main storage.

The HASP Buffer Pool

The first two requirements for main storage are provided for by the HASP Buffer Pool, a group of buffers with the following basic format:

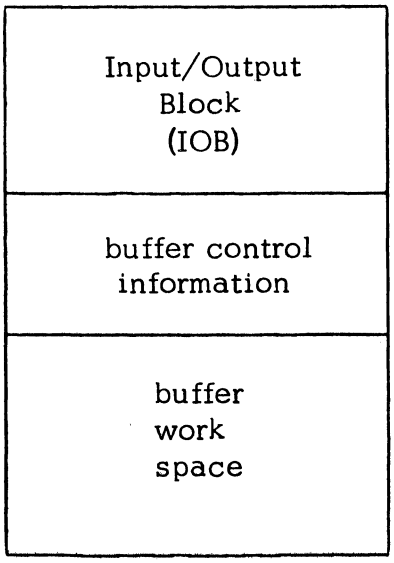


Figure 3.1.1 - The HASP Buffer

Since the use of this buffer always involves some input/output activity, a standard Operating System Input/Output Block (IOB) is provided with each buffer for the purpose of being used to initiate this input/output activity.

The "buffer control information" area is an extension of the IOB used by HASP for input/output synchronization.

The "buffer work space" is a fixed-length (set by HASPGEN) area into which data is read and/or out of which data is written.

In addition to a fixed number of buffers (set in accordance with region or partition size), the buffer pool contains a one-word control field called the Buffer Pool Control Block which contains the address of the first available buffer in the buffer pool. Each available buffer contains the address

of the next available buffer with the last available buffer containing a zero address. If no buffers are available, the Buffer Pool Control Block contains zero.

The above technique is called "chaining," the buffers are said to be "chained," and the field containing the address of the next element in the chain is referred to as the "chain field." Chaining is used throughout HASP for the maintenance of resources.

To obtain an available buffer from the buffer pool, the Buffer Pool Control Block is tested for an available buffer. If one exists it is removed from the available chain by moving its chain address into the pool control block.

To release a buffer to the available chain, the contents of the pool control block are moved to the chain field of the buffer, and the address of the buffer is placed in the pool control block.

The Console Message Buffer Pool

The third requirement for main storage is provided for by the Console Message Buffer Pool. This buffer pool is organized similarly to the HASP Buffer Pool except for the format of the buffers which is as follows:

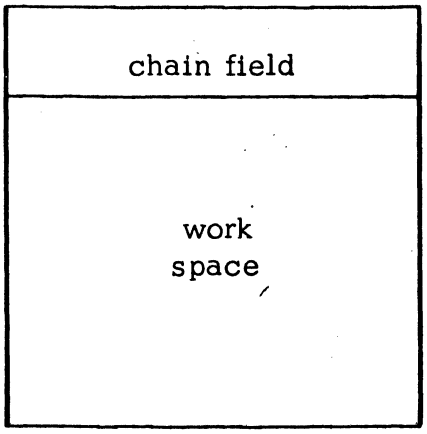


Figure 3.1.2 — The Console Message Buffer

Since IOB's are provided for each console, it is not necessary to provide such a control block with each buffer.

The length of the work space is consistent with the maximum length of a console message.

Buffers in this buffer pool are obtained and released by exactly the same procedure used in the HASP Buffer Pool.

The HASP Job Queue

The fourth requirement is provided for by the HASP Job Queue. For more information about this facility see section 3.6.

The HASP Overlay Area Pool

The HASP Overlay Area is similar to the HASP Buffer in format; however, the size of the "work space" is set to accommodate the largest non-resident HASP control-section (CSECT). Although the fixed number of overlay areas (set by HASPGEN) are chained together, control fields indicate the area status and contents for the purpose of sharing areas containing the same CSECT or for selecting an area to overlay with a new CSECT.

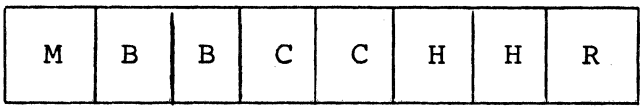
3.2 ALLOCATION OF DIRECT-ACCESS SPACE

The direct-access allocation technique employed by HASP must meet the following requirements:

- It must use a minimum of CPU time.
- It must not use an excessive amount of main storage.
- It must not be susceptible to the "hashing" or "fragmentation" problem.
- It must be capable of allocating for any direct-access device which is supported by Operating System/360.
- It must be device transparent to the user.
- It must be consistent with the checkpoint/restart technique used by HASP.

The HASP Track Address

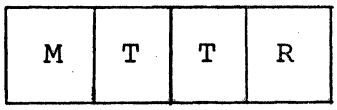
The standard Operating System track address is defined to be an eight-byte field with the following format:



where:
M = Module
BB = Bin
CC = Cylinder
HH = Head
R = Record

Figure 3.2.1 - The Operating System Track Address Format

For the purpose of HASP, this track address can be reduced to a four-byte field with the following format:



where:
M = Module (DEB extent number)
TT = True Track Number
R = Record

Figure 3.2.2 - The HASP Track Address Format

The reduction in the length of the track address permits it to be kept in a single word of storage or in a general purpose register simplifying the handling of the track address.

The HASP Master Track Group Map

The HASP Master Track Group Map is a table which represents the sum total of all track groups or logical cylinders available on all HASP direct-access SPOOL volumes. (A track group contains one or more tracks which are considered a single resource.) Each bit in the HASP Master Track Group Map represents a single track group on one direct-access volume. If the bit is one, it indicates that the corresponding logical cylinder is available for allocation; if the bit is zero, the logical cylinder is not available to HASP or has already been allocated by HASP.

The HASP Job Track Group Map

The HASP Job Track Group Map is identical to the HASP Master Track Group Map except that one word has been added to the front to save the last track address which was allocated to the particular job with which the map is associated. The bits in the Job Track Group Map represent the same track groups as the bits in the Master Track Group Map except that a one bit indicates that the respective track group has been allocated to the associated job and a zero indicates that the group has not been allocated to the job.

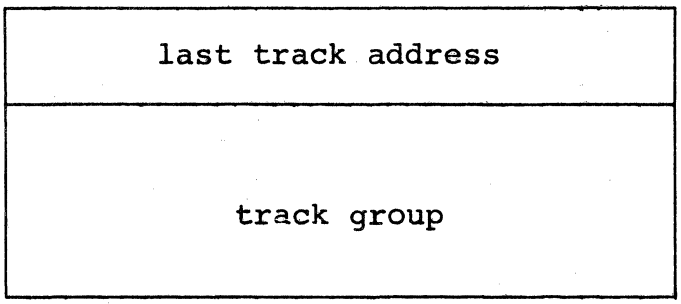


Figure 3.2.3 - The HASP Job Track Group

Two Job Track Group Maps are associated with each job. One represents the track groups used to contain the input data (SYSIN), and the other represents the groups used to contain the output data (SYSPRINT and SYSPUNCH).

Direct-Access Space Allocation Procedures

When the direct-access space allocation subroutine is entered, it first examines the first four bytes of the appropriate Job Track Group Map to determine if a new track group is required. A new group is required whenever no tracks have been allocated to this job (the last track address is zero) or if all of the tracks in the last group allocated have been used.

If a new track group is not required, the record or head field of the last track address is incremented to provide a new track address.

If a new track group must be allocated, the Master Track Group Map is scanned for an available group. When the next group to be allocated is determined, the appropriate bit in the Master Track Group Map is set to zero, and the corresponding bit in the Job Track Group Map is set to one. A track address is then constructed to represent the first track in the new group, and this track address is saved in the first four bytes of the Job Track Group Map.

When any direct-access input/output operation is initiated by HASP, the HASP I/O interface saves the cylinder which was referenced by module. When a new track group must be allocated, the allocation routine first tries to allocate a group corresponding to the last cylinder referenced on each module. If these groups are not available, the routine attempts to allocate within one cylinder of the last references. If track groups within these cylinders are not available, the routine tries to allocate a group within two cylinders, and so on, until the entire track group map has been examined.

Direct-Access Space De-Allocation Procedure

To de-allocate the direct-access space allocated to a particular function, it is necessary only to "OR" the track group map portion of the Job Track Group Map associated with the particular function into the Master Track Group Map. This will reset to one all bits in the Master Track Group Map which correspond to the track groups which have been allocated to the particular function.

3.3 ALLOCATION OF INPUT/OUTPUT UNITS

The HASP Device Control Table (DCT) is used by HASP to allocate all input/output units. It has the following basic format:

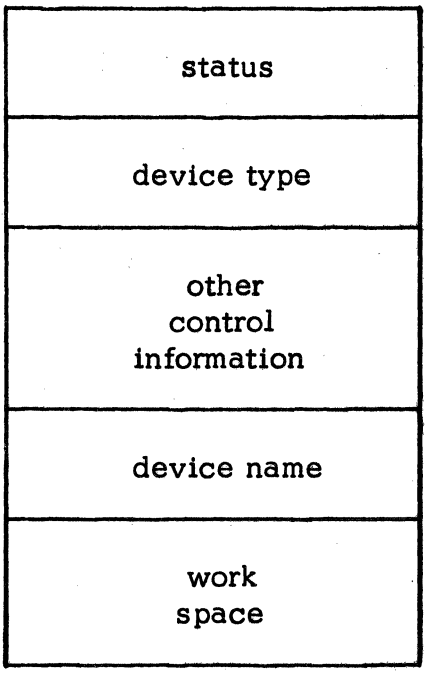


Figure 3.3.1 — The HASP Device Control Table (DCT)

The "status" field is used to indicate whether the device is available and whether it is in use.

The "device type" field specifies whether this DCT represents a card reader, printer, punch, or other type of I/O device.

The "other control information" field contains such information as the Data Control Block (DCB) address, the chain address, indications of operator commands, and other fields for synchronization purposes.

The "device name" field contains an eight-byte EBCDIC device name (such as READER1) which is primarily used for console messages.

The "work space" is a device dependent area used by some devices for extended control of the device.

All DCT's are chained together for allocation purposes. They are initialized by the HASP initialization phase if the associated devices are attached to the system.

Input/Output Device allocation consists of "running" the DCT chain and looking for a DCT of the specified type which is available and which has not been allocated. If one is found, the "in use" bit is set to one to indicate that the device has been allocated.

De-allocation consists of setting the "in use" bit to zero.

The Device Control Table is also used as a parameter list whenever Input/Output activity is initiated through the HASP I/O interface.

3.4 ALLOCATION OF CENTRAL PROCESSING UNIT TIME

The Operating System controls the allocation of Central Processing Unit (CPU) time to different tasks through the means of a Task Control Block (TCB) chain. In a similar fashion, HASP controls the allocation of CPU time to the different functions within HASP through the means of a Processor Control Element (PCE) chain. The basic format of the Processor Control Element is as follows:

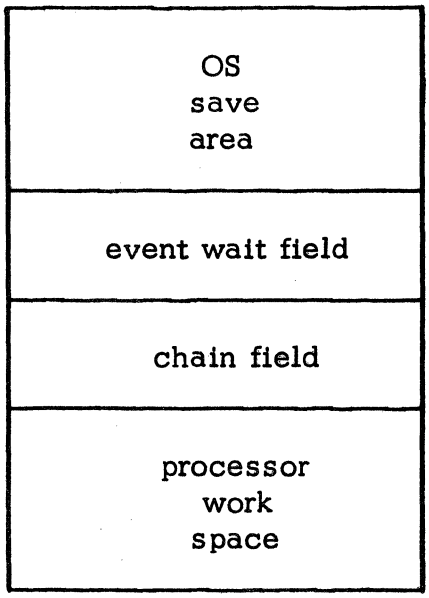


Figure 3.4.1 — HASP Processor Control Element (PCE)

Whenever a particular function is being processed, general purpose register 13 always contains the address of the Processor Control Element which is allocating the time to that function. For this reason the first eighteen words of the PCE are a standard OS register save area.

The "event wait field" is a two-byte field which describes the dispatchability of the function under the control of this PCE. If this field is zero, the function is dispatchable. If this field is non-zero, the function is not dispatchable and the bit which is one specifies upon what event the function is "waiting".

The "chain field" contains the address of the next PCE in the PCE chain.

The "processor work space" is a variable length area which is used by the program processing the function as a scratch area.

HASP searches the PCE chain looking for a PCE which is dispatchable. When a dispatchable PCE is located, the general purpose registers are loaded from the PCE/OS save area and control is passed to the location specified in register 15.

When control is returned to the dispatching program, the general purpose registers are saved in the PCE and the search for dispatchable PCEs continues. If a notable event occurred since the last PCE dispatch such as the freeing of a common resource or the "posting" of a specific event, the search starts at the beginning of the PCE chain; otherwise, it starts with the PCE following the last dispatched. The program returning control to the dispatching program must set the return address in register 15 before returning.

When no PCEs are found to be dispatchable, the HASP task enters an OS WAIT state to allow the Operating System to allocate CPU time to other tasks.

3.5 ALLOCATION OF PROGRAMS

The programs of which HASP is composed can be divided into the following classifications:

- The Dispatcher
- Processors
- Control Service Programs
- Miscellaneous Programs

The Dispatcher is the dispatching program described in Section 3.4. Its function is to distribute CPU time among the various processors described below.

Processors are programs which control the execution of various HASP functions such as reading cards, printing, punching, etc. With each processor is always associated at least one Processor Control Element which causes the dispatcher to give control to the processor and allows the processor to synchronize with various HASP events. The PCE work space also permits the processors to be written re-enterably such that by defining more than one PCE for a given processor, the processor can control an essentially unlimited number of functions simultaneously. For instance, by defining ten PCEs for the Print Processor, up to ten printers can be serviced simultaneously utilizing and requiring only one copy of the processing program.

The Control Service Programs are subroutines used by the processors in accomplishing their functions. By using the PCE/OS save area, the control service programs can maintain the re-enterability of the processors.

Miscellaneous Programs are those special purpose programs which do not fall into any of the other three categories, such as the HASP Initialization Program. They are executed only once and need not be considered in the normal HASP job flow.

3.6 ALLOCATION OF JOBS

HASP maintains its job pointers in the HASP Job Queue, a table of elements with the following basic format:

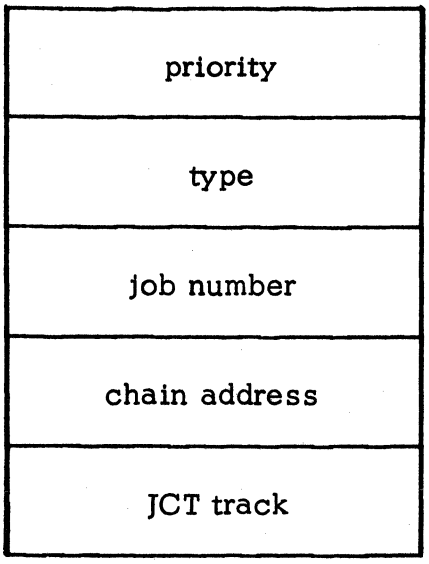


Figure 3.6.1 — The HASP Job Queue Element

The "priority" represents the dynamic priority of the job within the HASP system.

The "type" represents the function for which the element is queued or the function in which the job is currently being processed.

The "job number" is the number sequentially assigned to each job by HASP as it enters the system.

The "chain address" is the address of the next element in the chain.

The "JCT track" is the track address of the HASP Job Control Table described below.

Two chains are maintained in the Job Queue. The first chain represents those jobs which are currently awaiting processing or being processed. Elements in this chain are chained in the order of their priority. The second chain represents the inactive or unused queue elements.

To add a job to the job queue, a queue element is obtained from the inactive chain, initialized with the information shown in figure 3.6.1, and inserted into the active chain according to its priority.

To obtain a job from the job queue, the active chain is searched for an element of the specified type. When found, the "type" field is modified to reflect the fact that the job is now being processed.

To return a job to the job queue, the element is moved from the active chain to the inactive chain. Since the priority is of no concern here, the element is placed at the head of the chain.

The HASP Job Control Table (JCT)

The HASP Job Control Table contains all of the information necessary to process the associated job in the following basic format:

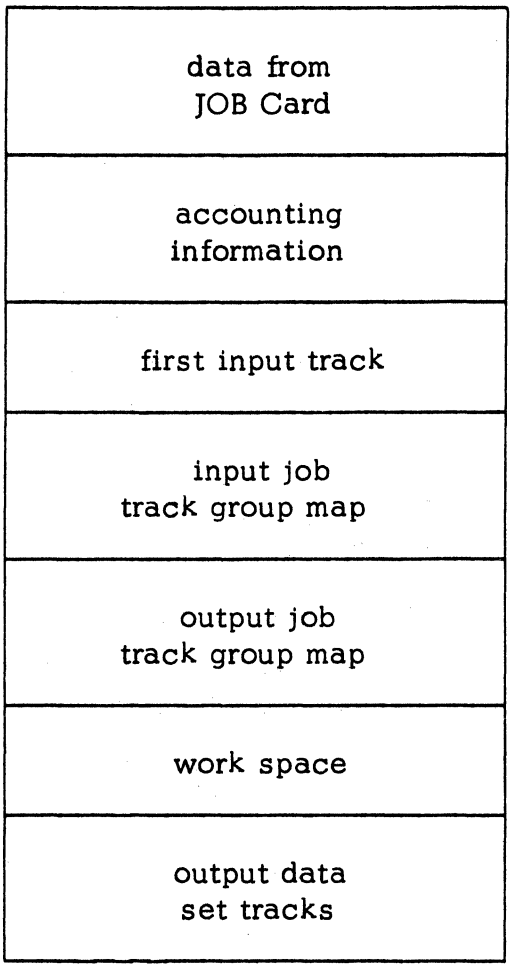


Figure 3.6.2 - The HASP Job Control Table (JCT)

The HASP Job Control Table is normally resident on a direct-access intermediate storage device. Once the HASP Job Queue Element is obtained, the "JCT track" in the element can be used to initiate a read into a HASP Buffer. Once this read has been completed, all information necessary to process the job can then be obtained.

3.7 ALLOCATION OF OVERLAY AREAS AND NON-RESIDENT CONTROL SECTIONS

Portions of the various programs of which HASP is composed are organized into non-resident control sections (CSECTs) and stored in an overlay library (OLAYLIB) on a direct-access volume. These control sections contain HASP re-entrant subroutines and/or data which may be requested for use by a Processor.

The user obtains an Overlay Area by requesting from the overlay control service program for use of a non-resident CSECT. If the CSECT requested is in main storage, the user is allowed to use the Overlay Area for processing. If, however, the CSECT is not already in an area, an area must be selected to hold the requested CSECT. The requesting Processor is made to "wait" until the requested CSECT is read from direct-access into main storage.

The algorithms for Overlay Area allocation cause multiple users of the same CSECT to use only one area, into which that CSECT is read. Competition for areas is resolved partially by the priority associated with each overlay CSECT. However, a "pre-empting" (roll) algorithm prevents any Processor from being indefinitely delayed, even if the system has only one Overlay Area.

The user releases an Overlay Area by requesting that overlay services remove his PCE from association with the area.

(The remainder of this page intentionally left blank.)

4.0 HASP PROCESSORS

This section contains detailed internal information about each of the HASP Processors and is intended primarily for use by system programmers.

4.1 INPUT SERVICE PROCESSOR

4.1.1 INPUT SERVICE PROCESSOR - GENERAL DESCRIPTION

The functions of the Input Service Processor are as follows:

- To read card images from an input device.
- To detect and scan JOB cards, extracting parameters for job accounting, job control, and print and punch identification.
- To detect and process other control cards such as the PRIORITY, MESSAGE, ROUTE, SETUP, COMMAND, DD*, and DD DATA cards.
- To assign a unique HASP job number to each job.
- To log jobs into the HASP System.
- To assign job priority based upon PRIORITY card or JOB card parameters.
- 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 (see Section 4.2).
- To generate HASP Job Control Tables, Job Queue Entries, and other HASP control blocks required for later job processing.
- To queue jobs for processing by the Execution Control Processor.

The Input Service Processor is coded re-enterably in such a way that it can accept jobs from a number of different input devices (with different hardware characteristics) simultaneously. The re-enterability is attained by retaining all storage unique to a job in the Processor Control Element (see figure 4.1.1) which must be unique for each input device.

4.1.2 INPUT SERVICE PROCESSOR - PROGRAM LOGIC

The Input Service Processor is divided into three phases, 13 subroutines, and three non-process 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 entire 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 a HASP buffer for use as an input buffer.

If the input device is not a remote terminal, a chain of Channel Control Words (CCW's) is then 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:

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

Job Card--The JOB card scan routine (HASPRJCS) is called to terminate the previous job (if any), to scan the JOB Card, and to initialize the PCE work area for the processing of the following job.

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 (see figure 8.15.1). The DD* or DD DATA statement is then added to the JCL file in normal fashion. Control is subsequently turned back to the main processor to process the input data.

When a hardware end-of-file is detected on the input device, or when "$DRAIN input device" command is entered by the operator, 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), issues a rewind and unload command to the input device if it is tape, frees the input buffer, closes the input DCT if it is a Remote Device, releases the input and direct-access devices, and returns control to Phase 1.

SUBROUTINES

HASPRCCS -- Subroutine to Process HASP /* Control Cards

The HASPRCCS subroutine, which is written as an overlay segment, is called whenever the Main Processor Phase encounters a /* control card. The control card type is first determined and then processing continues as follows:

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

/*SETUP Card -- The volumes to be mounted are listed on the operator's console and the job is placed in "hold" status.

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

If the control card type is not recognized, the card is ignored and treated like any other /* card.

HASPRJCS--Subroutine to Scan and 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. The job control information in the PCE Work Area (see figure 4.1.1) is initialized by scanning the JOB card and extracting parameters relative to job control. The first JCL block is initiated, and control is passed to the Job Initialization Subroutine: HASPRJBI.

RSCAN - RSCANA -- Subroutine to Scan Parameters from JOB Card

This subroutine has two entry points; the entry point: "RSCAN" is used to scan numeric parameters from the JOB card, while the entry point: "RSCANA" is used to scan alphameric parameters from the JOB card. There are also 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 final parameter on the JOB card was returned on the previous call and that there are no more parameters. If return is made to the fourth byte following the Branch and Link instruction, it indicates that parameter register "R1" contains the next parameter, right-adjusted with leading binary zeroes. If the parameter was a "null" parameter, "R1" will be zero. If this subroutine detects an illegal character (such as a non-numeric character in a numeric field) or more than four characters in a parameter, control is transferred to the RBADJOBC subroutine.

RCONTNUE -- Subroutine to Validate Continuation Cards

This subroutine 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 subroutine for further processing.

REBCDBIN -- Subroutine to Convert from EBCDIC to Binary

This subroutine expects to find numeric EBCDIC characters with leading binary zeroes in parameter register "R1". 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 parameter register now contains the binary equivalent of the EBCDIC input. If return is made to the fourth byte following the Branch and Link instruction, it indicates that the parameter register was zero (null parameter) and contained no EBCDIC to translate.

HASPRJBI -- Subroutine to Initialize Job Processing

This subroutine, which is written as an overlay segment, receives control from the JOB Card Scan Routine (HASPRJCS) and completes the initialization of the various control blocks for input job processing. A "job on" message is issued to the operator, the job's priority is assigned based upon JOB card or /*PRIORITY card parameters, and the job is queued in the active input queue. Control is then returned to the Main Processor Phase.

RBADJOBC -- HASPRIJC -- Subroutine to Process Illegal Job Cards

This subroutine notifies the operator of an illegal JOB card, calls the subroutine: "RJOBKILL" to delete the job, and returns control to the Main Processor Phase.

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. 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's JOB CLASS. Control is then returned to the calling program.

RGET -- Subroutine to Get Next Card from Input Buffer

This subroutine returns the address of the next card to be processed by the Input Service Processor in register "RPI". 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 a remote terminal, a "call" is made on the Remote Terminal Access Method to procure 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 "$STOP input device" and "$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 (see section 8.15). 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, it indicates that the processor is skipping for a JOB card and the subroutine returns without taking any action.

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 RJOBTERM subroutine is called to terminate the input processing of the job, and the job is placed in the Print Processor Queue for subsequent processing. Control is then returned to the calling program.

RJOBTERM -- Subroutine to Terminate Job

This subroutine terminates the last output buffer and schedules it for output. It then acquires a HASP buffer, and from information kept in the PCE Work Area (see figure 4.1.1) constructs the Job Control Table (JCT) and schedules it for output. Control is then returned to the calling program.

RGETBUF -- Subroutine to Initialize Output Buffers

This subroutine acquires a HASP buffer for an output buffer and returns with the address of the buffer in register "R1".

NON-PROCESS 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.

- RNOUNIT -- A HASP Unit was not available.
- RNOCMB -- A HASP Console Message Buffer was not available.
- 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.

Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT

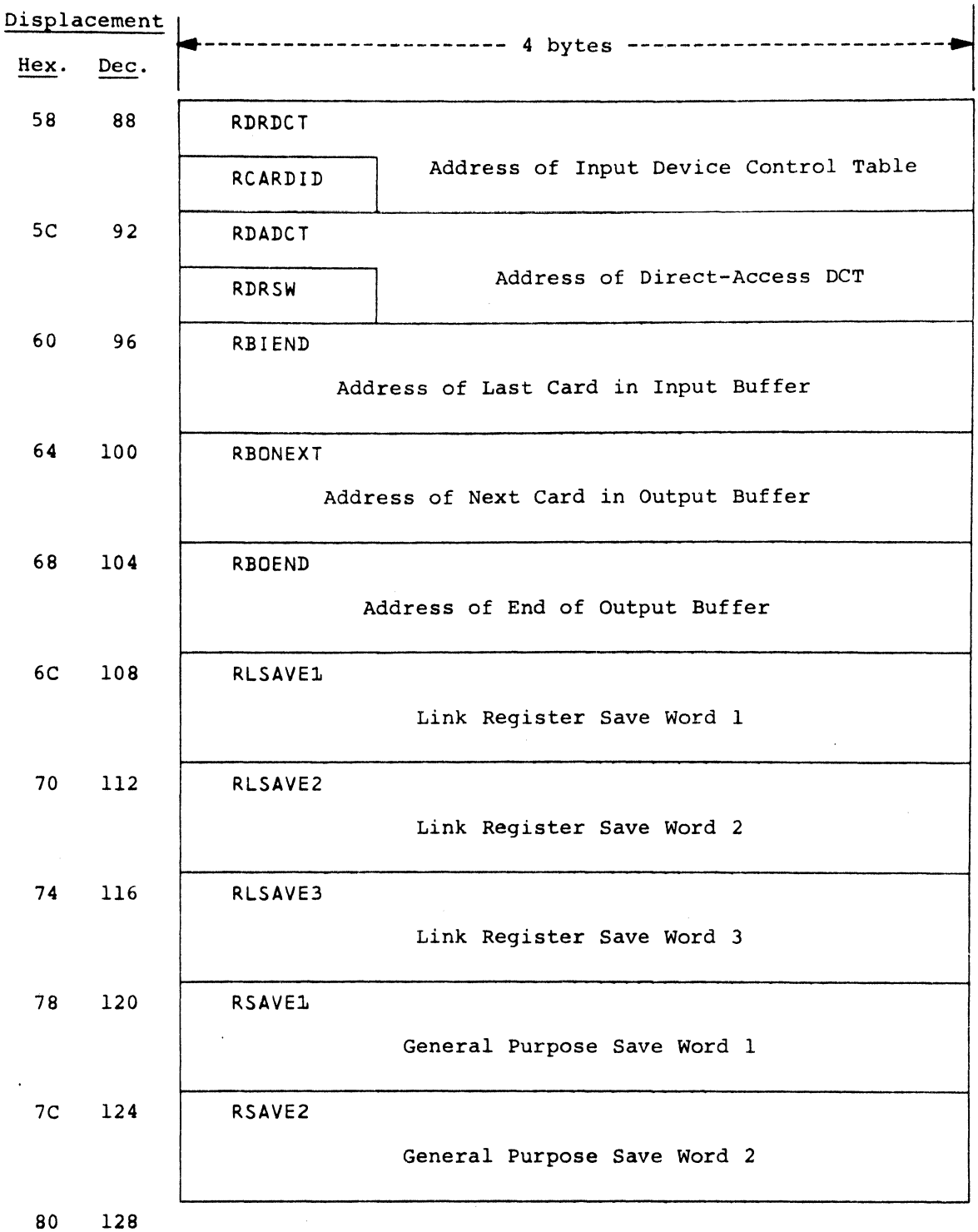


Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

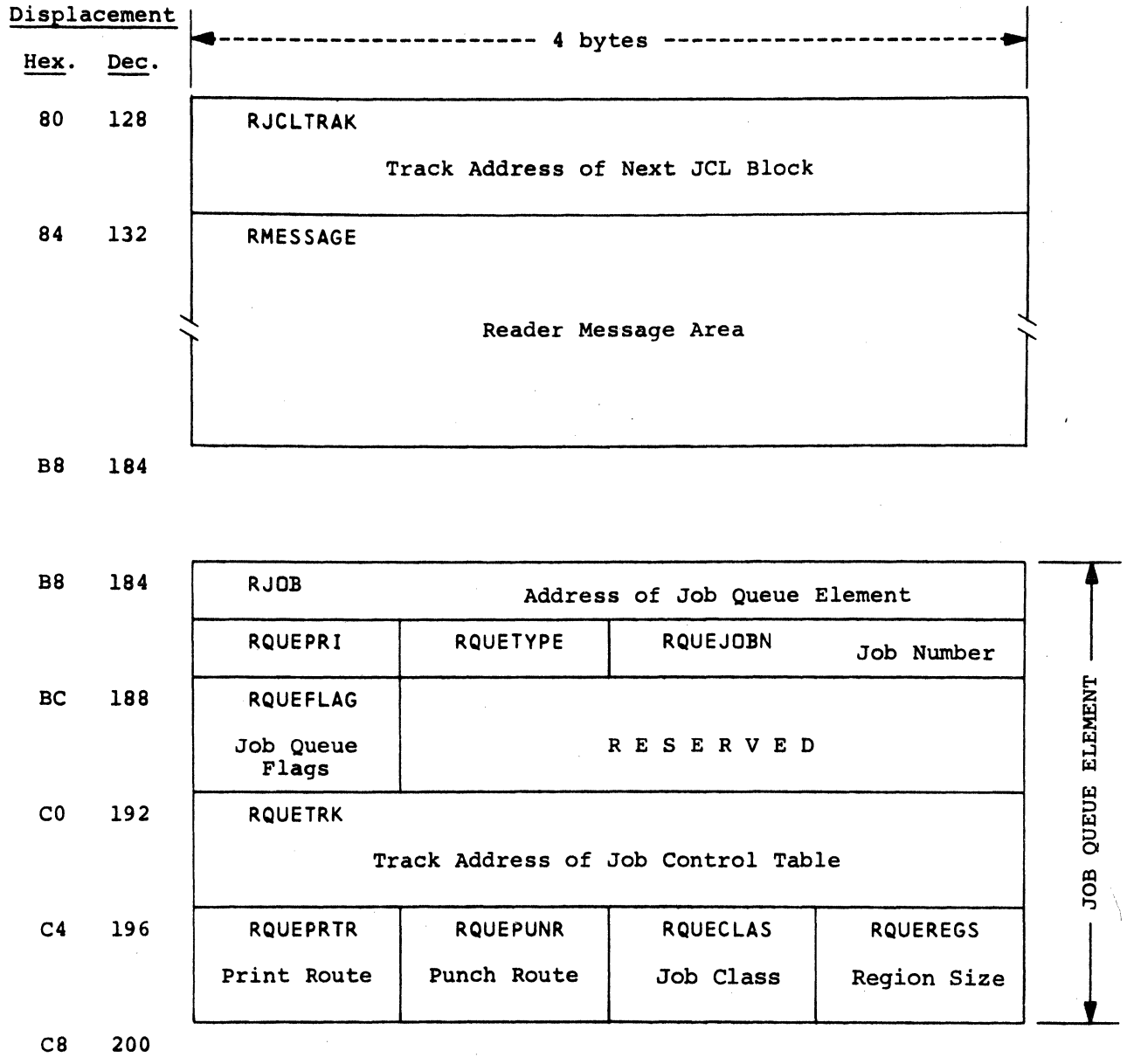


Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

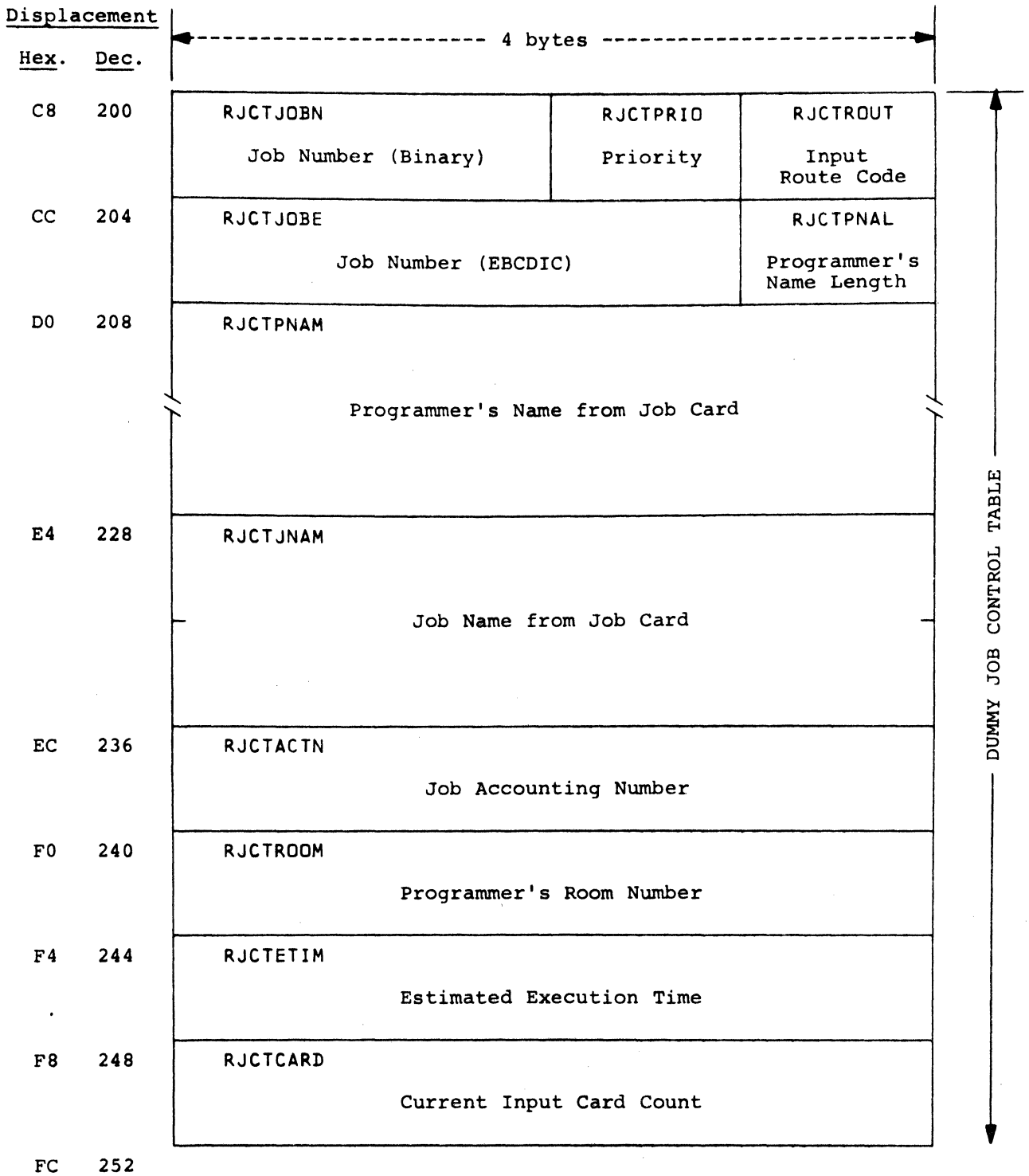


Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

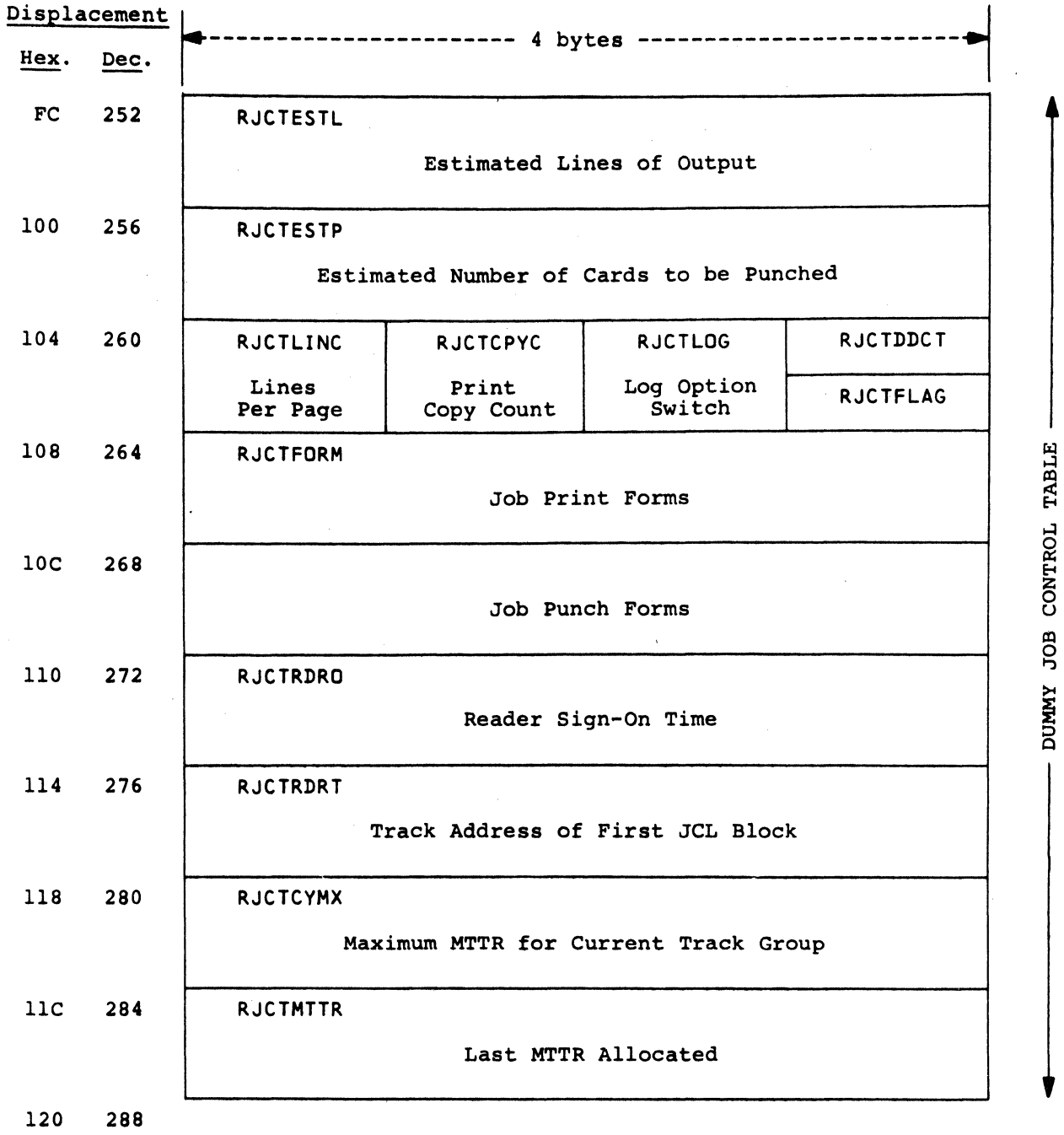


Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

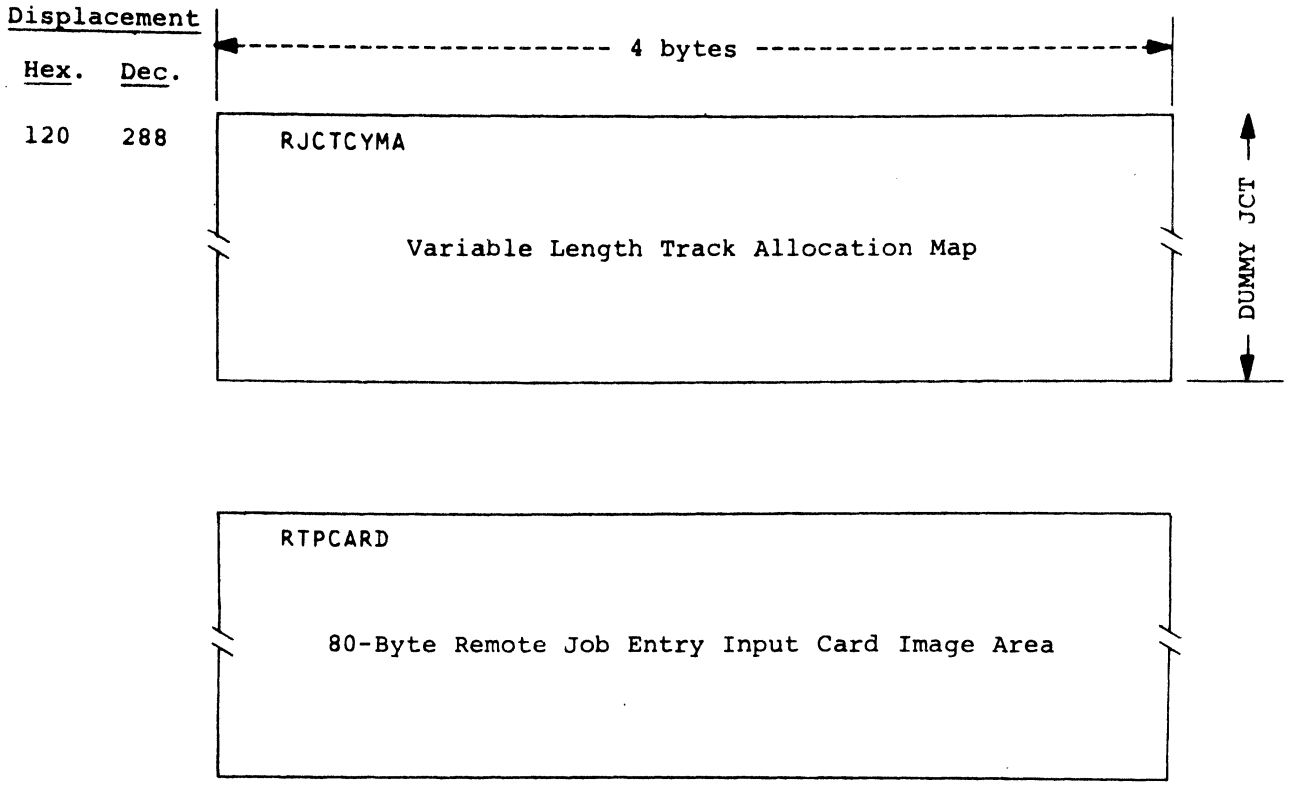


Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
58	88	RCARDID	1	Type of card being processed --

Hex. Value	Meaning
00	Normal Card.
03	Internally Generated Card.
04	HASP Control Card.
13	Illegal Control Card.
19	Last JCL Card.
73	Dummy Track Address Record.

58	88	RDRDCT	4	Address of Reader, Tape, Internal Reader, or Remote Device Control Table.
5C	92	RDRSW	1	Reader Switches --

Bit	Name	Meaning
0	RJOBQUED	Job has been Queued.
1	RSYSINSW	Processing Internally Generated DD * Card.
2	RXBJOBSW	Processing XEQ Batch Class Job.
3	ROSINSW	Processing O/S Input Data Set.
4	RJCLSW	Processing JCL.
5	RDREOFSW	End of File Indication.
6	RNOSCAN	Not Scanning JCL (DD DATA).
7	RJFLUSH	Job Flush Message has not been issued.

5C	92	RDADCT	4	Address of Direct-Access Device Control Table.
60	96	RBIEND	4	Address of Last Card in Input Buffer.
64	100	RBONEXT	4	Address of Next Card in Output Buffer.
68	104	RBOEND	4	Address of End of Output Buffer.
6C	108	RLSAVE1	4	Link Register Save Word 1.
70	112	RLSAVE2	4	Link Register Save Word 2.
74	116	RLSAVE3	4	Link Register Save Word 3.

Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
78	120	RSAVE1	4	General Purpose Save Word 1.
7C	124	RSAVE2	4	General Purpose Save Word 2.
80	128	RJCLTRAK	4	Track Address of Next JCL Block.
84	132	RMESSAGE	52	Reader Message Area.
B8	184	RJOB	4	Address of Job Queue Element (when Job has been Queued).
B8	184	RQUEPRI	1	Job Queue Priority (Before Queueing) --
		Bits 0-3		Priority (0-15).
		Bits 4-7		Zero.
B9	185	RQUETYPE	1	Job Class - X'80' (Before Queueing).
BA	186	RQUEJOBN	2	Job Number (Before Queueing).
BC	188	RQUEFLAG	1	Job Queue Flags --
				Bits 0 Name QUEHOLD1 Meaning: Job Held: TYPRUN=HOLD or Input Device Held.
				Bits 1-7 Meaning: Reserved.
BD	189		3	Reserved.
C0	192	RQUETRK	4	Track Address of Job Control Table.
C4	196	RQUEPRTR	1	Print Routing: 0 = Local. n = Remote n.
C5	197	RQUEPUNR	1	Punch Routing: 0 = Local. n = Remote n.
C6	198	RQUECLAS	1	Job Class - X'80' (After Queueing).
C7	199	RQUEREGS	1	Region Size -- Reserved.
C8	200	RJCTJOBN	2	Job Number (Binary).
CA	202	RJCTPRIO	1	Priority from /*PRIORITY Card.

Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
CB	203	RJCTROUT	1	Input Route Code: 0 = Local. n = Remote n.
CC	204	RJCTJOBE	3	Job Number (EBCDIC).
CF	207	RJCTPNAL	1	Programmer's Name Length.
D0	208	RJCTPNAM	20	Programmer's Name from Job Card.
E4	228	RJCTJNAM	8	Job Name from Job Card.
EC	236	RJCTACTN	4	Job Accounting Number.
F0	240	RJCTROOM	4	Programmer's Room Number.
F4	244	RJCTETIM	4	Estimated Execution Time.
F8	248	RJCTCARD	4	Current Input Card Count.
FC	252	RJCTESTL	4	Estimated Lines of Output.
100	256	RJCTESTP	4	Estimated Number of Cards to be Punched.
104	260	RJCTLINC	1	Lines per Page.
105	261	RJCTCPYC	1	Number of Copies of Print.
106	262	RJCTLOG	1	Log Option Switch.
107	263	RJCTDDCT	1	Count of Input Data Sets SPOOLed by HASP.
107	263	RJCTFLAG	1	JCT Flags.
108	264	RJCTFORM	4	Job Print Forms.
10C	268		4	Job Punch Forms.
110	272	RJCTRDRO	4	Reader Sign-On Time.
114	276	RJCTRDRT	4	Track Address of First JCL Block.
118	280	RJCTCYMX	4	Maximum MTTR for Current Track Group.
11C	284	RJCTMTTR	4	Last MTTR Allocated.

Figure 4.1.1 -- INPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
120	288	RJCTCYMA		Variable Length Track Allocation Map.
		RTPCARD	80	Remote Job Entry Input Card Image Area.

4.2 EXECUTION CONTROL PROCESSOR

4.2.1 Execution Control Processor — General Description

The Execution Control Processor is responsible for the interface between HASP and OS/360. It presents jobs to the Operating System for execution and communicates with the I/O supervisor to supply SYSIN data for a job and to accept SYSPRINT and SYSPUNCH from a job for later printing and punching.

This Processor is re-enterably coded and has the capability to present any number of jobs to OS/360 for simultaneous execution by maintaining unique INPUT/OUTPUT streams for each job. All storage unique to a job is retained in the Processor Control Element (see Figure 4.2.2) to provide re-enterability.

The Execution Control Processor is also responsible for monitoring job limit excessions (such as time, line, or punched card estimates). Jobs are selected for OS processing based on a logical partition structure defined by HASPGEN. Each logical partition is controlled by a partition information table (PIT) which indicates the eligibility of jobs for execution by that logical partition. There is a direct correlation between the HASP logical partition and the number of initiators active in the system. Jobs thus selected for OS processing are passed to a single OS/360 Reader/Interpreter which remains constantly STARTed to a

HASP pseudo device which appears as a 2540 card reader. Only the Job Control Language statements of a job are passed to the R/I. Input stream data sets, defined by DD* or DD DATA cards have been previously transcribed to a SPOOL disk by the HASP input service processor. The JCL for a job is dynamically modified by HASP to assign pseudo unit record devices to all SYSIN and SYSOUT data sets to permit interception by HASP. The job is interpreted by the R/I and is placed in the OS job queue for immediate selection by an initiator. At the completion of a job's execution, it is placed in the OS SYSOUT queue to be processed by an output writer. Because of the assignment of pseudo unit record devices to all SYSOUT files, the output writer is required only to "print" the System Message Blocks from SYS1.SYSJOBQE. These SMB's are intercepted by HASP and are stored on the SPOOL disks as another print data set. After receiving the last SMB, HASP terminates the XEQ phase, queues the job for the HASP output processors and indicates that the logical partition requires another job. All information concerning SYSIN and SYSOUT files assigned to HASP pseudo devices is kept in Data Definition Tables (DDT). There is a DDT for each active file of a job which indicates buffer addresses, file status, record count, etc. and is correlated with the proper file through the HASP pseudo device address.

4.2.2 Execution Control Processor — Program Logic

The Execution Control Processor (XEQ) consists of the three following logical phases:

PHASE 1 — Job Control — Initiates and terminates job processing.

PHASE 2 — Asynchronous I/O Handler — Interfaces with OS/360 via the Input/Output Supervisor (IOS) to perform SYSIN/SYSPRINT/SYSPUNCH I/O requests.

PHASE 3 — Synchronous I/O Handler — Performs the SPOOL I/O required by Phase 2.

Figure 4.2.1 indicates the relationship between these three phases and OS/360.

An OS execution is initiated by Phase 1 by obtaining a suitable job from the HASP job queue and reading its Job Control Table from disk. Job limit parameters are initialized and the status of the OS/360 R/I is interrogated. If the R/I is currently processing the input for another job, Phase 1 $WAITs until it has completed. A DDT describing the JCL file for the selected job is constructed and associated with the HASP pseudo 2540 used by the R/I. The dormant R/I is then POSTed to signal the availability of

a job and control is transferred to Phase 3 to await I/O requirements from Phase 2. The OS/360 Supervisor call table has been modified by HASP initialization so that all I/O requests are diverted to Phase 2 of the XEQ processor. If the I/O request thusly intercepted refers to a HASP pseudo device, it is processed by HASP; otherwise it is passed to the Operating System Input-Output Supervisor for normal processing. Since XEQ has the capability to control the simultaneous execution of many jobs, the PCE for the job issuing the I/O request to a pseudo device must be identifiable. This is done by using a combination of the JOB name and the TCB address (Job Step TCB for MVT). Once the PCE is located, the DDT for this particular pseudo device is found by the pseudo device address from the UCB. Phase 2 verifies that there is a buffer still associated with the file and simulates the I/O request. Each channel command word in the request is examined and, when a data select type is recognized, the I/O operation is simulated by a MOVE CHARACTERS to or from the current HASP buffer for that file. Input requests are serviced by stripping (deblocking) the next card image from the HASP buffer and moving it as indicated by the CCW. These moves (only) are done while operating under the requesting program's protect key to prevent an undetected protect violation by HASP, which normally operates under protect key zero. Requirements for I/O

activities associated with Phase 2 processing are indicated by a series of status bits in each DDT. Requests to get buffers, read buffers and write buffers, are indicated in the appropriate DDT, Phase 3 of the XEQ processor is $POSTed and the HASP task is POSTed. If the requested activity must be completed before an I/O request can be satisfied by Phase 2, the I/C requestor is made to WAIT. This is done by saving the current CCW location and using the OS WAIT routine to hold the requestor. When the required I/O activity is complete, the WAITing task is POSTed and the pseudo device I/O request is re-issued. At the end of all successful I/O operations, the appropriate user appendage (channel-end appendage, etc.) is entered, the I/O is POSTed complete if required and a CSW is constructed to indicate the normal I/O completion.

When Phase 3 of the Execution Processor is entered after initiation of the job it immediately enters a HASP $WAIT state to await direction from Phase 2. Upon being activated via a $POST from Phase 2 or by a timer interrupt, this PHASE examines various status bits in the PCE and DDT's to determine the required action. This action may be either the priming of an input buffer, writing and re-initialization of an output buffer, or the notification to the operator of expiration of the estimated time of the job. An input buffer is primed by obtaining the track address

of the next buffer from the current buffer and issuing a read for the record. Status bits are set in the DDT to indicate that a read is in progress on this buffer and are reset at channel end time to indicate that the record is available. A full output buffer from Phase 2 is scheduled for transcription to disk and a new buffer is immediately obtained and initialized for use. When the buffer is initialized a track address is acquired and inserted as a forward chain in the buffer to be written. If Phase 3 is for any reason unable to get a HASP buffer, a special service called Buffer-roll is invoked. The function of Buffer-roll is to make a HASP buffer currently being utilized by another file (in this or another job) available to the requestor. This is done by selecting a low frequency DDT which owns a buffer and forcing a "free" of that buffer. To free a primary or secondary input buffer, a switch is set in the DDT to force a re-read of the buffer when the input file is next required. Output buffers are freed by terminating and writing the buffer to the SPOOL disk. Future references to this output file will cause a new buffer to be obtained and chained to the partial buffer.

A count of the number of logical records contained in each output buffer is maintained by the Phase 2 routine and is used by Phase 3 upon writing a buffer to maintain the total line and card count for this job. This accumulated figure is also compared, after each write, to

the estimated number of output records with the operator being notified on its excession. If a job exceeds either cards, lines, or time, the operator is so advised and a HASPGEN value is added to the original estimate which will cause repeated excession messages as this new estimate is reached. The job continues through normal OS/360 processing until the end of execution is reached. The job, as part of normal OS job termination, is then placed in the OS SYSOUT queue for processing by an output writer. Because of the dynamic modification of all SYSOUT= cards to pseudo devices, the only data set to be processed by the output writer is that containing the System Message Blocks. The Output Writer therefore "prints" the SMB's to a HASP pseudo device. When the last SMB is received, Phase 3 is notified (via an OS POST) to return control to Phase 1 for HASP job termination.

The job termination section of Phase 1 must now prepare the job for passage to the print queue. First, all partially filled output buffers are truncated and written out, and all input buffers are freed. The timer interval for the job is cancelled and all job execution statistics are added to the JCT. At this point the areas of the SPOOL disks used to store the job input information are made available to be re-allocated by HASP (Purged), the JCT is written to disk and the job is passed to

the print queue for printing. If no priority card was present, the job priority is recalculated as a function of the number of lines of print generated before the job is placed in the print queue.

A branch is then made to the beginning of XEQ to begin another job if available, or to display a message indicating that the logical partition is idle.

The process of dynamic examination and modification of selected JCL statements is accomplished by invoking the standard OS Reader/ Interpreter exit list option which allows users to inspect all JCL encoded by the reader. A detailed discussion of the HASP processing algorithm is contained in Appendix 12.8: HASP JCL Processing.

Figure 4.2.1 -- Execution Control - OS/360 Relationship

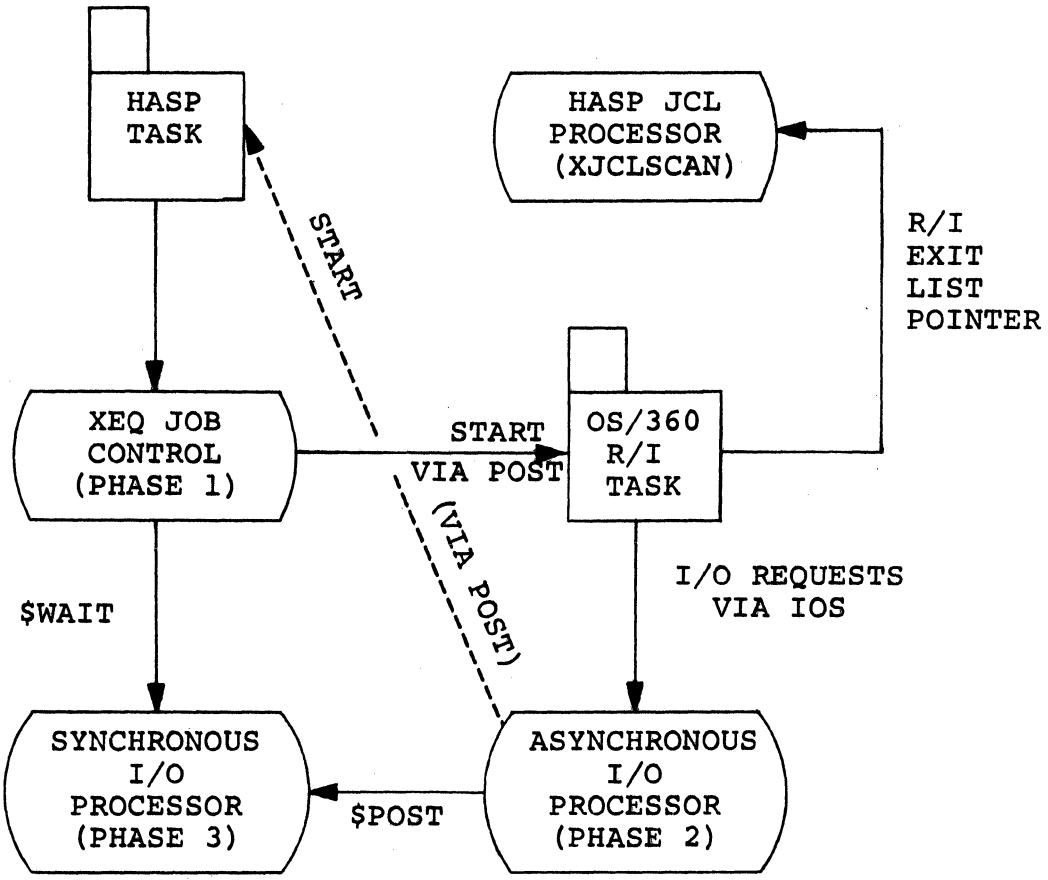


Figure 4.2.2 -- EXECUTION CONTROL PCE WORK AREA FORMAT

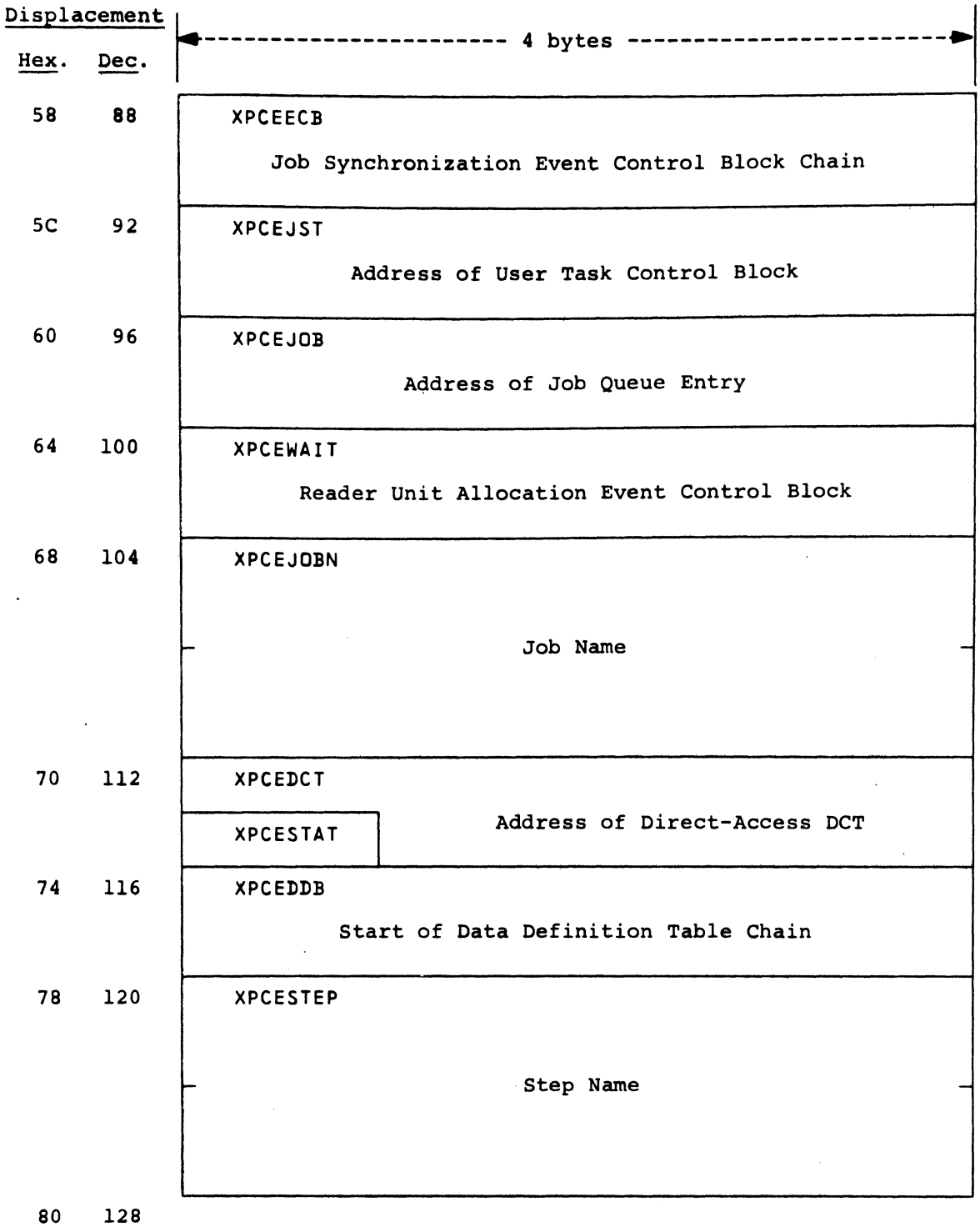


Figure 4.2.2 -- EXECUTION CONTROL PCE WORK AREA FORMAT (CONTINUED)

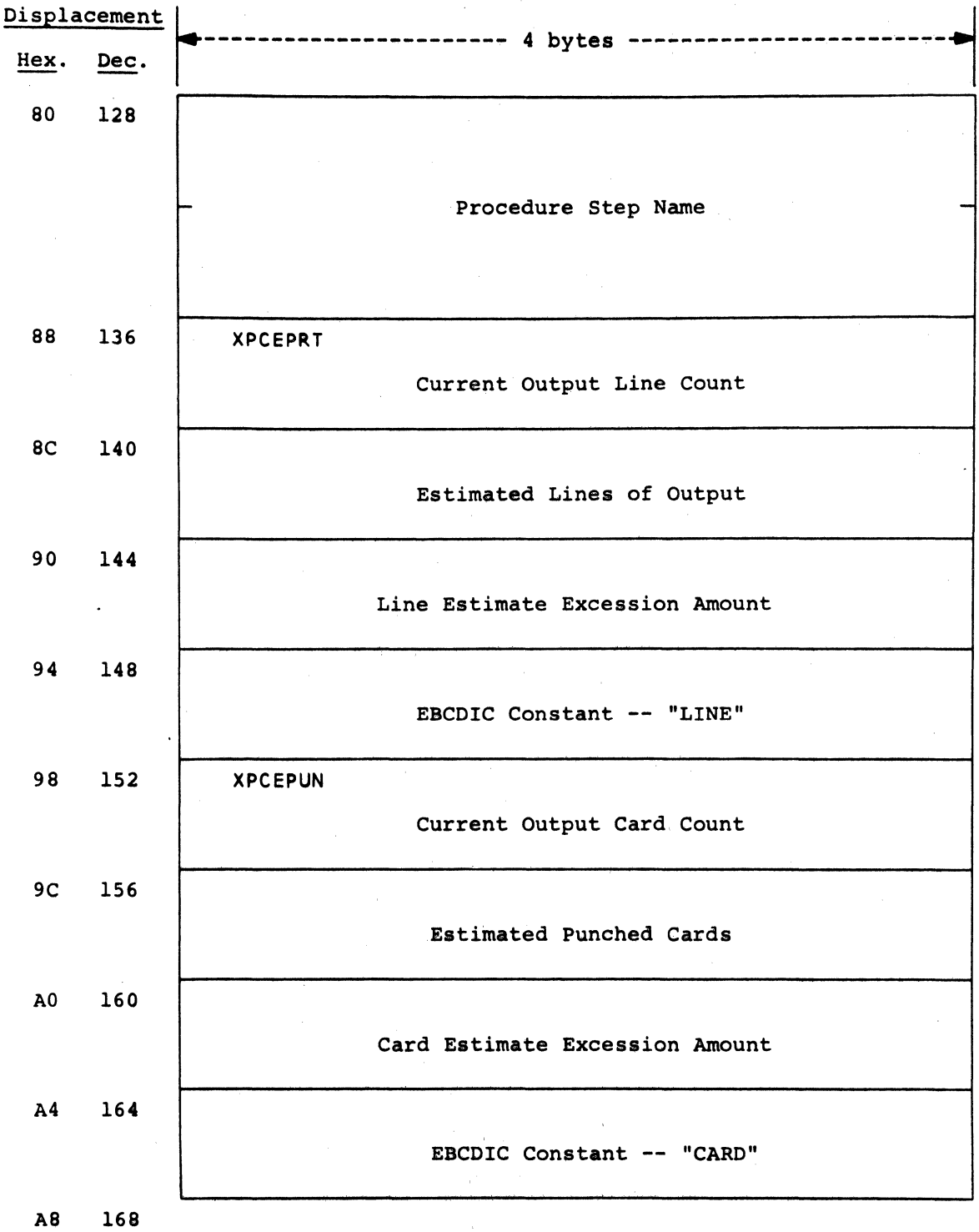


Figure 4.2.2 -- EXECUTION CONTROL PCE WORK AREA FORMAT (CONTINUED)

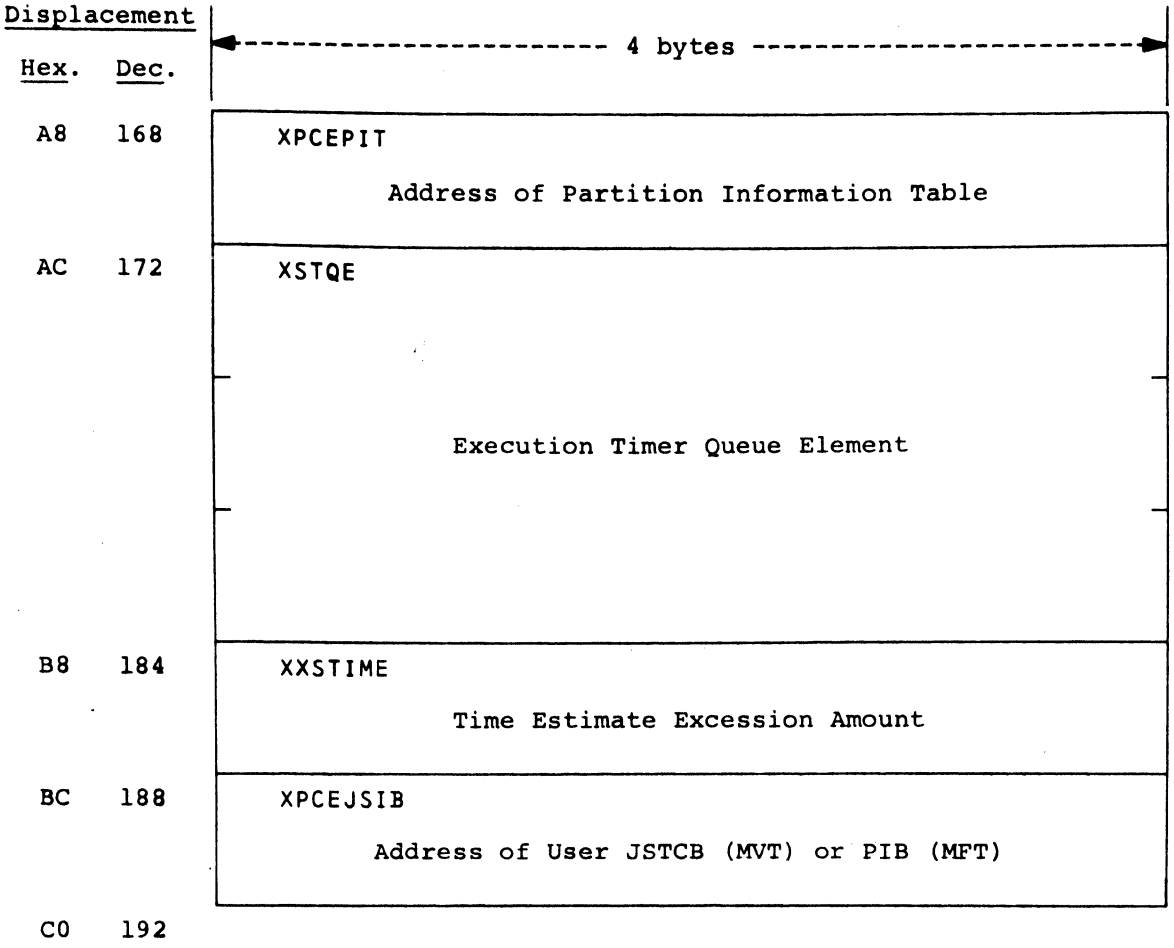


Figure 4.2.2 -- EXECUTION CONTROL PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
58	88	XPCEECB	4	Job Synchronization Event Control Block Chain.
5C	92	XPCEJST	4	Address of User Task Control Block.
60	96	XPCEJOB	4	Address of Job Queue Entry.
64	100	XPCEWAIT	4	Reader Unit Allocation Event Control Block.
68	104	XPCEJOBN	8	Job Name.
70	112	XPCESTAT	1	Status --

Bit	Name	Meaning
0-1		Reserved.
2	XPOSTBIT	POST Request for XTHAW.
3	XRDRACT	Reserved.
4	XEOJMES	End Execution Message Sent.
5	XDUPBIT	Job with Duplicate Job Name Waiting.
6	XUCBDDB	UCB/DDT Required by Execution Interface.
7	XEOJBIT	End of Job Flag.

70	112	XPCEDCT	4	Address of Direct-Access DCT.
74	116	XPCEDDB	4	Start of Data Definition Table Chain.
78	120	XPCESTEP	8	Step Name.
80	128		8	Procedure Step Name.
88	136	XPCEPRT	4	Current Output Line Count.
8C	140		4	Estimated Lines of Output.
90	144		4	Line Estimate Excession Amount.
94	148		4	EBCDIC Constant -- "LINE".
98	152	XPCEPUN	4	Current Output Card Count.
9C	156		4	Estimated Punched Cards.

Figure 4.2.2 -- EXECUTION CONTROL PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
A0	160		4	Card Estimate Excession Amount.
A4	164		4	EBCDIC Constant -- "CARD".
A8	168	XPCEPIT	4	Address of Partition Information Table.
AC	172	XSTQE	12	Execution Timer Queue Element.
B8	184	XXSTIME	4	Time Estimate Excession Amount.
BC	188	XPCEJSIB	4	MVT -- Address of Job Step Task Control Block. MFT -- Address of Partition Information Block.

4.3 OUTPUT SERVICE PROCESSOR (PRINT AND PUNCH)

4.3.1 OUTPUT SERVICE PROCESSOR - GENERAL DESCRIPTION

The functions of the Output Service Processor are as follows:

- To convert the print and punch output generated by the Execution Control Processor to hard copy.
- To provide for the unique identification of both print and punch output to facilitate collection and delivery.
- To provide for the routing of special data sets to printers and punches reserved for special forms processing.
- To produce multiple copies of print output upon request.
- To count print lines and produce automatic page overflow.
- To translate all illegal print characters to blanks (optional).
- To load the Universal Character Set Buffer (optional).
- To load the Forms Control Buffer (optional).
- To provide additional information for checkpoint which allows print to continue in the event of a "warm start".
- To punch a Job Accounting Card (optional).
- To process all printer and punch I/O errors with automatic error recovery (no operator intervention).
- To respond to all operator commands directed toward any printer or punch.
- To queue jobs for the next stage of processing when the current print/punch function has been completed.

The Output Service Processor is coded re-enterably in such a way that it can deliver output to a number of different output devices simultaneously. The re-enterability is attained by retaining all storage unique to a job in the Processor Control Element (see figure 4.3.1) which must be unique for each output device.

4.3.2 OUTPUT SERVICE PROCESSOR - PROGRAM LOGIC

The Output Service Processor is divided into three phases, nine subroutines, and two non-process 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 begins by attempting to get an output unit. If an output unit is not available, the processor enters a HASP $WAIT state until a device is made available and then the process is repeated.

Next, an output function is determined. If the device acquired is a remote printer, the appropriate entry in the Remote Message Table is examined to determine if any remote messages have been queued, and if so processing continues. The general purpose register: "JCT" is set to zero to indicate that remote messages are being processed.

If the device is not a remote printer, or if there are no messages queued, an attempt is made to obtain a job from the Job Queue which matches the type, routing and special forms of the device obtained. If no jobs are queued which fit these qualifications, the special forms processing type is checked to see if the forms requirement can be dropped. If so, another attempt is made to obtain a job from the Job Queue which matches the type and routing specifications only.

If a job cannot be found, then the output unit is released and control is returned to the start of the Initialization Phase.

If the output device is a remote terminal, output activity is initiated by calling upon the Remote Terminal Access Method (RTAM) to "open" the Remote Device Control Table.

The processor then acquires a direct-access Device Control Table (DCT) and a HASP buffer into which the Job Control Table (JCT) is then read. A message is sent to the operator notifying him that a particular job is now on the respective device and the initialization of the Processor Control Element Work Area (see figure 4.3.1) is completed.

If the processor is processing print output, and if the output is not a data set which has been routed for special forms, the PRINTID subroutine is called to generate the print identification header and control is transferred to Phase 2.

If the processor is processing punch output, and if the output is not a data set which has been routed for special forms, the Punch ID Card is generated for later punching, and control is transferred to Phase 2.

Phase 2 - Main Processor

The function of the Main Processor is to read the data blocks which are produced by the Execution Control Processor and build a channel program to print or punch the data. The PRDBUF and PRDCHK subroutines are used to read the data blocks, the PPPUT subroutine is used to construct the channel program and the PPWRITE and PPCHECK subroutines are used to initiate and check the execution of the channel program.

If the processor is processing print output, the "Control Byte" fields of the Data Block (see figure 8.15.1) are used to build the CCW operation codes. These control bytes are also used to count the actual lines of paper spaced and when this line count exceeds the parameter JCTLINCT, an eject is inserted to force a new page and the count is restarted. If an illegal control byte is encountered, or if the operator has entered a "$T PRTn,C=1" command, a single-space CCW is generated and used rather than the one provided in the data block. In such cases line counting continues and automatic page overflow is still provided.

If the processor is processing punch output, a "Punch, Feed, and Select Stacker P2" command is generated.

When the last data block has been printed or punched, control is transferred to Phase 3.

Phase 3 - Processor Termination

The Processor Termination Phase first reads the Job Control Table and scans the Peripheral Data Description Blocks (see figure 8.8.1) for the next data set to be processed. If another data set is encountered, control is returned to Phase 2 for processing. If no more data sets are to be processed, the termination phase then proceeds depending upon the type of output which is being processed.

If the processor is processing print output, the "Print Copy Count" field in the JCT (see figure 8.8.1) is compared with the current number of copies which have been printed. If more copies are needed, control is transferred to Phase 1 for the production of another copy. If no more copies are required, the PRINTID subroutine is called to generate the print identification trailer.

If the processor is processing punch output, the job accounting subroutine is called, and the accounting card is punched followed by a blank card to clear the punch and check the punching of the Job Accounting Card.

The Job Control Table is then re-written, the Job Queue Element is passed to the next processor queue, the Device Control Tables are released, and control is transferred to the start of Phase 1.

SUBROUTINES

PLOADUCS -- Subroutine to Load the UCSB and FCB

This subroutine determines the Universal Character Set Type from the Printer Device Control Table. The UCSB Table is then searched and the corresponding UCS image (if one is found) is $LOADed and moved into a HASP buffer. The UCS Buffer is then loaded using the PPPUT, PPWRITE, and PPCHECK subroutines.

If the output device type specifies a 3211 printer, then the Forms Control Buffer is loaded in a manner similar to the UCS Buffer. After loading the FCB, the FCB type is reset so that no more FCB loads will occur until the operator specifies that the buffer should be re-loaded.

PRINTID -- Subroutine to Generate Print Identification

This subroutine builds up the line image which is used to produce the Print Identification Page from information in the Job Control Table and information passed to the subroutine at the time it is called. This line image is built up in the "Job Accounting Storage" section of the Job Control Table (see figure 8.8.1). The subroutine then builds a channel program which starts with an eject command and follows with enough print commands to completely fill a page with print identification lines. The channel program is then executed and checked and control is returned to the calling program. The PPPUT subroutine is used to construct the channel program, and the PPWRITE and PPCHECK subroutines are used to initiate and check the execution of the channel program.

PPFORMCK -- Subroutine to Mount Forms

This subroutine compares the forms being requested with the forms currently mounted on the associated device. If a match is found, the subroutine returns immediately. Otherwise, a forms mount message is issued to the operator and the subroutine $WAITs for a "$Sdevice" command to be entered. The DCT Forms field is then set to reflect the new forms type and processing continues.

PRCOMENT -- Subroutine to Add Comment to Printer Output

This subroutine constructs and adds to the printer output (using the PPPUT, PPWRITE, and PPCHECK subroutines) a comment of the form:

PRINT xxxxxxxxx BY OPERATOR.

"xxxxxxxxx" is specified at subroutine entry by parameter register "R1" and will be one of the following:

DELETED
RESTARTED
REPEATED
BACKSPACED
FWD-SPACED
SUSPENDED

PRDBUF -- Subroutine to Initiate Read from Direct Access Storage

This subroutine initiates a read from the track address specified by register "PNP" into the appropriate HASP buffer.

PRDCHK -- Subroutine to Check Read from Direct Access Storage

This subroutine checks the read initiated by the PRDBUF subroutine. If the read is not complete, the processor is placed into a HASP $WAIT state until the read is completed. If an I/O error is detected, a "$IOERROR" macro-instruction is issued and the processing of the rest of the data set is deleted.

This subroutine also checks for any operator command which would cause the Main Processing Phase to be completed and forces any indicated completion by zeroing the chain track in the data block just read.

PPPUT -- PPUTOLAY -- Subroutine to Build a Channel Program

This subroutine accepts a CCW from the calling program and, if the output device is not a remote terminal, constructs a channel program in the Processor Control Element Work Area (see figure 4.3.1). Each command is examined and if it is an immediate printer space or skip, and if the previous command was a "Write, No Space", the two commands are combined into one. When the channel program storage area is full, this subroutine calls the PPWRITE subroutine to initiate the execution of the channel program. Upon the next entry, the execution of the channel program is checked by calling the PPCHECK subroutine.

If the output device is a remote terminal, the Remote Terminal Access Method is "called" to process the output line or card. Control is then given to the PPCHECK subroutine to test for operator commands.

PPWRITE--Subroutine to Initiate Execution of the Channel Program

If the output processor is being deleted by operator action, this subroutine returns immediately. Otherwise a write is initiated on the respective output device, using the channel program developed by the PPPUT subroutine.

PPCHECK--Subroutine to Check the Execution of the Channel Program

This subroutine checks for the successful completion of the channel program execution initiated by the PPWRITE subroutine. If the execution has not yet completed, the subroutine enters the processor into a $WAIT condition until the output has been completed. If an unsuccessful completion is detected, the subroutine performs the error recovery described in the paragraph below. This subroutine also interprets all operator commands directed at the processor and initiates appropriate action.

NON-PROCESS EXITS

The following routines are used to place the Output Service Processor into a HASP $WAIT state if a HASP resource is not available. In both cases the non-process register ("PNP") must have been set to the restart address before the routine is entered.

- PNOUNIT -- A HASP unit was not available.
- PNOBUF -- A HASP buffer was not available.

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

PRINTER "WARM START" LOGIC

When the Output Service Processor is successful in acquiring a job from the print queue, the print checkpoint area is searched for an available Print Checkpoint Element (see figure 4.3.2). This element is thereafter used to record the job number, copy count, and line and page counts.

In the event of a "warm start", the elements are searched and each Print Checkpoint element is moved into the Job Control Table for the job which it represents.

When the job is printed, the JCT is examined, and if the Print Checkpoint Element is present, the processor continues printing from the point when the last checkpoint was taken.

OUTPUT PROCESSOR BUFFER LOGIC

The buffer logic that the output processor employs is determined by the HASPGEN parameters: $PRTBOPT, $PUNBOPT, $RPRBOPT, and $RPUBOPT.

Buffer Option = 1

One buffer will be obtained at the beginning of output processing and will be used through the entire processing of a job's output. A read for the following data block will not be initiated until the current data block has completed its output. Periods of high Input/Output activity could cause the printers and punches to operate at less than their maximum rate when this option is used.

Buffer Option = 2

Two buffers will be obtained at the beginning of output processing and will be used through the entire processing of a job's output. A read for the following data block will be

initiated as soon as the previous data block has completed its output and will be performed while the current data block is completing its output. This option represents the most efficient utilization of the output devices.

PRINT AND PUNCH ERROR RECOVERY

Print Errors

The operator will be informed of all printer errors, but they will be ignored by the Output Service Processor.

Punch Errors

The card which causes a punch check and the card following this card are selected automatically into the reject stacker. The Output Service Processor will attempt to punch these two cards correctly until no error occurs or the operator deletes the job. Since all normal punch output is selected to another stacker, no operator intervention will be required to clear the punch. Every error will be recorded on the operator's console.

Figure 4.3.1 -- OUTPUT SERVICE PCE WORK AREA FORMAT

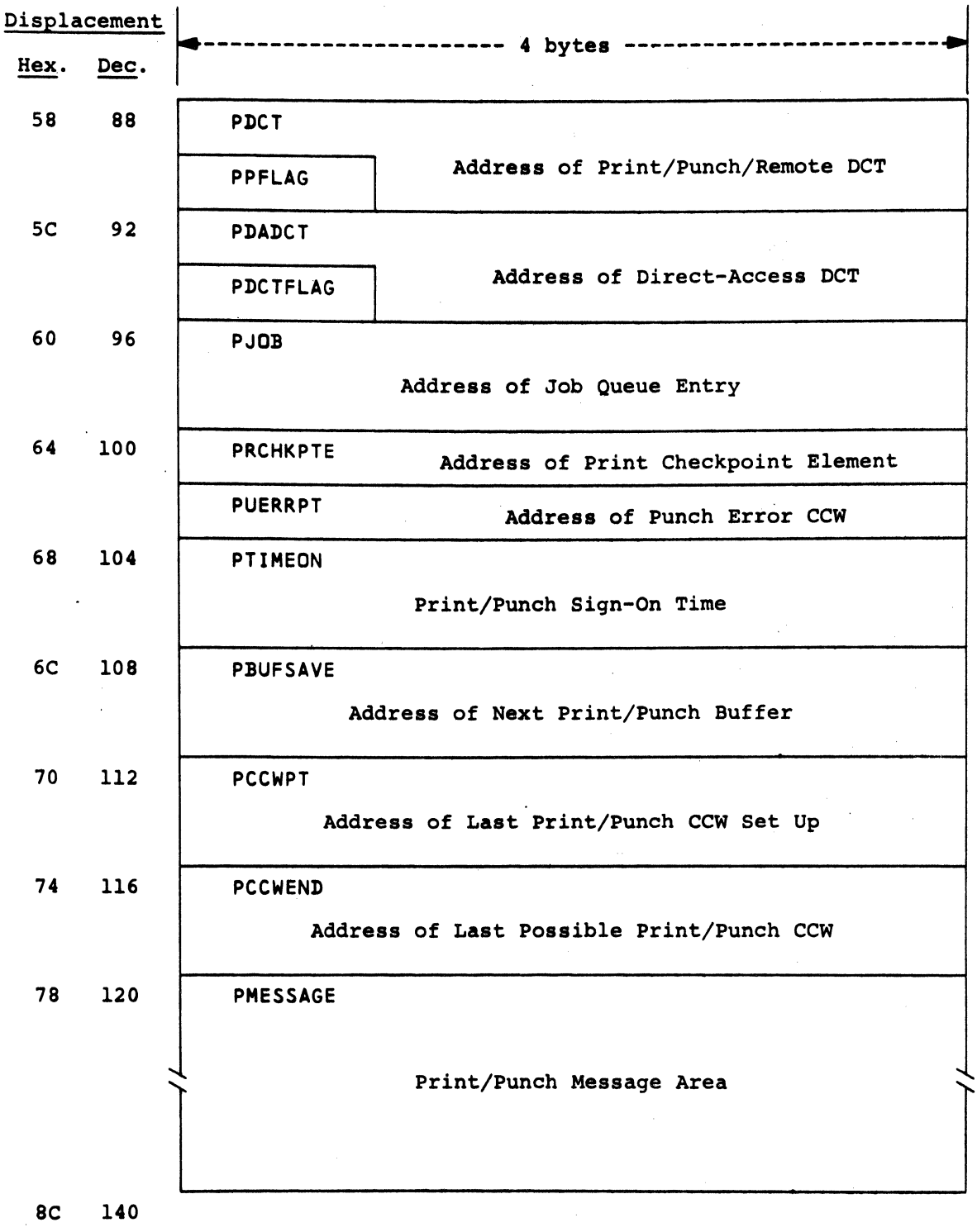


Figure 4.3.1 -- OUTPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

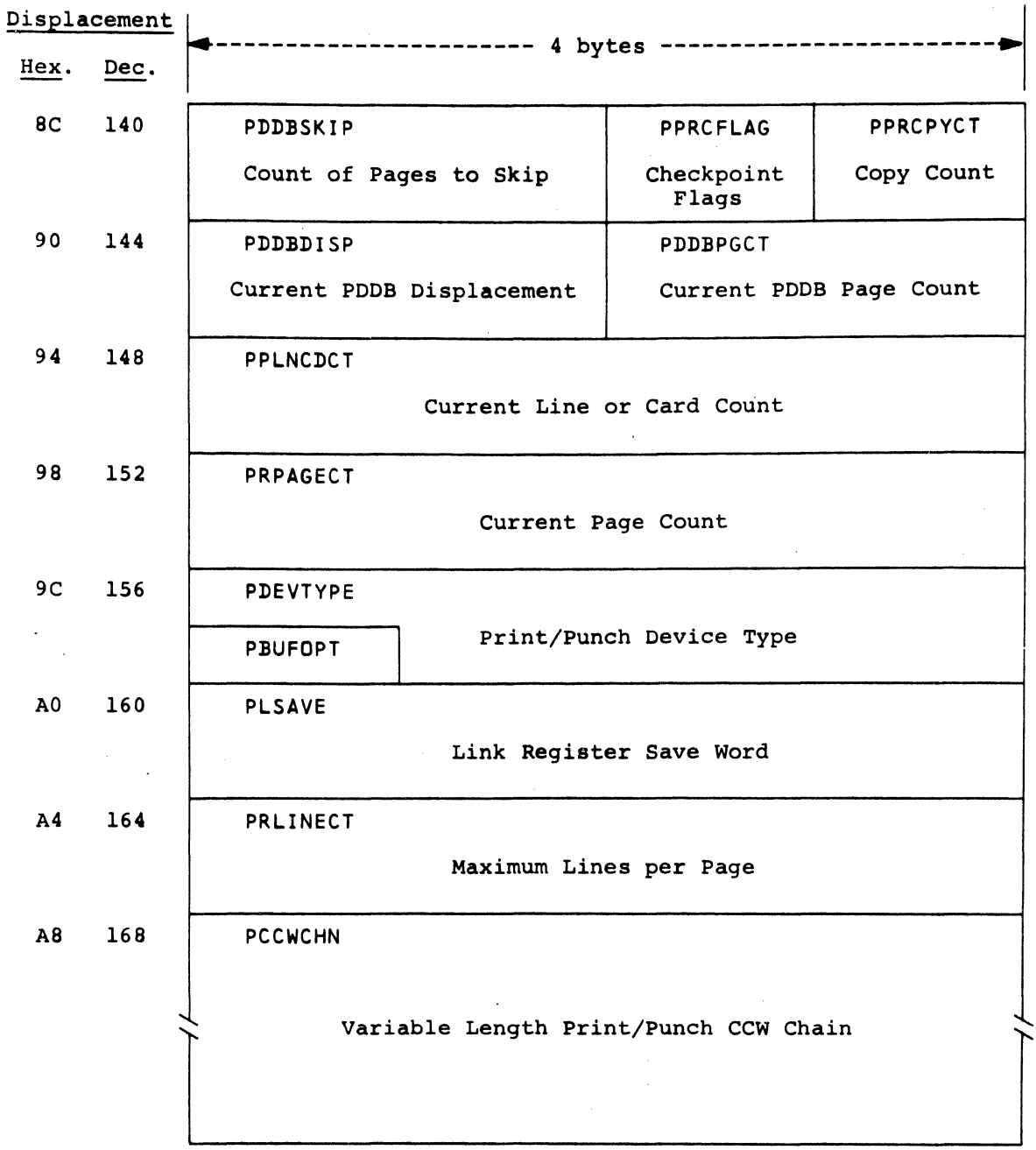


Figure 4.3.1 -- OUTPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
58	88	PPFLAG	1	Print/Punch Synchronization Flags --

Bit	Name	Meaning
0	PPWSW	Write has been Initiated.
1	PPDELSW	Function has been Deleted.
2	PPNOJOB	No Job is Active.
3	PRDELSW	Print was Deleted by Operator.
4	PRRSTSW	Print was Restarted by Operator.
5	PPRDERR	Function Terminated by Read Error.
6-7		Reserved for Future Use.

58	88	PDCT	4	Address of Print/Punch/Remote Device Control Table.
5C	92	PDCTFLAG	1	Print/Punch/Remote Operator Commands --

Bit	Name	Meaning
0	DCTSTOP	$Z ($STOP) Command.
1	DCTDELET	$C ($DELETE) Command.
2	DCTRSTRT	$E ($RESTART) Command.
3	DCTRPT	$N ($REPEAT) Command.
4	DCTBKSP	$B ($BACKSPACE) Command.
5	DCTSPACE	$T...,C=1 Command.
2+4		$I Command.
6-7		Reserved.

5C	92	PDADCT	4	Address of Direct-Access Device Control Table.
60	96	PJOB	4	Address of Job Queue Entry.
64	100	PRCHKPTE	4	Print Only: Address of Print Checkpoint Element.
64	100	PUERRPT	4	Punch Only: Address of Punch Error CCW.
68	104	PTIMEON	4	Print/Punch Sign-On Time.
6C	108	PBUFSAVE	4	Address of Next Print/Punch Buffer.

Figure 4.3.1 -- OUTPUT SERVICE PCE WORK AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
70	112	PCCWPT	4	Address of Last Print/Punch CCW Set Up.
74	116	PCCWEND	4	Address of Last Possible Print/Punch CCW.
78	120	PMESSAGE	20	Print/Punch Message Area.
8C	140	PDDBSKIP	2	Count of Pages to Skip.
8E	142	PPRCFLAG	1	Checkpoint Flags --

Bit	Name	Meaning
0	PRCHKUSE	Checkpoint Element Assigned.
1	PRCHKJOB	Job Active Indication.
2-7		Reserved.

8F	143	PPRCPYCT	1	Current Copy Count.
90	144	PDDBDISP	2	Current PDDB Displacement.
92	146	PDDBPGCT	2	Current PDDB Page Count.
94	148	PPLNCDCT	4	Current Line or Card Count.
98	152	PRPAGECT	4	Current Page Count.
9C	156	PBUFOPT	1	Buffering Option --

Value	Meaning
1	Single Buffering.
2	Double Buffering.

9C	156	PDEVTYPE	4	Device Type from UCB (UCBTYP).
A0	160	PLSAVE	4	Link Register Save Word.
A4	164	PRLINECT	4	Maximum Lines per Page.
A8	168	PCCWCHN		Variable Length Print/Punch CCW Chain.

Figure 4.3.2 -- PRINT CHECKPOINT ELEMENT FORMAT

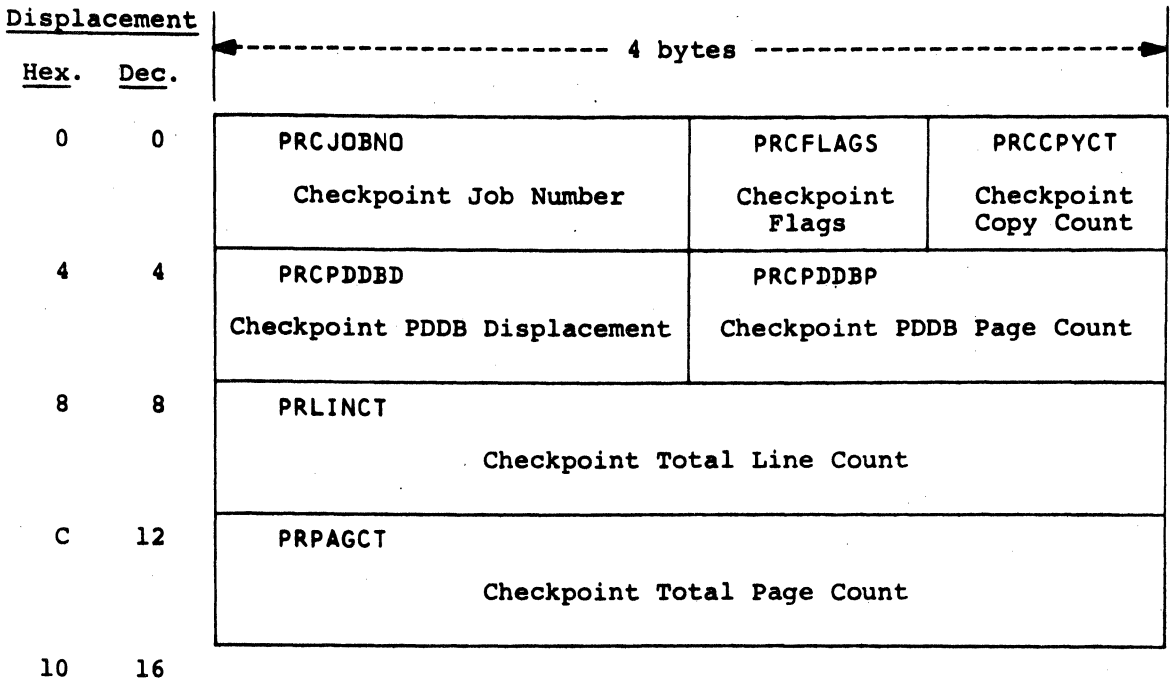


Figure 4.3.2 -- PRINT CHECKPOINT ELEMENT FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	PRCJOBNO	2	Job Number.
2	2	PRCFLAGS	1	Checkpoint Flags --

Bit	Name	Meaning
0	PRCHKUSE	Checkpoint Element in Use.
1	PRCHKJOB	Job Active Indication.
2-7		Reserved for Future Use.

3	3	PRCCPYCT	1	Current Copy Count.
4	4	PRCPDDBD	2	Current PDDB Displacement.
6	6	PRCPDDBP	2	Current PDDB Page Count.
8	8	PRLINCT	4	Total Line Count.
C	12	PRPAGCT	4	Total Page Count.

4.4 PURGE PROCESSOR

4.4.1 PURGE PROCESSOR - GENERAL DESCRIPTION

The Purge processor frees the job's acquired HASP direct-access space and removes the Job Queue Element from the system.

4.4.2 PURGE PROCESSOR - PROGRAM LOGIC

The processor first 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 a HASP buffer are acquired and initialized so that the job's Job Control Table (JCT) 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 then issued to return the job's direct access tracks. If a permanent I/O error occurs while the JCT is being read, the DISASTROUS error routine is called and the $PURGE macro instruction is not executed. Next, the Job Queue Element is removed from the HASP Job Queue and the following message is issued to the operator:

JOB xxx IS PURGED

Finally, the buffer 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.

4.5 HASP COMMAND PROCESSOR

4.5.1 HASP Command Processor - General Description

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.

4.5.2 HASP Command Processor - Program Logic

The HASP Command Processor is initially entered at the beginning of the Control Section (CSECT) HASPCOMM which is a part of the resident portion of HASP. Subsequent re-entries are returns from the various command sub-processors 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 queue $COMMQUE is examined for the presence of the next command to process. If no buffer is queued, the Command Processor waits on WORK. When $POSTed or if a buffer is present upon entry, the Command Edit Routine is entered via $LINK macro.

Command Edit Routine - HASPCOME

VERB CONVERSION - The Command Edit Routine converts the command text from the long form to the standard single character verb form. The data portion of the Console Message Buffer up to the first comma (,) or apostrophe (') is made upper case and non-blank characters are shifted to the left. The resulting text is compared against arguments in the VERB CONVERSION TABLE. If a match is found, the corresponding standard form of the command is substituted.

COMMAND EDIT AND BREAK OUT - The information in the HASP Console Message Buffer is moved to the COMMAND field in the PCE work area. The two bytes CMBFLAGS and CMBCONS of the buffer are moved to the COMFLAGS and COMROUTE fields of the PCE workarea. These two bytes when combined with the two succeeding bytes in the PCE 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. Solid characters are ORed (moved with upper casing) into the COMMAND area. Blanks encountered in the buffer 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 buffer is released, 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 COMMULOP field (see Figures 4.5.1 to 4.5.3). 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 SUB-PROCESSOR - The SELECTION TABLE is used to determine the appropriate command sub-processor 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 $COMTAB macro in Section 4.5.4 for format of the SELECTION TABLE element.)

VALIDATING THE SOURCE AND ENTERING THE SUB-PROCESSOR - 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 restriction 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 local console support routines will contain the restriction flags assigned to the Console Device Control Table. (Restriction is the opposite of authority which is set by the operator command $TCONn,A=authority or by the system programmer.)
3. 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. Otherwise Register 1 is set with the value in the SELECTION TABLE entry COMTOFF field and control is passed to the CSECT indicated by the Overlay Constant ($OCON) field of the SELECTION TABLE element via the $XCTL macro.

Command Sub-Processor Control Sections

The Entry routine of each command sub-processor 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 sub-processor. Normally the sub-processor is entered directly by the special Command Processor macro: "Branch Relative Register" on R1 ($BRR R1). However, some control section entry routines will pre-process the operands of the command prior to entering the sub-processor. Each sub-processor performs the desired functions and returns to the main command processor for the next command.

4.5.3 HASP Command Processor Organization

The HASP Command Processor is created by a single assembly with multiple Control Sections (CSECT). The main CSECT HASPCOMM is the only portion of the Command Processor that is part of the HASP resident load module. It contains all V type address constants required by the sub-command processors and all "BASE2" service routines. The Command Edit Routine HASPCOME receives control from the main processor and determines which COMMAND SUB-PROCESSOR CSECT to enter for processing of the command entered. One or more of the various COMMAND SUB-PROCESSOR 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 grouping is as follows:

JOB QUEUE COMMANDS
JOB LIST COMMANDS
MISCELLANEOUS JOB COMMANDS
DEVICE LIST COMMANDS
SYSTEM COMMANDS
MISCELLANEOUS DISPLAY COMMANDS
REMOTE JOB ENTRY COMMANDS

HASP Command Processor Workarea

The HASP Command Processor PCE workarea shown in Figure 4.5.1 is the primary workarea for the processor and is the only area which may be used to save information in the event a $WAIT is issued by the processor or any of the "BASE1" service routines on behalf of the processor. The fields are generally used as described in the following paragraphs.

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 resident portion 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 is inserted into the COMFLAGS and COMROUTE cells and are used to 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 workarea and 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
$CFJDCT	address of HASP job queue element for requested job
$CFJMSG	same as $CFCVE

COMDWORK - This field is aligned on a double word boundary and is used as a workarea 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 sub-processor is entered. The command is overlayed by the reply message text for all $WTO messages issued by any Command Processor routine. Some command sub-processors use the area as scratch areas and in some cases the right end for storage of critical information while message replies are generated in the left end of the area.

COMPNTER-COMNULOP - 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 non-existant or "null" operand. Operand 1 through n pointers are right adjusted in COMPNTER so that operand n pointer is adjacent to the "null" pointer (see Figures 4.5.2 and 4.5.3 for illustrations). Command sub-processors use these areas for additional workspace after the operand pointers are no longer needed. Examples of other uses are listed as follows:

1. Job queue command $DN and $DQ commands place queue scanning control elements in the COMPNTER area.
2. Job list commands place the job range numbers (j-jj) in the corresponding operand pointer element area.
3. $DR uses the right end of the COMMAND area and COMPNTER-COMNULOP area to hold the reply ID numbers.

Figure 4.5.1 -- HASP COMMAND PCE WORK AREA FORMAT

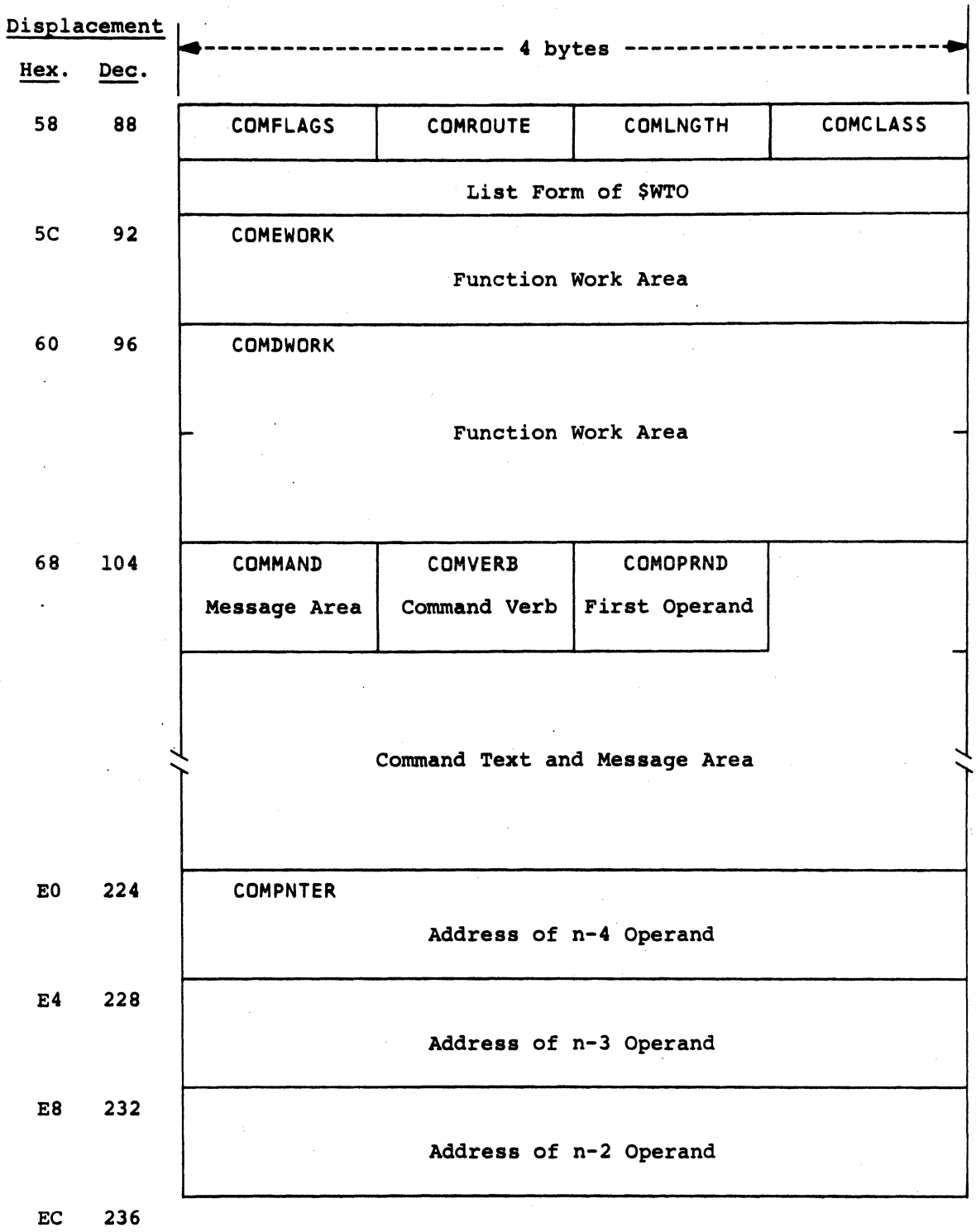


Figure 4.5.1 -- HASP COMMAND PCE WORK AREA FORMAT (CONTINUED)

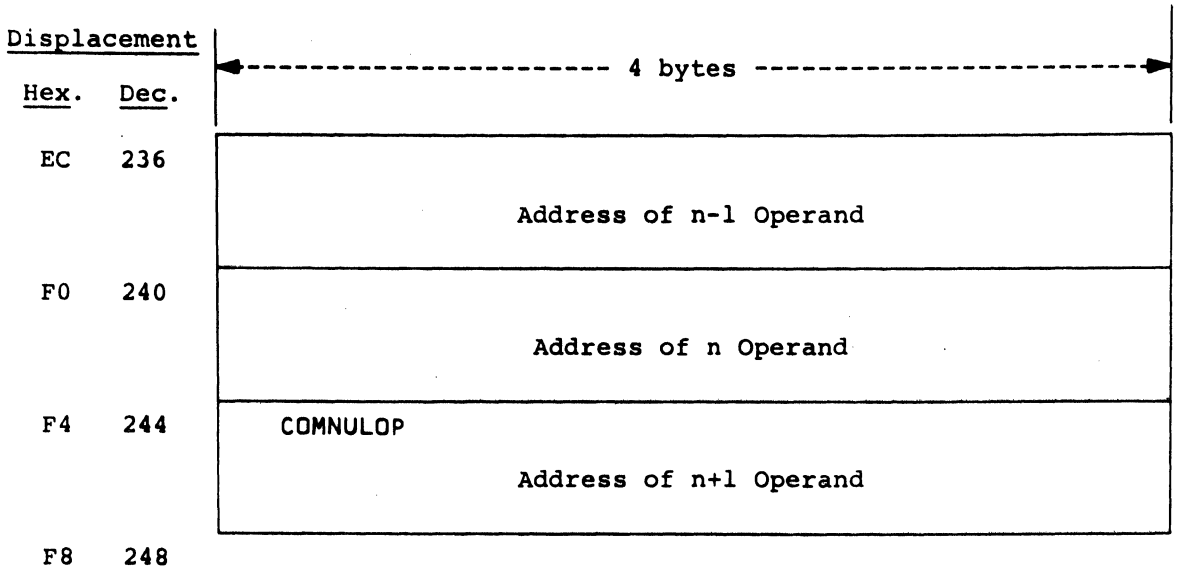
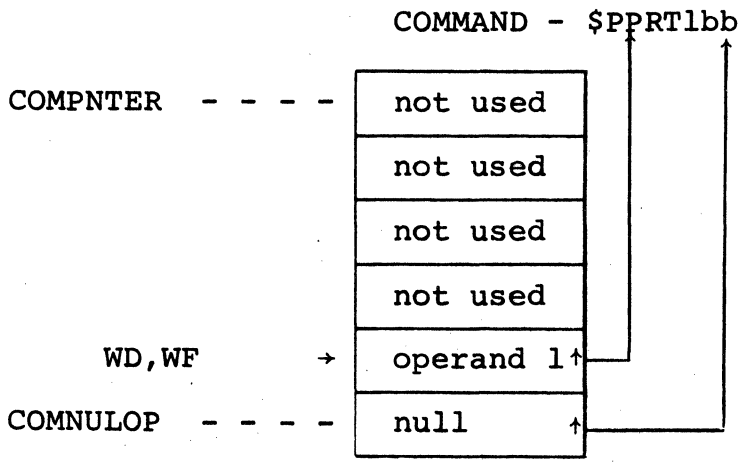


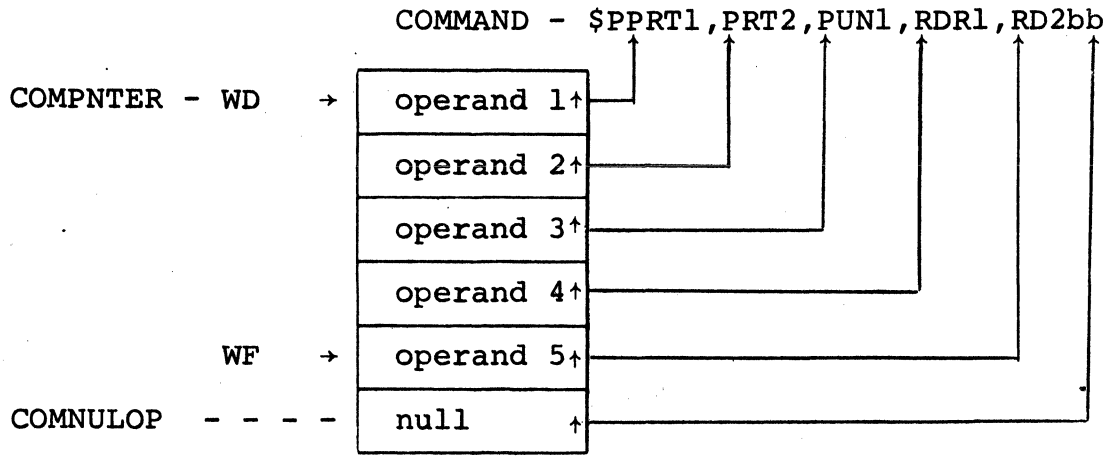
Figure 4.5.2 COMMAND - COMNULOP Areas With Single Operand Command



Upon exit of Edit Routine, Registers WD, WE and WF are set for testing instructions:

BXLE WD,WE,loop
(for next operand)
BXH WD,WE,exit
(if no more)

Figure 4.5.3 COMPNTER -COMNULOP Areas With Five Operand Commands



NOTE: b = blank character

Upon exit of Edit Routine, Registers WD,WE and WF are set for testing instructions:

BXLE WD,WE,loop
(for next operand)
BXH WD,WE,exit
(if no more)

Coding Conventions

The symbols with the command processor conform to the following conventions:

1. All main processor, Edit Routine, and PCE workarea symbols start with the characters "COM".

2. All Function macro generated symbols start with "COF".

3. All command sub-processors 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 (CDNxxxxx 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 sub-processor. At the point 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 appears within the group, the value of register R1 will be set by the $COMGRUP entry routine to a value so that a $BRR R1 will enter the command sub-processor.

4.5.4 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 obtainable from the HASP generation and assembly process in assisting the user to understand the generated code as specifically used in the current HASP as distributed.

Each HASP Command Processor macro may be dependent upon the definitions contained within the Command Processor source deck 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 links 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 macros which are supplied under each category are summarized in Table 4.5.4. 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.

Table 4.5.4 Command Processor Macro Summary

Op-Code	Definition
ORGANIZATIONAL:	
$COMWORK	COMMAND PROCESSOR WORKAREA (symbolic definitions)
$COMGRUP	DEFINE GROUP OF COMMAND SUB-PROCESSORS
$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
$CFJMSG	DISPLAY JOB INFORMATION MESSAGE
$CFJSCAN	SCAN JOB QUEUE ASSISTANCE
$CFSEL	SELECT A ROUTINE BASED ON CHARACTER
$CFVQE	VERIFY CONSOLE CONTROL OVER JOB
RELOCATIBILITY AIDS:	
$ARR	ADD RELATIVE REGISTER
$BRR	BRANCH RELATIVE REGISTER
$SRR	SUBTRACT RELATIVE REGISTER

Organizational Macros

$COMWORK - COMMAND PROCESSOR WORKAREA (symbolic definitions)
This macro adds to the PCEDSECT definitions for fields located in the Command Processor PCE workarea. Additional symbolic constants for BASE2 services and some externally defined parameters are defined.

$COMGRUP - DEFINE GROUP OF COMMAND SUB-PROCESSORS
This macro defines the Command Processor overlay control section via the $OVERLAY macro. It provides an optional entry point routine which locates the command sub-processor 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 sub-processor located within the group. Example:

specification	command	sub-processor 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 sub-processor 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 - The 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 - The exact characters used in the specification in the symbol field of the appropriate $COMGRUP macro statement.

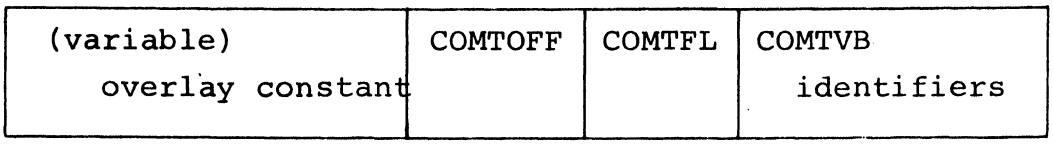
REJECT= - The command source rejection mask. One or more of the following symbols may be specified as follows:

- "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"

Figure 4.5.5 - Selection Table Element



COMTOFF = Offset for the overlay control section to locate the command sub-processor entry point.

COMTFL = Rejection flags.

COMTVB = Command identification characters. Verb with:

1. First character of the first operand.

2. X'FF'
   If X'FF' is specified all commands which have not been specified by the previous entries in the table will be considered "selected".

BASE2 Services

$CRET - RETURN TO MAIN COMMAND PROCESSOR

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

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 for the ability of the author of the command sub-processor 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:

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

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 the last is smaller than the previous).

POINTER=(R1) - The address of the COMPNTR field which addresses the operand containing one or more numerical values separated by dash (-).

NUM=2 - return 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) - The address of the DCT to display.

$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) - The 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 "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 procesor 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 the instruction plus 4 (NSI+4).
REGISTERS USED: R1, LINK, R15

JOBQE=(R1) - The address of the HASP job queue entry for the desired job.

$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) - The address of the HASP job queue entry for the desired job.

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

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

JOB= - This parameter 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= - This parameter may be ignored by the macro; however, if specified as "JOB=Q" all jobs given to the macro expansion are queued (not active) or if specified as "JOB=A" all jobs given to the expansion are active.

$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 whatever 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 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 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 prohibited.

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

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

EMPTY= - The name of the user exit routine desired to be entered when the job queue is found to be 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" sub-parameter specifies an argument to compare against. The corresponding address sub-parameter indicates the address of the desired routine to enter if the character matches the argument. Register form is prohibited.

OPERAND=(R1) - The address of the designated input character to examine.

$CFVQE - VERIFY CONSOLE CONTROL OVER JOB
This macro tests 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) - The address of the HASP job queue entry for the desired job.

OK= - Address of the routine desired 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 desired 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 - The symbolic register containing the address to be made true.

$BRR - BRANCH RELATIVE REGISTER
This macro instruction is used in conjunction with $COMGRUP to enter a sub-processor routine using the offset provided by the $COMGRUP routine.

condition - Condition required to be met in order to 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 - The 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 - The symbolic register containing the address to be made relative.

4.6 OPERATOR CONSOLE ATTENTION PROCESSOR

This processor is included in HASP only if the value of the HASPGEN variable &NUMCONS is greater than 0 (see Section 7.1). The HASP interface to OS Console Support if &NUMCONS=0, is described in Appendix 12.15.

4.6.1 Operator Console Attention Processor - General Description

The function of this processor is to stage a read on a console whenever an attention is received from that console.

4.6.2 Operator Console Attention Processor - Program Logic

During HASP initialization, the first three words of the OS Console Attention Routine (IEEBA1) are overlayed with instructions which cause IOS to enter the HASPATTN routine of this processor whenever an attention interrupt occurs.

When an attention request is signalled by a console device, HASPATTN saves the device address in the processor's PCE workarea, $POSTs the PCE, and POSTs HASP.

When the Attention Processor is dispatched, it locates the physical console whose address is in the processor's PCE workarea and links to the $WTO Processing Routine (see Section 5.7) to queue a read on that console.

4.7 CHECKPOINT PROCESSOR

4.7.1 CHECKPOINT PROCESSOR - GENERAL DESCRIPTION

The purpose of this processor is to write the necessary information onto disk to affect a subsequent restart of the system. This processor will write the information at a predefined time increment and at the completion of each stage of each job.

4.7.2 CHECKPOINT PROCESSOR - PROGRAM LOGIC

The first entry into the Checkpoint Processor is into a section which 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.

The information to be checkpointed consists of the Job Queue which contains the status of each job in the system, the track allocation map which indicates the track groups of each disk that have been assigned, a save area which contains added information as to the status of the system, the print checkpoint table which is used to effect a warm start of the jobs being printed, and (if generated) the Job Information Table which contains additional information concerning each job in the system. The Job Queue and the Job Information Table reside within the checkpoint buffers, but the remaining fields must be moved into these buffers.

The track allocation map is the first to be moved and the track groups that have been reserved for the jobs that are currently executing and reading in are returned to the track allocation map to avoid loss of tracks in case of an emergency restart. Next the write buffers are completed by moving the save area and the print checkpoint tables. An $EXCP is issued to write the checkpoint buffers and a $WAIT on I/O is initiated.

The Job Information Table (if generated) is written with CCW's which are chained to the CCW's used to write the rest of the checkpoint information. The Job Information Table is not written with each checkpoint but only when the processor which requests the checkpoint indicates that he wishes the JIT to be written. This indication is made by setting the "JITJCKPT" bit in the "$JITSTAT" field to one.

At the completion of the I/O operation, the HASP ECB is posted and the timer is reset to a predefined time increment that was specified as a HASPGEN parameter. A test is now executed to determine if the previous write was successful and if so, a $WAIT macro-instruction is issued to place the processor into an inactive state until the time increment has expired or a stage of a job is completed.

If the previous write was unsuccessful, a message is issued to indicate to the operator that a restart is needed and a permanent HASP $WAIT state is entered so that no further checkpoint will be attempted.

4.8 ASYNCHRONOUS INPUT/OUTPUT PROCESSOR

4.8.1 ASYNCHRONOUS INPUT/OUTPUT PROCESSOR - GENERAL DESCRIPTION

Since the completion of all HASP I/O operations are signalled 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, then, of the Asynchronous Input/Output Processor ($ASYNC) is to, at non-interrupt time, notify all processors of their I/O completions which were indicated by the OS I/O supervisor at interrupt time.

4.8.2 ASYNCHRONOUS INPUT/OUTPUT PROCESSOR - PROGRAM LOGIC

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 Device Control Table entry (DCT) associated with this buffer is located and the active I/O count for the device is reduced by one. 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 by the processor.

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.

4.9 HASP LOG PROCESSOR

4.9.1 HASP Log Processor - General Description

The function of the HASP Log Processor is to construct output buffers for eventual processing as part of each Job's printed output. Input to the Log Processor is through a queue of CMBs associated with the queue pointer $LOGQUE which is defined in the HCT. The nature of the information in the input queue, and consequently the printed output, varies as a function of the HASPGEN Parameters &NUMCONS and &WTLOPT.

4.9.2 HASP Log Processor - Program Logic

Log processing of a message buffer is started by locating the corresponding execution PCE. PCEs for output buffers are found by using the job number in the buffer, and "reply" message PCEs are located by using the TCB address which is placed into bytes six through eight of the buffer by the Operator Console Input/ Output Processor's asynchronous exit. Reply message processing is valid only for &NUMCONS>0.

A test is made to ascertain if the message will fit in the HASP buffer currently being used by the job for log output. If space is available, the message is placed in the HASP buffer and the CMB is processed as follows: If the CMB status bits indicate a "read" or a "log only" condition, then the CMB is returned to the free queue via the routine $FREEMSG. The "log only" condition is used when &NUMCONS=0. "Read" and "Write" have meaning only when &NUMCONS>0. If the status bits indicate a "write" condition, then the CMB is queued for display via the $WQUEBUF subroutine.

4.10 OPERATOR CONSOLE INPUT/OUTPUT PROCESSOR

This processor and associated routines are included in HASP only if the value of &NUMCONS is greater than 0 (see Section 7). The HASP interface to the OS console support which is included if &NUMCONS=0, is described in Appendix 12.15.

4.10.1 Operator Console Input/Output Processor - General Description

The function of the Operator Console Input/Output Processor is to process all I/O activity on all operator consoles. The processor also processes all console errors, making a number of retries. If the error continues, the message is ignored.

4.10.2 Operator Console Input/Output Processor - Program Logic

The Operator Console Input/Output Processor examines each entry in the console message buffer I/O queue, $BUSYQUE. Each bit in the console byte is tested for an available console. If one is found the appropriate operation is initiated with a $EXCP macro-instruction and testing of the queue is resumed. When all available consoles have been processed, the processor enters a $WAIT condition until an I/O interrupt is received on one of the consoles, or until another console message is added to the queue.

4.10.3 Operator Console Input/Output Appendage - Program Logic

The Operator Console Input/Output Processor's asynchronous exit is entered from the Asynchronous Post Processor following the completion of an I/O operation on a console device. The IOB completion code is tested for abnormal end, and if an error exists, an error routine is entered to retry the operation.

If the completion is normal the appropriate physical console bit is shut off and the console byte is tested to see if the operation is complete on all consoles. If any bits are still on the Operator Console Input/Output Processor is $POSTed and an exit is taken.

If all bits are now off and the operation code is a write, a link is made to $FREEMSG, the Input/Output Processor is $POSTed and an exit is taken.

If the completed operation is a read, the response is processed according to type. If the buffer contains a HASP command (i.e., an input message whose first character is a dollar sign ($)), it is chained to the end of a queue for the Command Processor ($COMMQUE), the processor is $POSTed, and an exit is made with a $POST of the Input/Output Processor.

If the message is a "reply", the reply number is converted to binary and the corresponding entry in $WTORQUE is located. Using the information in the entry, the message is moved to the WTOR's reply area and the WTOR's ECB is POSTed. The reply queue entry is merged into the free queue ($WTORFRE), and a link is made to the Log Queuing Routine. The Input/Output Processor is $POSTed and exit is made.

If the message is not a "reply" or a HASP command, it is assumed to be an OS command. The message buffer is set to the proper format for the Master Command Routine and an SVC 34 is issued. When control is returned from the Master Command Routine, the buffer is released, the Input/Output Processor is $POSTed and an exit is made.

4.11 TIMER PROCESSOR

4.11.1 TIMER PROCESSOR - GENERAL DESCRIPTION

The function of this processor is to reset the OS interval timer after a timer interrupt has occurred.

4.11.2 TIMER PROCESSOR - PROGRAM LOGIC

This processor calls the IPOSTIT and ISETINT subroutines in the $STIMER/$TTIMER Control Service Routine (see Section 5.6), 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.

4.12 REMOTE TERMINAL PROCESSOR (360/20-STR)

4.12.1 Remote Terminal Processor (360/20) - General Description

The Remote Terminal Processor (RTP), although not a part of HASP proper, can be considered in the same catagory as other HASP processors. RTP is created by HASPGEN to operate as an extension of HASP on a System 360 Model 20 used as a remote terminal to HASP. RTP, in the Model 20, maintains constant communications with HASP at the central computer site via several classes of telephone lines to 1) encode and transmit jobs submitted at the remote site to HASP for execution on the central computer, and 2) print and/or punch the output from jobs thus submitted as the output becomes available. Various techniques are utilized by RTP and HASP to obtain maximum performance of both the Model 20 devices and the communication lines used. RTP currently requires an 8K Model 20 with any reader and printer attached. The program can be made to operate in a 4K environment at a somewhat degraded performance with reduced ease of operation.

RTP has been designed to allow the addition of "background" functions to operate in a multiprogrammed environment with normal remote terminal processing.

4.12.2 Remote Terminal Processor (STR Model 20) - Program Logic

Upon completion of the loading of the RTP program deck, control is transferred to the initialization phase of the program to prepare for job processing. Initialization first checks the card reader for the presence of patch (REP) cards and, if present, makes the appropriate patches (the RTP REP card format is identical to the HASP REP card as described in Section 6.4). Encountering a /*SIGNON card within the REP cards, will cause initialization to replace the default remote SIGN-ON identification and password by the contents of the card. After loading REPs, or if no REP cards are present, the dynamic configuration card (which follows REPs if present) is decoded and appropriate commands for the system punch selected are established. (The formats of the SIGN-ON and dynamic configuration card are given in the Model 20 Operator's Guide-Section 11.2). The final process of initialization is the dynamic construction of the buffer pool. Buffers are built, according to the HASPGEN parameter &TPBFSIZ until the memory size of the machine is reached or the assembly parameter &NUMBUFS is reached. Construction of the buffer pool overlays the complete initialization routine. Control is then passed to the processing section of RTP.

The processing phase of the program consists of four principal processors and a communications adapter (CA) I/O supervisor. Allocation of CPU time to the various processors is accomplished via a commutator. A processor is entered into contention for CPU time by changing its commutator entry from a NOP to a BRANCH command. Through the use of the WAIT macro, a processor may await the occurrence of a certain event and be entered, via the commutator, below the wait instruction upon completion of the event.

PROCESSORS

Card Read Processor

Upon initial entry, this processor checks the system card reader for ready status. If the device is not ready, HASP is notified, via a SEND EOT, of the lack of jobs to transmit, the CA receive processor is activated, the card read processor is deactivated, and entry is made to the commutator. If the card reader was ready, the transmission phase is immediately begun. Cards are read (double buffered) and are passed to the ENCODE subroutine which compresses and translates the card for transmission. The encoded card images are blocked in a buffer obtained from the dynamic buffer pool until the capacity of the buffer is reached. The buffer is then chained into a queue of buffers awaiting transmission by the CA transmission processor to HASP in the central computer. Another buffer, if available, is obtained from the buffer pool and is processed in a like manner. When, and if, the supply of buffers is exhausted, the reader processor enters a WAIT state to await the freeing of a transmitted buffer by the CA transmission processor. When the last card of the job stack has been read, a SEND EOT (zero word count buffer) is queued for transmission and the steps described previously are done to terminate transmission and activate reception. In order to

minimize CPU utilization, the card read processor-compression routine only compresses "n" or more blank characters (where "n" is the value of the assembly parameter &CCT). The format of transmission records to HASP is described in Section 12.9.3.

Communications Adapter Transmission Processor

The CA Transmission processor removes buffers from an ordered queue, dynamically being built by the Card Read Processor, and transmits their contents to HASP in the central computer. All transmissions are via the Communications Adapter-I/O Supervisor (CAIOS) which provides for line re-instruct at interrupt time to make optimum use of the line (See CAIOS description). As posting of successfully completed writes occurs, the buffers are returned to the free buffer chain for reuse by another processor. This processor continues to dequeue and transmit buffers, as they become available, until a buffer with a transmission word count of zero is encountered. An EOT is then sent to HASP to indicate the end of the input stream, the CA Transmission Processor is deactivated and return is made to the commutator.

Communications Adapter-Receive Processor

The CA Receive Processor is activated by the Card Read Processor when it is determined that no jobs are available to transmit to the central computer. Upon being entered, CA Receive establishes communication with HASP in the central computer to await the output of a previously submitted job. The lack of jobs to transmit is indicated by HASP with an immediate EOT signal to the Model 20. When this EOT is received, the CA Receive Processor deactivates itself and activates the Card Read Processor to again check for the presence of jobs to send to HASP.

If a job is avialable to be printed or punched, the CA Receive Processor activates the Print/Punch processor and immediately begins reading transmittal records into buffers obtained from the dynamic buffer pool. Buffers, thus filled, are placed in an ordered queue to await processing by the Print/Punch Processor. All CA reads are via the Communications Adapter I/O Supervisor (CAIOS) which provides for line re-instruct at interrupt time to make optimum use of the line (see CAIOS description). Processing continues, as buffers and/or transmittal records become available, until an EOT signal is received from HASP indicating end-of-job. A buffer with a word count of zero is added to the queue to inform the Print/Punch processor of the end-of-job.

Communication is then, once again, established with HASP to ascertain if additional output for this job is available (i.e. the punch output of the job which has just completed printing). After the additional output has been processed, or if none existed, the CA Receive Processor is deactivated, the Card Read Processor is activated, and return is made to the commutator. Note that this logic, of activating the Card Read Processor prior to beginning processing output from the next job, allows the Model 20 Operator to interrupt print/punch processing, at a job boundary, to transmit a job to the central computer.

Print/Punch Processor

When activated, the Print/Punch Processor begins dequeuing and processing buffers from the queue (being) created by the CA Receive Processor. Records to be punched are indicated by "carriage control" characters of X'0F0F' and are routed to the punch section of the processor. In order to minimize CPU requirements, the print processor does not provide for 1-7/8 encoding of print characters (see Section 12.9). The 16 4 of 8 characters normally reserved for 1-7/8 encoding are redefined for print records only, as additional print characters, thus yielding a 64 character print set.

After reconstructing and printing or punching all records in a buffer, that buffer is returned to the buffer pool for use by another processor. When a buffer with a zero word count is encountered in the queue (indicating end-of-job), the Print/Punch Processor is deactivated, unless records from the next job have already been queued, and return is made to the commutator.

If a dynamic configuration card described the system punch unit as DUMMY, the punch section of the processor is dynamically altered (by initialization) to immediately free all punch buffer encountered in the Print/Punch buffer queue. This results in eliminating punched output; however, punch records are still transmitted to the Model 20.

By setting the assembly parameter &PUNCH to 0, all code concerned with processing punched output will be eliminated from RTP. The appropriate HASPGEN must be done on the central system to force all punch output for a "punchless" terminal to be processed locally.

Communications Adapter I/O Supervisor

The primary purpose of CAIOS is to assure the maximum possible communication line utilization by re-instructing the line at the earliest possible moment after completion of a previous generation.

All requests to read and/or write the communication line are passed to CAIOS for execution by the CA processors. Upon receipt of an I/O request, CAIOS immediately initiates the operation if the line is dormant, or queues the request to await completion of the currently active operation.

The completion of a CA I/O operation causes an interrupt which immediately transfers control to CAIOS. If the operation indicated as complete by the interrupt was successful (error free), any queued I/O request is immediately initiated. The Event Control Block of the requestor of the just completed I/O operation is posted (with a X'7F') to indicate the successful completion of the request. Return is then made to the interrupted processor. CAIOS recognizes and attempts to correct all transmission errors encountered on any CA I/O operation. Since both CA processors are designed to double buffer I/O requests, CAIOS insures virtually total line utilization during transmission periods.

4.12.3 Remote Terminal Processor (360/20)-Assembly Parameters

The following indicates the variable name and function of certain RTP assembly parameters which can be of general use.

&TPBFSIZ - defines the size of the buffers used for transmission to and from the HASP system. (Since this variable must exactly agree with the corresponding variables in the HASP system, the values of both are automatically set at HASPGEN time.

&NUMBUFS - limits the number of CA buffers created dynamically at initialization time. Initialization will create buffers until the capacity of memory, or the value of &NUMBUFS is reached. It is suggested that this value be made large enough to allow sufficient buffering (hence line load-leveling) to occur.

&CCT - represents the minimum number of consecutive blank characters which will be compressed by the Card Read Processor. This value should never be less than 4 and, significantly

reduces CPU requirements by the Card Read Processor as it is increased. A value of 80 will effectively prevent all blank compression (except on totally blank cards). The value of &CCT may never exceed 80.

&PUNCH - controls the existance of code within RTP to process punch output received from HASP.

If &PUNCH=1, punching capabilities will exist in RTP.

&PUNCH=0, no punching capabilities will be created in RTP (NOTE: the HASPGEN of the central computer system must agree with this option.)

&MACHINE - defines the Model of SYSTEM/360 on which RTP is to operate. This value must presently be set to 20. This option can subsequently be used to assemble RTP, at HASPGEN time, for any Model of SYSTEM/360 being utilized as a HASP remote terminal. Although certain parts of this feature are currently in RTP, it is incomplete and totally untested.

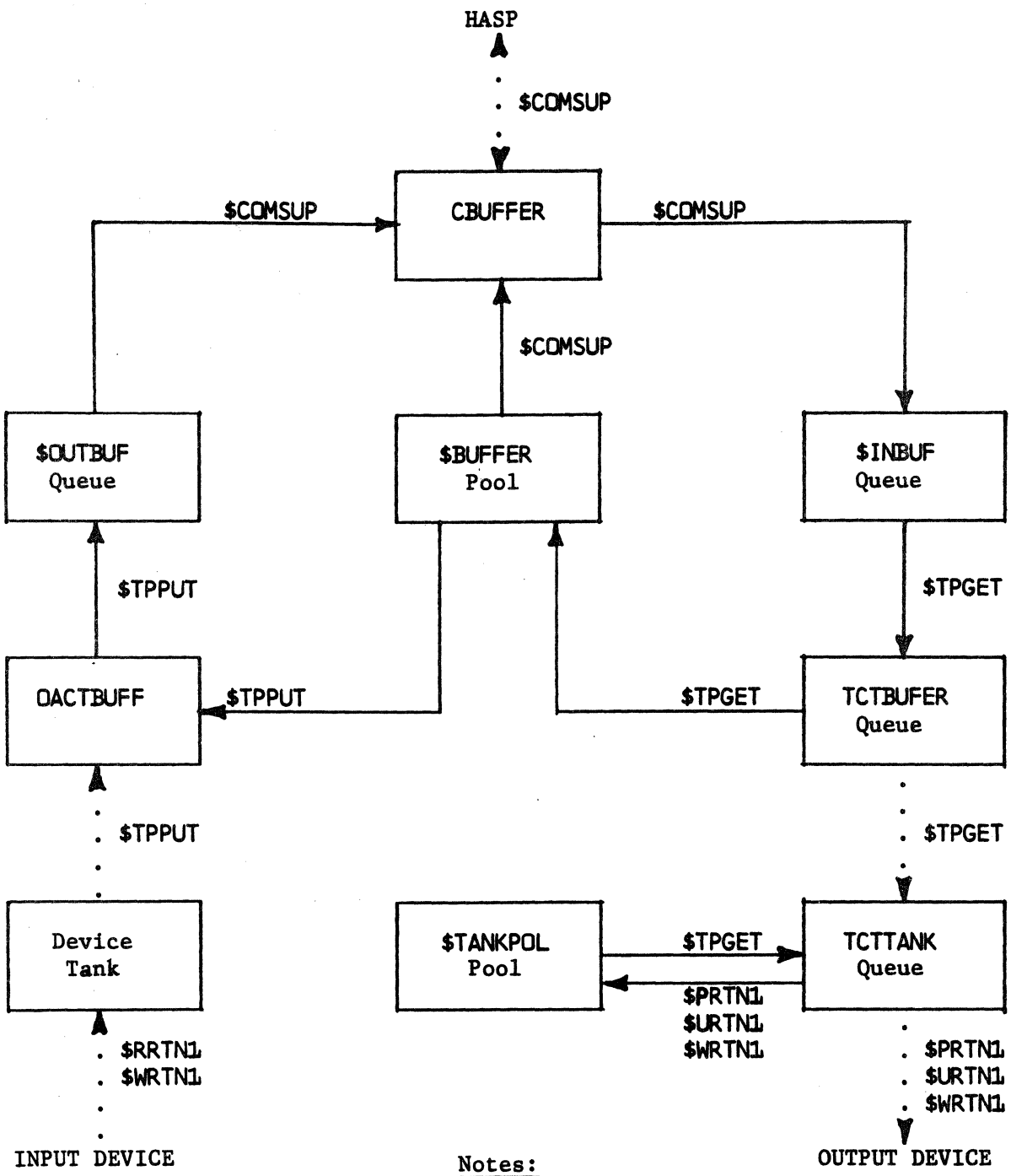
4.13 REMOTE TERMINAL PROCESSOR (SYSTEM/360-BSC)

The following sections outline the basic logic flow of the MULTI-LEAVING Remote Terminal Processor program for System/360 (including Model 20) workstations utilizing Binary Synchronous communications devices. The same workstation program is utilized for both the Model 20 and System/360 workstations with generation parameters for the machine type.

4.13.1 General Description

The MULTI-LEAVING Remote Terminal Processor program is created by HASPGEN to operate as an extension of HASP on any Model of SYSTEM/360 used as a remote workstation for HASP. This terminal program maintains constant communications with HASP at the central site via several classes of telephone lines to (1) encode and transmit jobs submitted at the remote site for OS/360 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 workstation program to obtain maximum performance of the remote devices and the communications line. Figure 4.13.1 indicates the basic information flow through the system.

Figure 4.13.1 MULTI-LEAVING Information Flow Diagram



Notes:

Solid lines indicate buffer or decompression tank flow with or without data.

Broken lines indicate data flow only.

Line comments indicate processor responsible.

4.13.2 Program Logic

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 command. A single control block, the Total Control Table (TCT) is utilized by all processors to provide for synchronization of concurrent operations, processor status information,re-enterability and both inter and intra processor 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 add the encoded record to its current output buffer. When the current buffer is filled or terminated, 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 route the decompressed records to the appropriate 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 responsible for metering the flow of each type of record from HASP. This also 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 workstation 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 insure full bi-directional capabilities and to avoid unprogrammed "time-outs" of the adapter.

In addition COMSUP maintains, verifies and corrects (if necessary) the MULTI-LEAVING block sequence checking feature and detects, logs and retries all communications errors.

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 CPU's.
5. The communication adapter is enabled and communications 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 PSW's 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 dequeuing decompression tanks containing print information from the printer Total Control Table, examining the sub-record 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 dequeuing 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 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 which sends a request to send a job stream to HASP. When permission is received the job stream submission routine is entered which reads 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; this time tests are made to determine

the existance of cards in the primary feed. The tests are also made in the event of no tanks available for dequeuing. 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 dequeuing 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 upon 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 dequeuing decompression tanks containing print 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 dequeuing 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 output "tank".

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 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, types the message, and passes control to the beginning of the processor. The console read routine tests for operator requests and if so, 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 , posts the appropriate Service Processor and returns. 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 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 sub-record 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 three 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 DSECT's are provided by generation macros in the event 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

NAME	DESCRIPTION
	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 op-code 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

NAME	DESCRIPTION
	compressed for transmission to HASP
TCTFLAG	CCW flags
TCTOPCOD	Op-code 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

NAME	DESCRIPTION
	device end - busy bit is reset at device end
TCTALTOP	Alternate op-code for DUAL reader/punch devices - processors requiring alternate op-codes have the option of setting the TCTCCW 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 bytes identical to this field will be queued for output on the corresponding device

NAME	DESCRIPTION
TCTSTAT	Status Flags - each bit has one or more meanings which are dependant 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 dequeuing 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

NAME	DESCRIPTION
	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 consoles "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 - two character identification for local command processors
TCTINRCB	Optional field - exists when DUAL devices are attached to the system - identifies the Input

NAME	DESCRIPTION
	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 de-compression into tanks
TCTTNKLM	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 TCTBUFER queue

NAME	DESCRIPTION
	before signalling 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 TCT's have extensions which are used as "tanks" for records which are 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 TCT's with prefix designators. RCT, WCT, and with DUAL devices UCT:

RCTTANK1, RCTTANK2	"Tank" origin and working storage
RCTTRCB1, RCTTRCB2	Input RCB for HASP identification
RCTTSRC1, RCTTSRC2	Sub-record control byte = X'80'
RCTTCT1, RCTTCT2	Count field - length of data portion
RCTTDTA1, RCTTDTA2	Data area - input card or operator command - will be blank for the DUAL 2520 and 1442 while in output status

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

4.14 REMOTE TERMINAL PROGRAMS (1130)

Introduction

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:

- 1442 Reader/Punch or Punch
- 2501 Reader
- 1132 Printer
- 1403 Printer
- Console keyboard/Printer

Programs developed for the 1130 in conjunction with the HASP Remote Job Entry feature are assembled using the OS/360 Assembler. The 1130 instruction set is generated thru the use of macro instructions (See Section 14.4.5) corresponding to the actual 1130 hardware commands. Additionally, pseudo (assembler) operations are available to aid in the development of 1130 programs on the 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 in the following paragraphs:

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 core and then loads the main terminal program (RTP1130) into the lower extent of 1130 core. 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.

4.14.1 Remote Terminal Processor (RTP1130)

Introduction

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 intended to be introductory in nature. The user intending to modify the system should use the documentation in conjunction with a program listing which 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:

- 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).
- A long form branch to the active commutator main routine used if the gate is open.
- A named return point for reference by the main commutator routine.
- 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 processors 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 "top" or 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 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

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 processors.

Control information pertinent to the unique requirements of each data type is provided through the associated UFCB.

TPPUT - Processor For Data Destined For HASP

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

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

A conditionally assembled processor which supports the 1442 - 5, 6 or 7 as a card reader, card reader/punch or as 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 sections of code monitor for a "ready" condition; reads cards for transmission to HASP via TPPUT; processes "end-of-file" communications and provide error recovery. The punch sections of code wait for data to be punched through interrogation of a queue developed by the TPGET processor and provide error recovery and and punch termination procedures.

PRFOT - 1403 Printer Processor

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; processing "end-of-file"

conditions; setting UFCB status information and error recovery are included in this processor.

PRETT - 1132 Printer Processor

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

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: 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 communtator cycle and, if key o is on, stored in location $ENTKEYS. All non-control character input is printed and the card code value stored for investigation at EOF time. If the first character of input is "." (period) then 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:

- Sets the interrupt transfer vectors for RTP1130 operation.
- Dynamically builds the TP buffer pool using the defined extent of 1130 core; the end of the 1130 program and the defined TP buffer size.
- Builds a TP buffer containing the sign-on information processed by RTPLOAD for transmission to HASP.
- Establishes SCA communications with HASP and prepares TPIOX for "sign-on".
- Opens the commutator gates for all SCA and input processors.
- Disconnects initialization from the commutator.
- 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 and 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

Converts a single EBCDIC character to the equivalent console printer Tilt-Rotate code using a table look-up method.

SXPPRNT - EBCDIC To 1403 Printer Code Conversion

Converts a single EBCDIC character to the equivalent 1403 printer 6 bit with parity code using a table look-up method.

SXCPNCH - EBCDIC To Card Code Conversion

Converts a single EBCDIC character to the equivalent 12 bit card code using a table look-up method and conversion algorithm.

STRACE - Trace Machine Registers

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

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: 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

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 re-instruct (read operation) of the communications adapter as soon as possible after the write completion.

DBLOCK - Deblock, Decompress, Convert and Store Data From HASP

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 same format as described under "Output Element (Tank) Description".

The operation of DBLOCK includes the priming 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

Constructs a logical record consisting of a physical input record 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.

- No compression or truncation
- Trailing blank elimination only (truncation)
- 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 must 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 - Constructs TP Buffers

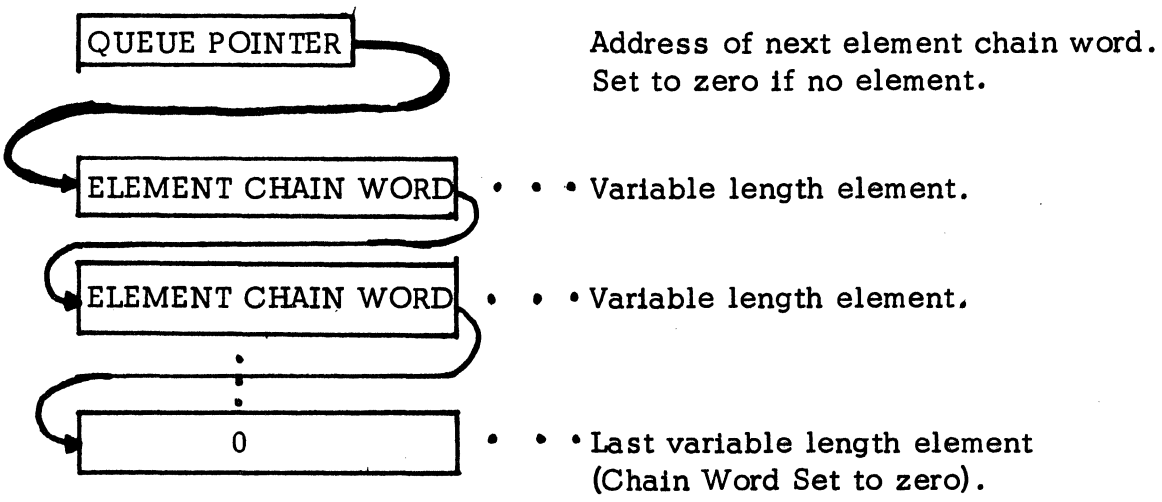
Constructs TP buffers for TPIOX transmission to HASP. Data to be inserted and length of insert are provided by user. TPPUT initializes this routine by providing the buffer to be used and setting pointers and variables.

The data to be inserted is usually in the form 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:



Examples of chained lists are: TP buffers, console message tanks, 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 processors 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 "&TPBUFSZE" 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	Checksum 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	Checksum of columns 1-72
75-76	Identification
77-80	Sequence number

REP Card Format

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 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

RRER L01772 X4C18,X1FF8
```

Would be ignored because columns 2-4 not equal to "REP"

4.14.2 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 in Section 4.14.1 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:

- Clears core from location 0 to "&RTPLORG-1"
- Tests for a 2501 or 1442 card reader and initializes the card read routine for the appropriate device.
- Reads RTP1130 program cards, performing the conversion from card code to EBCDIC and loading the data into the specified locations.
- Sets up the entry to RTP1130 when the end card is processed.
- Reads and processes REP cards, if they exist.
- Reads, converts and stores /*SIGNON and sets indicator for RTP1130 signalling existance if /*SIGNON encountered.
- Transfers control to RTP1130

4.14.3 Remote Terminal Bootstrap (RTPBOOT)

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 core locations defined by "&RTPLORG" at RMTGEN time. RTPBOOT loads into lower 1130 core 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.

Figure 4.14.3 - Remote Terminal Bootstrap Card Format

Card Col.	Card No. 1	Card No. 2	Card No. 3	Card No. 4
1	12-11-7	12	12-11-1-2-3-4-5	12-11-1
2	1-2-9	11-0-3-5	blank	12
3	12-11-1-8	11	5	12-11-2-3-4-5
4	12-11-7-8-9	blank	11-0-1-5	12-11-1
5	11-0-1-6-9	5	4	5
6	0-2-6	11-0-1-5	11-0-1-5	11-0-1-5
7	4-7-8-9	12-11-0-1-2-4-5	0-1-2-3-4	12-11-0-2-3-4-5
8	blank	12-11	11	11-0-1
9	4-6	blank	blank	blank
10	0-1-2	12-11-1-5	12-11-1-4	11-0-3-5
11	blank	5	5	0-3
12	11-2-5	1-2	11-0-1-4	5
13	4-5-9	12-11-0-1-4-5	12-11-0-1-2-3-4-5	blank
14	12-0-1-2-5-6	12-11-1	11-0-1-4-5	12-1-5
15	1-2-8	blank	12-11-0-1-2-4	5
16	12-11-1-3-4-5-6-8-9	12-11-3	11-0-1	12-1-5
17	12-11-3-4-5-6-7	5	12-3-4-5	12-11-2
18	1-2-8-9	blank	12-11	12-11-1
19	12-11-1-3-4-5-6-8	12-11-0-1-2-3-4-5	12-0-3	1-5
20	12-3-4-5-7-9	11-0-1-3	11-0-1	11
21	12-11-1-3-4-5-7	11-2-4-5	blank	12-11-3-4
22	12-11-4-7	blank	12-11-3	12-11-0-1
23	1-6	12-11-3-4-5	12-11-1-2-3	1-2
24	12-11-1-4-8	11-0-1	blank	11-2-3
25	12-4-7-8	2-3-4-5	blank	11-0-2-3-4-5
26	12-11-1-4-7-9	11-0-1-3	11-0-3-4-5	blank
27	12-4-8	11-2-4	2-3-4-5	12-11-0-1-2-3-4-5
28	12-11-1-4-9	blank	4	11-0-1
29	12-11-3-4-6-9	12-11-3	0-2-4	5
30	1-6-9	11-0-1	11	11-0-1-4-5
31	12-11-1-3-4-6	3-5	5	12-11-0-1-2-3-4-5
32	1-2-6	11-0-5	11-0-1	11-0-1-4
33	12-11-1-4-6-7-9	3-4-5	3-4	12-11-0-2
34	12-11-1-5-6-7-8	11-0-1-3	11-0-1	11-0-1
35	12-11-1-5-6-8-9	11-2-4	blank	12-11-0-1-2-3-4-5
36	12-11-1-3-4-8	blank	11-0-3-4-5	11-0-1
37	12-11-1-3-4-7-9	blank	12-11-0-1-5	5
38	2-3-5-6-7-8	11-0-3-4	5	blank
39	2-3-5-6-7-8-9	11-0-2-4-5	0-3-5	blank
40	11-0-1-3-4-5-6-7-8-9	4	11	12-11-0-1-4-5

Figure 4.14.3(CONT) - Remote Terminal Bootstrap Card Format

Card Col.	Card No. 1	Card No. 2	Card No. 3	Card No. 4
41	9	1-3	12-11-0-1-4-5	0
42	2-3-4-8	11-0-1-3	11-0-1	11-2-3
43	12-11-3-5-6-7-8-9	2-3	11-2-3-4-5	12-0-1-3-5
44	12-8-9	blank	11-0-1-3	blank
45	12-11-1-3-5-6-7-9	12-0-1	12-0-2-5	5
46	2-3-5-6-7	11-0-1-4	blank	11-0-1-3
47	11-2-3-4-5-6	12-0-4-5	1	12-0-2-3-4-5
48	9	11-0-2-4	1-2	blank
49	11-0-1-3-4-5-6-7-8-9	12-1-2-3-4	12-11-1-2-3-4-5	blank
50	9	11-0-2-4	12-11-1	12-11-0-1-4-5
51	12-11-6-7-9	11-2-3-4-5	12-0-1-3-4	12
52	12-3-4-5-6-8-9	12-11	11-0-5	11-2-3
53	12-11-1-6-8-9	blank	blank	12-0-2-3-4-5
54	12-11-6	12-11-1-4	12-11-3-5	blank
55	12-3-4-5-6	12-11-0-1-2-3-4-5	0-3-4	11-1
56	12-11-1-8-9	11-0-1-5	5	blank
57	12-3-4-5-7-8	12-0-1-2-5	blank	blank
58	12-11-1-7	11-0-1	11-0-3-4-5	12-11-5
59	3-7	1-2-4-5	0-2-3	2
60	blank	11-0-1-3	5	1
61	1-2-6	11-2-4	blank	5
62	1-2	blank	11-0-3-4	12-11-0-2-5
63	blank	12-0-1-3-4	blank	12
64	1	11-0-1	5	11-2-3
65	blank	blank	1-2	12-0-1-2
66	12-11-7-8-9	11-0-3-5	11	blank
67	12-1-3-4-6-7	12-11-1-2-3-5	1-4-5	blank
68	12-11-1-7-9	blank	11-0-1	11-0-3-5
69	11-2-4	11-3-4	blank	12-11-1-2-3-5
70	11-0-1-3-4-6-7	11	12-11-3	blank
71	2-3-7-9	blank	4-5	12-11-0-1-3-4-5
72	2-3	12-11-1-5	blank	11
73	11-2-3-4-5-6	12	12-11-0-1-2	5
74	4-7-8-9	11-0-3-4	12-1	12-11-1
75	11-0-1-7	12-11-1-2-3-5	blank	blank
76	8	blank	12-11-1-3-5	11-2-3
77	blank	12	0-3-4	blank
78	blank	11-0-3-4-5	5	blank
79	0	0	0	0
80	1	2	3	4

Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

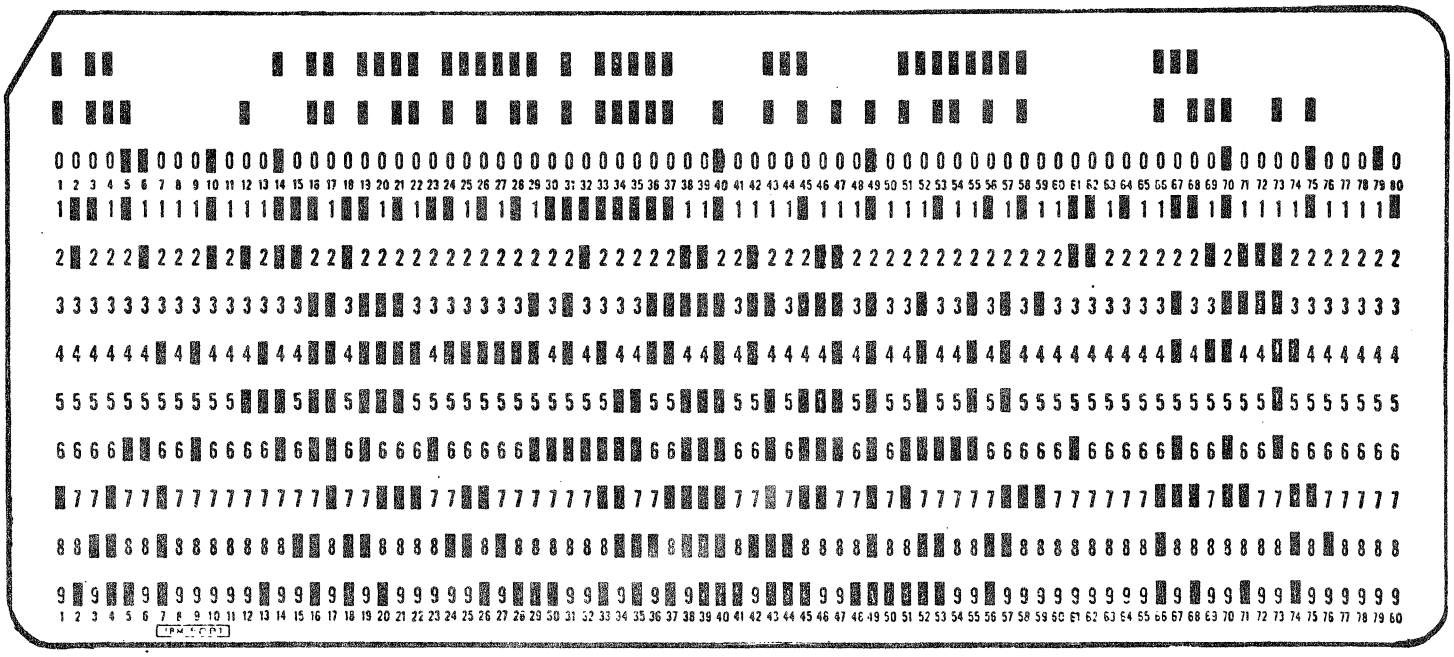
Card Col.	Card No. 5	Card No. 6	Card No. 7	Card No. 8
1	0	11-0-1-8	12	12-0-1-8-9
2	1	11-0-1	12-11-1-9	12-1-9
3	2	11-0-2	12-11-2-9	12-2-9
4	3	11-0-3	12-11-3-9	12-3-9
5	4	11-0-4	12-11-4-9	12-4-9
6	5	11-0-5	12-11-5-9	12-5-9
7	6	11-0-6	12-11-6-9	12-6-9
8	7	11-0-7	12-11-7-9	12-7-9
9	8	11-0-8	12-11-8-9	12-8-9
10	9	11-0-9	11-1-8	12-1-8-9
11	12-11-0-2-8-9	11-0-2-8	11-2-8	12-2-8-9
12	12-11-0-3-8-9	11-0-3-8	11-3-8	12-3-8-9
13	12-11-0-4-8-9	11-0-4-8	11-4-8	12-4-8-9
14	12-11-0-5-8-9	11-0-5-8	11-5-8	12-5-8-9
15	12-11-0-6-8-9	11-0-6-8	11-6-8	12-6-8-9
16	12-11-0-7-8-9	11-0-7-8	11-7-8	12-7-8-9
17	blank	12-11-0-1-8	11	12-11-1-8-9
18	.	12-11-0-1	0-1	11-1-9
19	.	12-11-0-2	11-0-2-9	11-2-9
20	.	12-11-0-3	11-0-3-9	11-3-9
21	.	12-11-0-4	11-0-4-9	11-4-9
22	.	12-11-0-5	11-0-5-9	11-5-9
23	.	12-11-0-6	11-0-6-9	11-6-9
24	.	12-11-0-7	11-0-7-9	11-7-9
25	.	12-11-0-8	11-0-8-9	11-8-9
26	.	12-11-0-9	0-1-8	11-1-8-9
27	.	12-11-0-2-8	12-11	11-2-8-9
28	.	12-11-0-3-8	0-3-8	11-3-8-9
29	.	12-11-0-4-8	0-4-8	11-4-8-9
30	.	12-11-0-5-8	0-5-8	11-5-8-9
31	.	12-11-0-6-8	0-6-8	11-6-8-9
32	.	12-11-0-7-8	0-7-8	11-7-8-9
33	.	12-0	12-11-0	11-0-1-8-9
34	.	12-1	12-11-0-1-9	0-1-9
35	.	12-2	12-11-0-2-9	0-2-9
36	.	12-3	12-11-0-3-9	0-3-9
37	.	12-4	12-11-0-4-9	0-4-9
38	.	12-5	12-11-0-5-9	0-5-9
39	.	12-6	12-11-0-6-9	0-6-9
40	.	12-7	12-11-0-7-9	0-7-9

Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

Card Col.	Card No. 5	Card No. 6	Card No. 7	Card No. 8
41	blank	12-8	12-11-0-8-9	0-8-9
42	.	12-9	1-8	0-1-8-9
43	.	12-0-2-8-9	2-8	0-2-8-9
44	.	12-0-3-8-9	3-8	0-3-8-9
45	.	12-0-4-8-9	4-8	0-4-8-9
46	.	12-0-5-8-9	5-8	0-5-8-9
47	.	12-0-6-8-9	6-8	0-6-8-9
48	.	12-0-7-8-9	7-8	0-7-8-9
49	.	11-0	12-0-1-8	12-11-0-1-8-9
50	.	11-1	12-0-1	1-9
51	.	11-2	12-0-2	2-9
52	.	11-3	12-0-3	3-9
53	.	11-4	12-0-4	4-9
54	.	11-5	12-0-5	5-9
55	.	11-6	12-0-6	6-9
56	.	11-7	12-0-7	7-9
57	.	11-8	12-0-8	8-9
58	.	11-9	12-0-9	1-8-9
59	.	12-11-2-8-9	12-0-2-8	2-8-9
60	.	12-11-3-8-9	12-0-3-8	3-8-9
61	.	12-11-4-8-9	12-0-4-8	4-8-9
62	.	12-11-5-8-9	12-0-5-8	5-8-9
63	.	12-11-6-8-9	12-0-6-8	6-8-9
64	.	12-11-7-8-9	12-0-7-8	7-8-9
65	.	0-2-8	12-11-1-8	blank
66	.	11-0-1-9	12-11-1	12-0-1-9
67	.	0-2	12-11-2	12-0-2-9
68	.	0-3	12-11-3	12-0-3-9
69	.	0-4	12-11-4	12-0-4-9
70	.	0-5	12-11-5	12-0-5-9
71	.	0-6	12-11-6	12-0-6-9
72	.	0-7	12-11-7	12-0-7-9
73	.	0-8	12-11-8	12-0-8-9
74	.	0-9	12-11-9	12-1-8
75	.	11-0-2-8-9	12-11-2-8	12-2-8
76	.	11-0-3-8-9	12-11-3-8	12-3-8
77	.	11-0-4-8-9	12-11-4-8	12-4-8
78	.	11-0-5-8-9	12-11-5-8	12-5-8
79	.	11-0-6-8-9	12-11-6-8	12-6-8
80	blank	11-0-7-8-9	12-11-7-8	12-7-8

Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

CARD 1



CARD 2

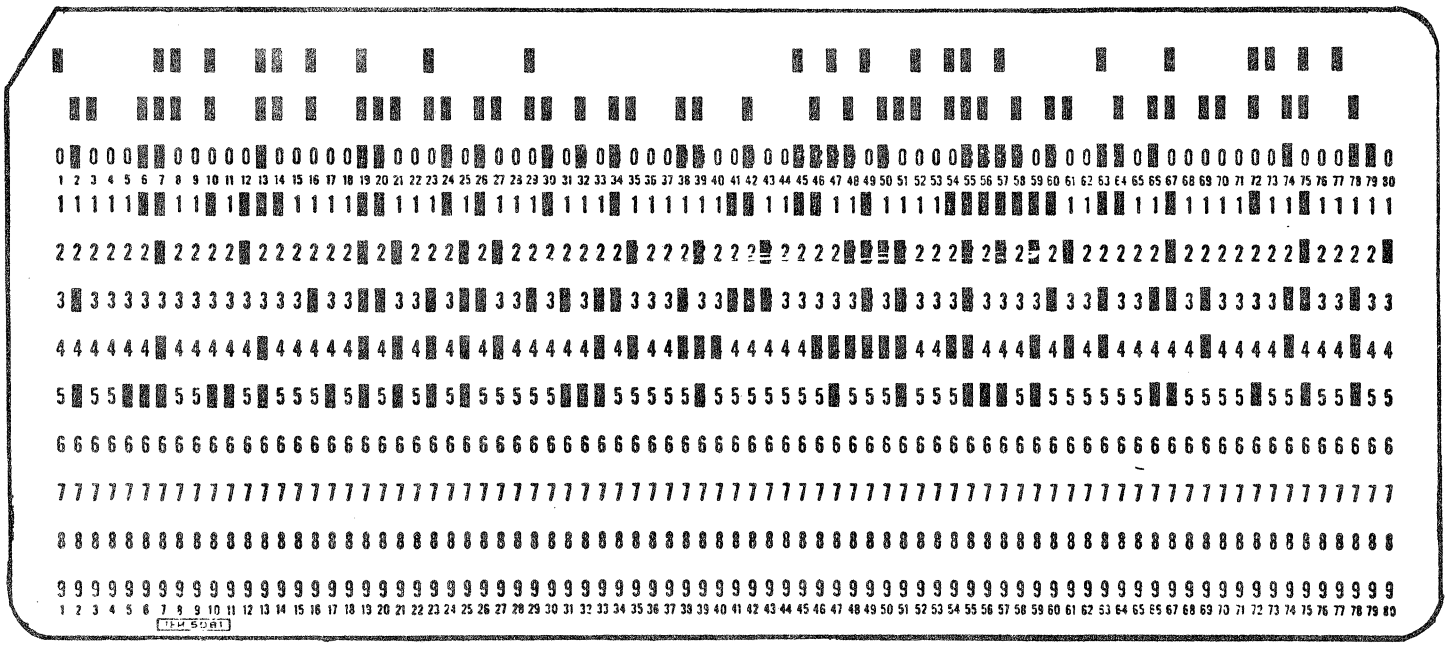
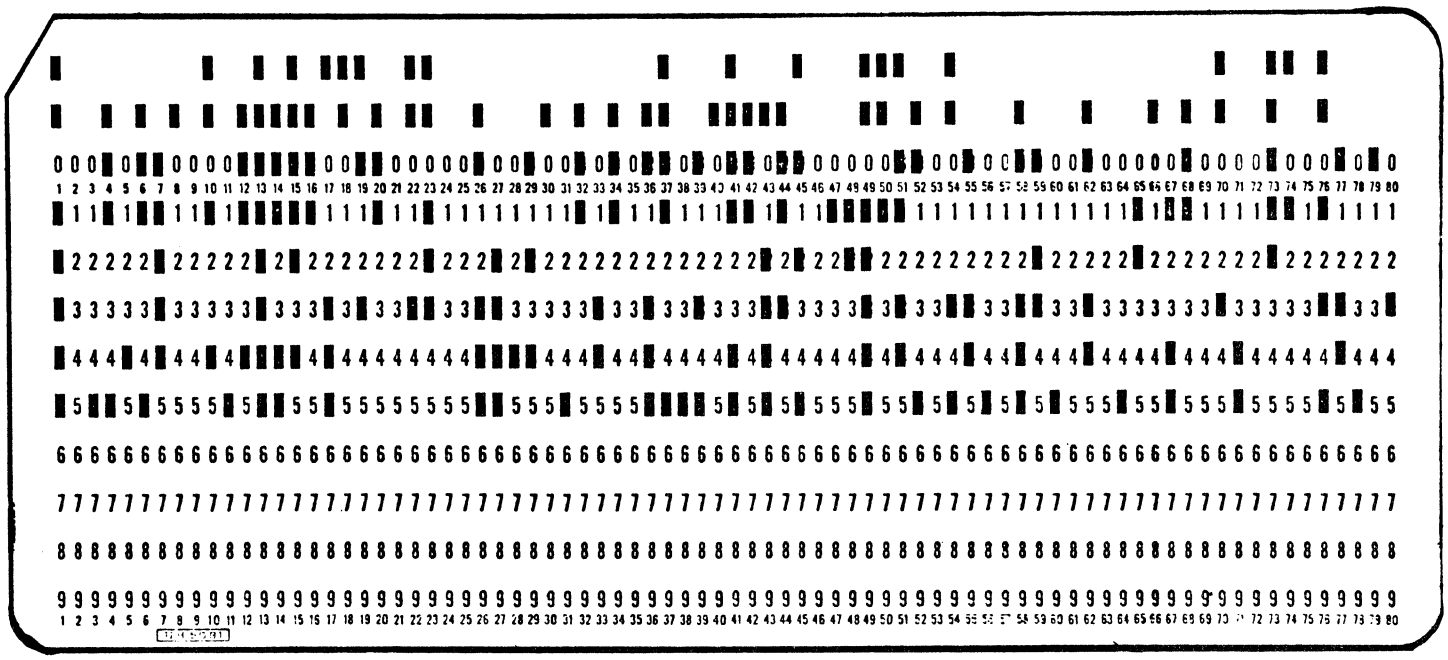


Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

CARD 3



CARD 4

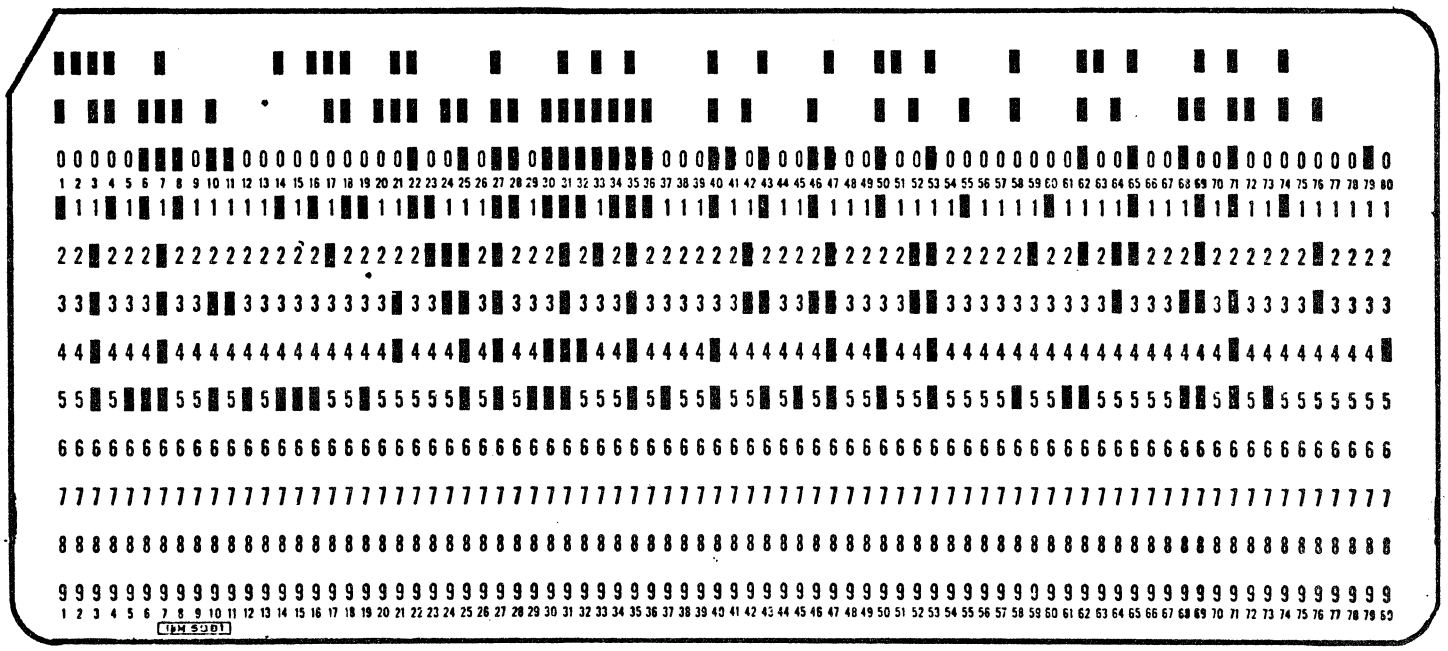
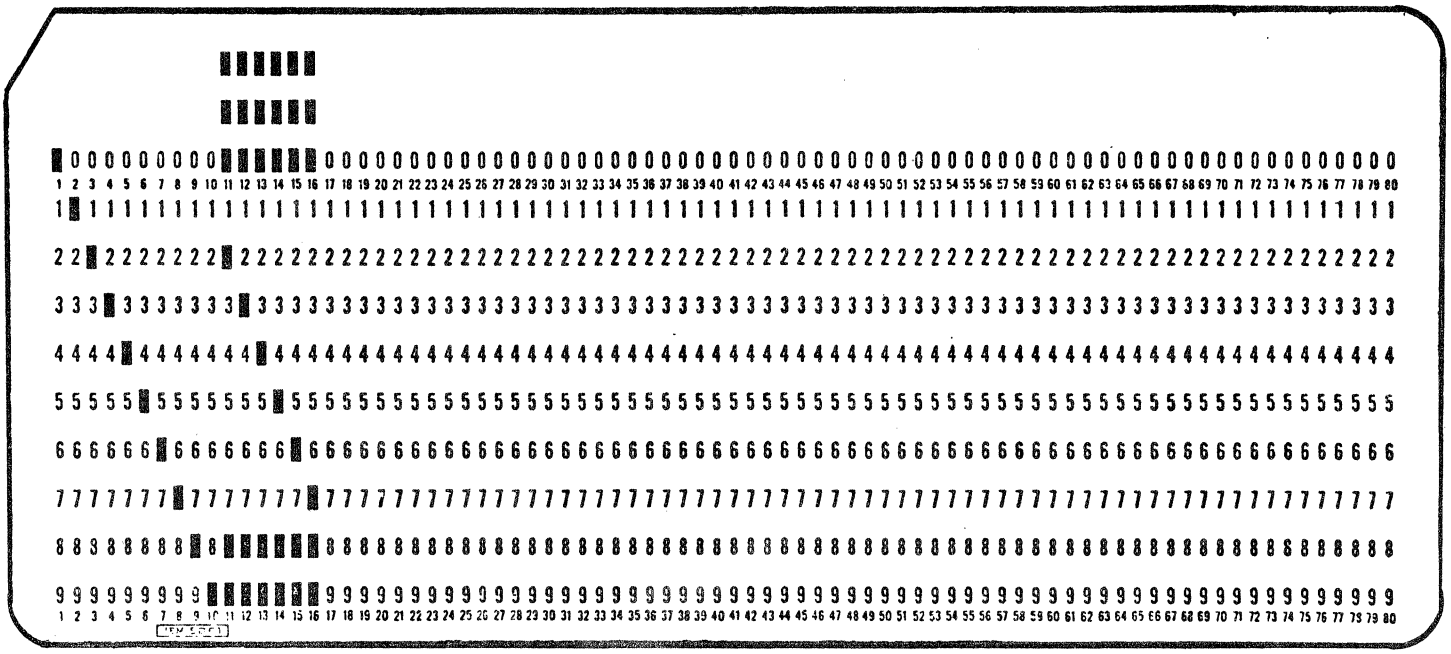


Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

CARD 5



CARD 6

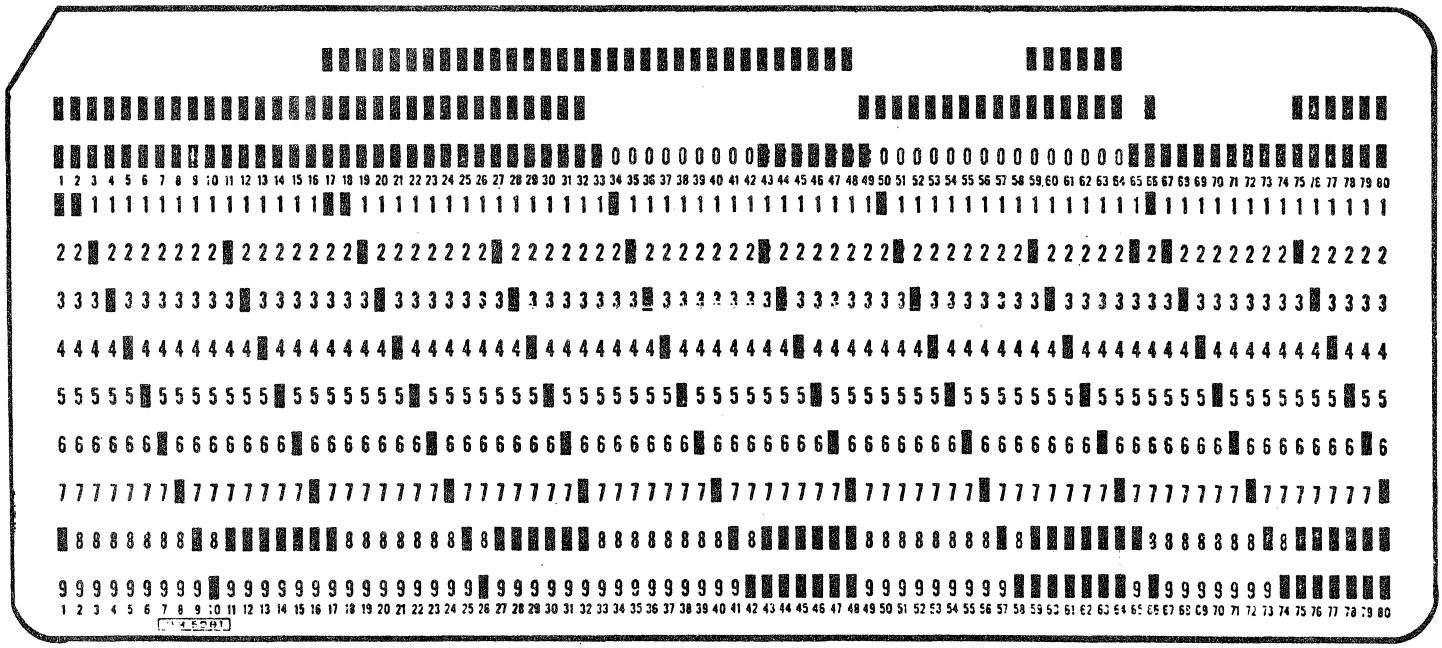
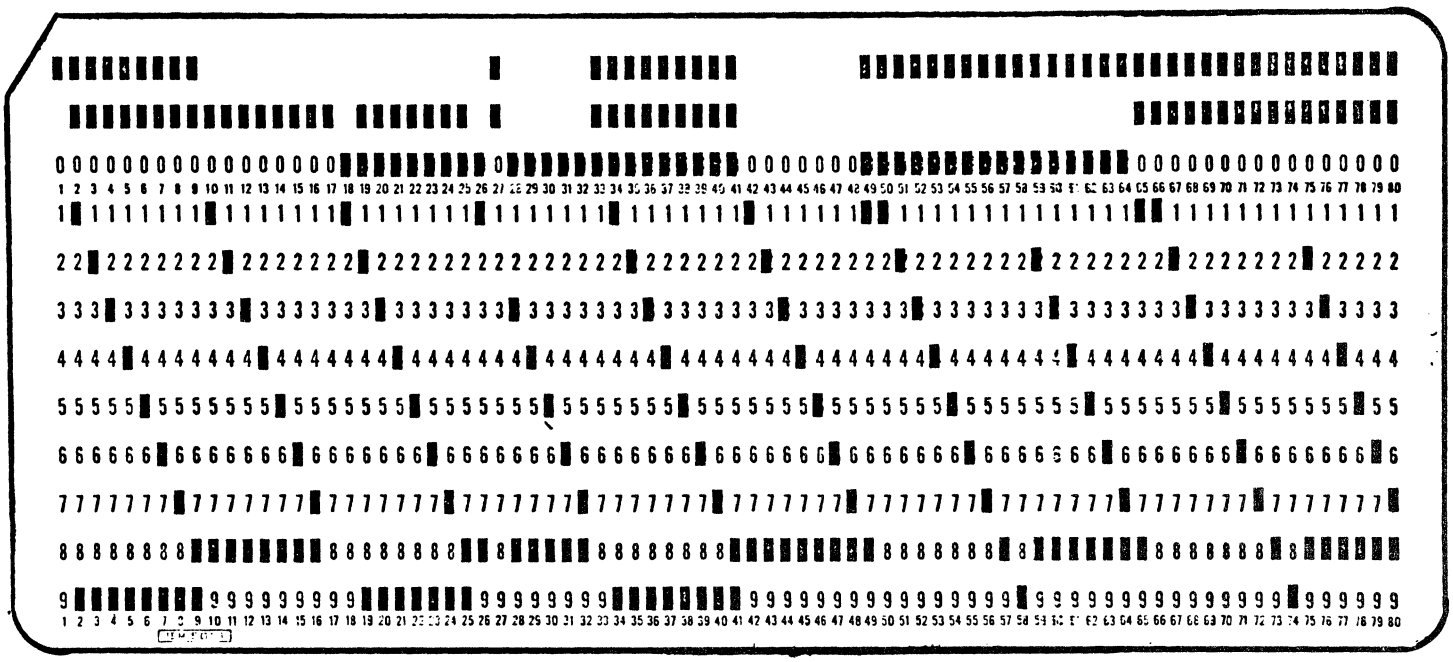
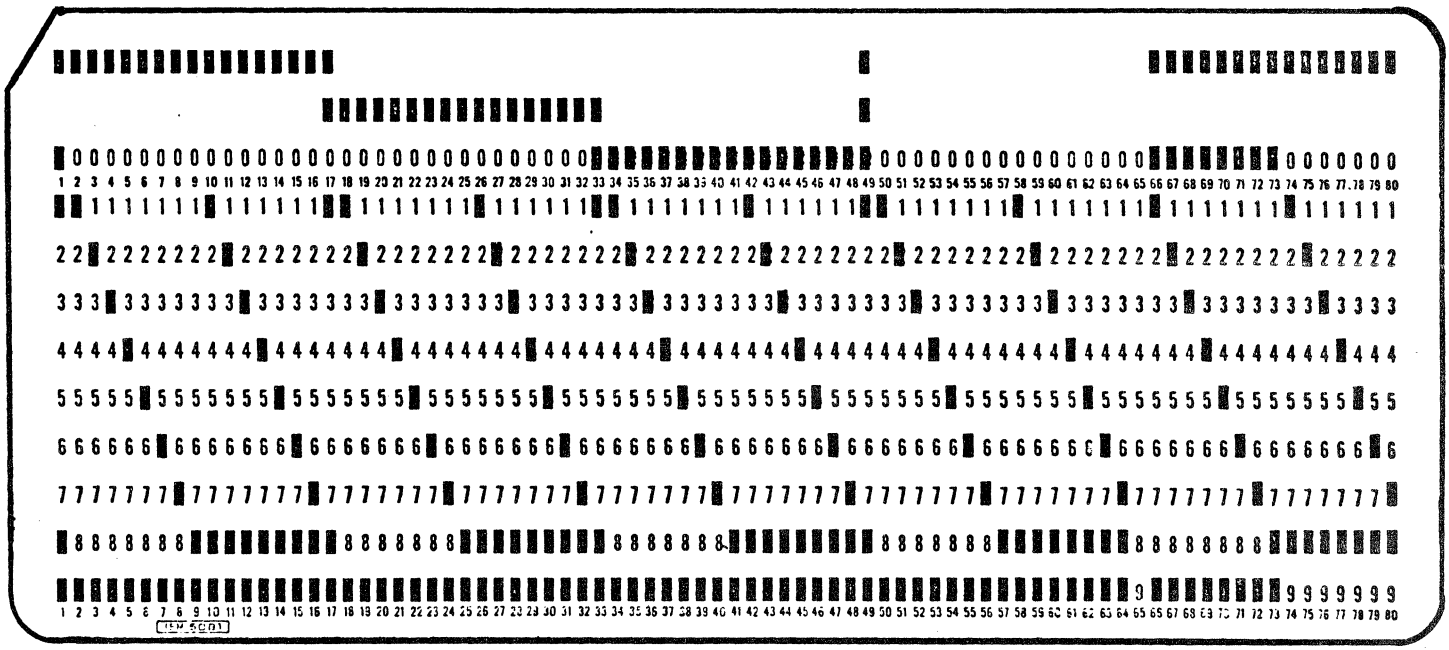


Figure 4.14.3 (CONT) - Remote Terminal Bootstrap Card Format

CARD 7



CARD 8



4.14.4 Remote Terminal Program 360 Processing (LETRRIP)

LETRRIP (Loader for Eleven-Thirty Relocatable Remote Interleaving Processor) is a 360 program executed under OS/360 as part of the RMTGEN procedure. The purpose of this program is to condense the object deck produced by the 360 assembler; relocate address constants according to the requirements of the 1130 and to produce a new object deck in the format as described in Section 4.14.1.

4.14.5 1130 Instruction Macros

The OS/360 Assembler Macro instructions 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 workstation 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/360 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/360 assembler SRL.

4. Parenthetical expressions are considered to be relocatable if contained in an instruction, e.g.,

```
     $AXT (*-*),WA,L
```

   is considered relocatable, where

```
     $AXT *-*,WA,L
```

   is considered absolute.

1130 Instruction Macros

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

Macro Form		Description And Notes
$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 non-zero (short)
$SKPZ		Skip ACC zero (short)
$SKPO		Skip overflow off (short)

Macro Form		Description And Notes
$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. FMT = X or blank generates $MDX ADD,TAG,FMT
$BSI	ADD,TAG,FMT,MOD	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

Macro Form		Description and Notes
$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

4.14.6 GENERAL INFORMATION

OS/360 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/360 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

VARIABLE	DESCRIPTION
&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.

4.15 EXECUTION TASK MONITOR

4.15.1 Execution Task Monitor - General Description

The Execution Task Monitor is a processor which periodically examines the CPU utilization of user tasks within a dynamic priority group and re-arranges the OS/360 task dispatching chain giving higher priority to those tasks, within the group, which use the least amount of CPU time. Tasks above and below the dynamic priority group are not affected by the rearrangement of the dispatching chain. Tasks with all of the following characteristics are included within the dynamic priority group:

1. The task is a job step of a job scheduled by HASP.
2. The current dispatching priority of the task is
   a. equal to priority of the dynamic group as specified by the value of the &XZPRTY parameter for MVT or
   b. not greater than the value &XZMFTH and not less than the value of &XZMFTL for MFT.
3. The job step is not multi-tasking. (The user does not ATTACH other tasks.)

The interval between the periodic examinations is controlled by the value of the &MONINTV parameter. Setting &MONINTV value to a positive integer will cause the processor to be generated. OS/360 must support Job Step Timing and must not have a Time Slicing Group Defined at the priority level(s) corresponding to the priority range of the dynamic group.

4.15.2 Execution Task Monitor - Algorithm

The Execution Task Monitor determines the CPU utilization history ($h_{t,n}$) for each task within the dynamic priority group using the following formula:

$$h_{t,n}=cpu_{t,n}+h_{t-1,n}-H_t/N$$

where: $H_t = cpu_{t,1}+h_{t-1,1}+cpu_{t,2}+h_{t-1,2}+\ldots cpu_{t,N}+h_{t-1,N}$

= Total CPU counts observed for the N tasks being monitored plus the sum of the previous history values.

N = The number of tasks being monitored at the end of the time interval.

$h_{t,n}$ = The history of CPU utilization for task (n) during the current time interval.

$h_{t-1,n}$ = The history of CPU utilization for task (n) taken at the previous time interval.

New tasks, entering the monitored group, will be assigned a history value of zero and temporarily placed at the low priority end of the group. Task with continuous low values of CPU counts will have (h) values which become increasingly negative. The (h) values will be prevented from falling below the range of one time interval; thus providing responsiveness to erratic changes in the corresponding task's CPU utilization.

Low values of h indicate the task (1) has not been able to utilize the CPU time given to it because of waiting for events such as I/O or (2)

has not been given the opportunity to utilize the CPU. High values of h indicate the task has had the opportunity and has utilized the CPU. The Execution Task Monitor performs a partial sort and rechains the monitored tasks, insuring that the task with the largest history of CPU utilization will have the lowest effective priority within the dynamic priority group during the next time interval. This will by default raise the effective priority of other tasks in the group.

4.16 INTERNAL READER

4.16.1 Internal Reader - General Description

The Internal Reader Processor is an Input Service Processor which reads card images from any system or user task running under OS/360. The Internal Reader recognizes, through the use of Execution Control Processor interface routines, an attempt by other tasks running under OS/360 to punch information into "cards" on pseudo 2520 punch devices, performs the function of the Input Service Processor on each card, and via OS/360 POST macro signals completion of I/O to the submitting task.

4.16.2 Internal Reader - Program Logic

The Internal Reader uses the code of the Input Service Processor with modifications in the following areas.

1. Processor Initialization - The Internal Reader attempts to obtain an internal reader device control table (DCT) which contains an 80 byte buffer area rather than a normal reader DCT. When a device is received the processor continues by acquiring a direct-access DCT and passes control to the main processor.
2. Main Processor - The Internal Reader RGET routine tests for the existance of a submitting task punch channel program.

If no channel program exists the Processor will wait for WORK. If a channel program exists RGET will simulate the punching of one card into the 80 byte DCT buffer area (no data chaining). If the channel command represents the end of the channel program RGET posts completion of I/O and resets channel program indicators. RGET returns to the main processor passing the card image for processing or in the event the "card" has "/*EOF" in columns 1 to 5 returns indicating end of file.

3. Processor Termination - Termination of the Internal Reader involves terminating the last job (if any), releasing the direct-access and internal reader device DCT's, and passing control to the processor initialization routines.

The Internal Reader requires supporting routines in the Execution Control Processor Asynchronous I/O Handler which perform the following functions:

1. Recognize EXCP macro references to designated internal readers as setup by HASP initialization.
2. Make the internal reader available to the Input Service Processor on first use.
3. Set up the first channel command word and IOB pointers.
4. Force the submitting task in the wait state if required.
5. Post the Input Service Processor for WORK.

4.17 MULTI-LEAVING LINE MANAGER

4.17.1 MULTI-LEAVING Line Manager — General Description

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 described in section 5.15.

4.17.2 MULTI-LEAVING Line Manager — Program Logic

When this processor receives control from the dispatcher it first determines whether an I/O operation has completed. If not, it then scans each line (via the line Device Control Tables) to check for requested processing. When all processing has been completed the processor then returns control to the dispatcher ($WAIT's) until such time as 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's) are scanned and when one is found to be available the Line Initiation routine is entered which acquires

the DCT, acquires a TP buffer, constructs an initial CCW chain, and initiates I/O on the line.

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 such that only the instructions required to process a given configuration will be generated.

4.18 REMOTE CONSOLE PROCESSOR

4.18.1 Remote Console Processor - General Description

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

4.18.2 Remote Console Processor - Program Logic

This processor receives control whenever a console message is queued for a remote terminal or whenever a console message is received from a remote terminal. 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 message is purged (the console message buffer is returned to the available queue) or the message is saved on the SPOOL1 volume if operator message SPOOLING is requested.

If the message is to be written, 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 messages which are queued are written to the terminal, and the DCT is "CLOSEd" and unchained. If the message to be written is for a currently inactive or for a non-MULTI-LEAVING active remote and HASP operator message SPOOLING space is specified (&SPOLMSG ≠ 0), an attempt to save the message on the SPOOL1 volume for later printing at the remote by printer support routines is made. 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. (Non-zero 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). 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 and any more messages for the same remote currently queued and written on the SPOOL1 volume

at the record location designated by the TRAILER MTTR for the remote. Upon completion of I/O the chain field replaces the TRAILER MTTR. The above process is repeated for additional buffers as required to empty the console message queue 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 SPOOL1 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, the messages are purged.

If an input message is to be read, 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 and then queued for the Command Processor.

4.19 EXECUTION THAW PROCESSOR

4.19.1 EXECUTION THAW PROCESSOR - GENERAL DESCRIPTION

XTHAW is a companion to the main Execution processor IOS interface routine called XFREEZE. XTHAW is responsible for discovering which tasks have been forcibly placed in an OS WAIT state by XFREEZE (frozen) and should now be activated (thawed) thru the OS POST ECB mechanism.

4.19.2 EXECUTION THAW PROCESSOR - PROGRAM LOGIC

XTHAW uses an IOB (HASP buffer) chain constructed thru the XTHAW PCE or the Execution Processor PCE(s) in the XPCEECB field of the PCE work area. The chain is constructed using the XTHAW or an Execution PCE depending on the reason for invoking XFREEZE. If the IOS interface section is entered while an Execution processor is active, then the XTHAW PCE is used. If an I/O request cannot be processed and an Execution processor is not active at the time of the request, then the PCE controlling the caller is used to build the chain.

XTHAW is activated ($POSTed) by the Execution Processor whenever a Job or the HASP controlled OS Reader/Interpreter is active and just prior to $WAITing for work. A special status bit (XPOSTBIT) in the XPCESTAT field of an Execution PCE is used as the primary test for processing the IOB chain. This bit is not turned on when the OS Reader/Interpreter is active and assigned to a PCE but does not have a job to process. This prevents unnecessary activation (thawing) of the Reader/Interpreter when no Jobs are available for initiation.

XTHAW performs the following major functions:

(1) Examines the XPCEECB field of the XTHAW processor. If this field is non-zero, it is used as the pointer to a chain of IOBs (HASP buffers) which contain ECBs to be POSTed (thawed) and the HASP/OS POST subroutine (WPOSTECB) is used to perform the POST. The chain address for the IOBs is contained in the IOBCSW field which is set to zero for the last IOB.
(2) Next, the Execution PCEs are searched for the XPOSTBIT condition and the XPCEECB field is processed as described in Step 1, if the XPOSTBIT is on.
(3) XTHAW $WAITs for work after processing all PCEs as described.

4.20 OVERLAY ROLL PROCESSOR

4.20.1 Overlay Roll - General Description

This Processor operates in conjunction with the Overlay Service Routines. Description of them in Section 5.16 should be read 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.

4.20.2 Overlay Roll - Program Logic

This Processor's PCE is placed lowest on the HASP Dispatcher chain and it $WAITs on ABIT when idle. 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. Overlay Roll always receives control, just before WAIT is executed.

Overlay Roll has local addressability provided in BASE2 and also establishes the base address for Overlay Service in register WC so that its subroutines and tables may be used. On each entry, the Queue beginning with $WAITACE (see 5.16.2) of PCEs waiting for an Overlay Area is tested. If empty, $WAIT ABIT is used to exit. Otherwise, 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 an overlay routine is in process, the area is never taken. Users of that routine are allowed at least one chance to execute after read completion is processed by Overlay $ASYNC Exit (see 5.16.9).

For each Overlay Area which does not have 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 on I/O. This would be I/O other than an overlay read operation, would be expected to complete "soon", and would, therefore, make it less desirable to pre-empt that area. The lowest priority area with no user $WAITing 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 disk, a pre-empted Overlay Area. Each PCE on the chain of current users (OACEPCE) is

processed to prevent further use of the pre-empted area by it. The re-entry address (PCER15) is "sized" to determine if it points into the Overlay Area and if so is relativized 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 (see 5.1.2) detects the PCE $WAITing OROL only and sets it to call on Overlay Service. OLOD subroutine (see 5.16.8) is eventually called to refresh the routine, either directly, or if the PCE gets into the $WAITACE Queue, by OEXIT subroutine (see 5.16.7) or by Overlay Roll.

The area thus pre-empted is used to read in a new overlay routine, to be used by the highest priority PCE group on $WAITACE. The OLOD subroutine (see 5.16.8) 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 by $WAIT ABIT, the $WAITACE Queue is either empty or all Overlay Areas have an overlay read operation started, to be posted by Overlay $ASYNC Exit.

4.21 HASP SMB WRITER (HASPWTR)

4.21.1 HASP SMB Writer - General Description

The primary function of this program 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 task, in the HASP region or partition, if the HASPGEN parameter &WTRPART is set to "*". Otherwise, the standard OS Output Writer is used to fulfill the same functions and is started by HASP in a separate partition, using a procedure named HOSWTR. The re-queuing feature described below is only available when using HASPWTR.

The program HASPWTR depends upon OS Queue Management structures (QCR, LTH, SMB, no-work ECB) as documented in OS/360 MVT Job Management PLM. Functions such as enqueue, dequeue or delete of a job; ENQ/DEQ to control access to Queue Management resources; and conversion of record addresses between NN, TTR0, and MBBCCHHR forms are all performed in a manner consistent with that described for the standard OS Job Management modules.

Microfiche listings for IEFQDELQ, IEFQMDQQ, and IEFSD086 were consulted as examples during the development of HASPWTR. However, no actual Job Management modules are executed by HASPWTR.

4.21.2 HASP SMB Writer - Program Logic

On initial entry after being ATTACHed, the program saves three addresses passed to it by HASP Initialization: memory 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 $P HASP. After signalling 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 $P HASP ECB which, if posted, causes the program to terminate as described below. All other ECBs are each part of an eight 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 re-written. If there were no jobs to dequeue or the one dequeued was the only one, the class ECB is cleared and the no-work element is chained from the QCR before re-writing.

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 original class in the list &WTRCLAS, is a valid SYSOUT class then the job is re-queued in the QCR for that new class. Any tasks (e.g., other system writers for perhaps CRBE, CRJE, TSO, CPS, etc.) whose no-work elements are chained from that QCR are POSTed. The re-queue action always places the job in the new QCR chains at highest priority.

If &WCLSREQ does not indicate re-queuing ("*" 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 re-writing 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 $P HASP, 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, then the QCR is re-written. 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 de-queue 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 signalled to HASP and no further blocks on that job's chain are read. If a QCR read error occurs during a re-queue attempt, the job is deleted (tracks are released).

4.22 PRIORITY AGING PROCESSOR

4.22.1 Priority Aging Processor -- General Description

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.

4.22.2 Priority Aging Processor -- Program Logic

When this processor is dispatched, it searches through the HASP Job Queue until it encounters a Job Queue Element whose priority field "QUEPRIO" (see figure 8.6.1) 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 sixteen times, however, the actual value of the priority will be increased by one. The value of the time interval is actually only one-sixteenth 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.

In order 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.

4.23 SYSTEM/3 REMOTE TERMINAL PROCESSOR PROGRAM LOGIC

The HASP System/3 Remote Terminal Program is assembled on a System/360 or System/370 computer under OS, using Assembler F (IEUASM). 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 former 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 "Card and Disk System 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 instructions 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 one might write "$MVC X'1234',(0,REG2),LENGTH" to move 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 two-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 one-address form exemplified by "$L reg,adr" and including $L, $A, $LA, and $ST.

3. The one-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 one-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, one might write "$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 one-to-eight-character expression with the location counter reference symbol "*" either not used or used as the first character, and "register" is a one-to-eight-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 "(displ,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(displ). If a match is found, the displacement is assembled as YL1(displ-(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, in order 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 one-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 one-card loader. Each such 96-column card image contains 64 bytes of information as follows:

- bytes 1-5 contain a System/3 $MVC instruction of the form "$MVC load-adr,(column-number,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;
- bytes 6-61 contain a maximum of 56 bytes of text, starting in column 6; and
- 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:

- 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;
- bytes 62-64 contain a three-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 three-digit card sequence number.

Certain of these 96-column card images contain descriptive information in bytes 33-64: these are the one-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 one-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:

- columns 1-2 are blank;
- 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;
- columns 51-72 are blank;
- 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);
- 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 punched 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 non-transparent mode.

The following pages constitute the Program Logic manual for the System/3 Remote Terminal program.

The 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 upon 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 two-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 two-byte bit counters is by a bit string (starting at symbol IHBIT1) 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 Remote Terminal program, based upon the status of the various Function Blocks (FBs).

If the Event Wait Field (FBEWF) of an FB has zeroes in the bit positions defined by EWFALL, the function is said to be dispatchable. $COM loads register one from field FBREG1 of the FB (register two 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 two 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 one; $COMRETA, where $COM will save the value of register one; 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 insure 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 non-process exits, exactly as it did for HTIO above. But if the condition code is equal, HEXCP non-process exits to be entered again at a $TIO which continues to non-process exit until the MFCU ceases being busy; then HEXCP calls third-level subroutine HSNS to determine the completion of the I/O operation.

HSNS 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 non-process exits until a SNS instruction shows that all error bits (except no-op) have been reset by the operator (who must do a non-process 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 non-process 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 non-process

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 non-process exits until the associated 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 insure 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 errors 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 non-blank, 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 status bit from the 1442 and sets 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 non-process exits on a $TIO until the hopper is ready. Then it issues the constructed $SIO and non-process 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 to 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 non-process 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 non-process 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 non-process 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 non-process 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 non-process 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, $READER processes the card normally, as above. But if column 81 equals "1", the card is the first card of a two-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 non-process 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 non-process 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 sixteen 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 non-process 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 non-process 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 print-buffer-busy bit, and non-process 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, $PRINTER 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 $PERM) and non-process 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 non-process 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 72, 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 non-process 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 &S3OBJDK=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, non-process 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.

At a Request key interrupt, CINT posts the console processor if an inspection of the console status byte CCFLG shows that neither input nor output is currently in process. In any case, it sets the request-pending bit and exits by issuing a SIO which turns on the request-pending light, disables request key interrupts, and leaves the proceed light and other key interrupt indicators as they were 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 starting 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, non-process 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 lock-out 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 non-process 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 non-process 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 four-byte coded error message to the equivalent eight-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 non-process 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 non-process 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 field-erase key causes CINT to reset the buffer pointer, set the column indicators to display "01", and exit.

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, function key, and 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 a 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 non-process 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 four-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 non-process 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 re-transmit.

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.

If no hardware error occurred, the starting sequence is checked at BCROK - it is valid if it is a NAK or a DLE-ACK0 or if its second byte is STX and the last byte received is ETB. If none of these is the case, BCROK sets up an error message of 03SSSS00 (where SSSS is the starting sequence) and then transmits a 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 re-transmits the buffer to HASP.

If the received sequence was DLE-ACK0, BSCINT sets flags to show $BSCA that a transmit-receive operation has completed; then it exits by starting a two-second timeout. If the two-second timeout completes before $BSCA has cancelled it, BSCINT sets the two-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 two-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 BCB's, and it builds and transmits to HASP a BCB-error control sequence.

$BSCA - Communications Adapter Processor

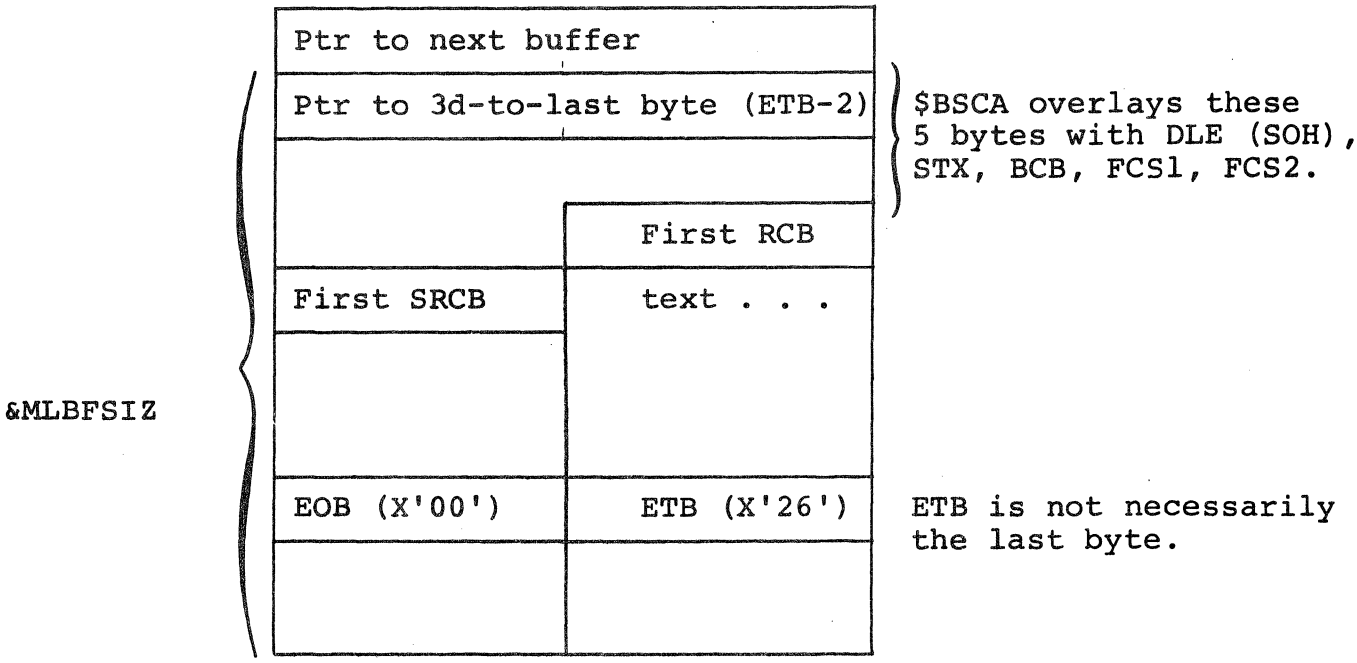
$BSCA non-process exits until BSCINT posts it with an indication that either an error message awaits synchronous processing, a receive operation has completed without error, or a two-second timeout has occurred.

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 re-entrant. $BSCA calls the $MSG subroutine to add the error message to the trace 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 FB's 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.

Having processed the received buffer or if a two-second timeout occurred, $BSCA determines what and when it is to transmit. $BSCA inspects the most-recently-received FCS for the wait-a-bit; if it is on, $BSCA transmits immediately a DLE-ACK0 (or a wait-a-bit sequence if there are no free MULTI-LEAVING buffers).

But if the received wait-a-bit is off, $BSCA inspects its buffer chain. If the chain is non-empty, $BSCA prepares to transmit the first-chained buffer to HASP. Each buffer on the chain has the following format, as set by subroutine $CKLEN:



If there is no text buffer to send, $BSCA checks for a free MULTI-LEAVING buffer. If there is no free MULTI-LEAVING buffer, $BSCA sends a wait-a-bit sequence if a two-second timeout has occurred; otherwise it non-process exits awaiting either a two-second timeout or the setting of its BFPOST flag by subroutine $CKLEN.

If a free MULTI-LEAVING buffer is available, $BSCA will use it only if the last-received buffer was a text buffer (that is, its second byte was STX) or if the master Function Control Sequence, $FCS, had changed from that last transmitted to HASP. If the latter is the case, $BSCA will transmit immediately a logical acknowledgment (DLE or SOH, STX, BCB, FCS1, FCS2, EOB, ETB) to inform HASP of the change; if the former is the case, $BSCA will transmit a physical acknowledgment (DLE, ACK0). If neither is the case, $BSCA will non-process exit until either a two-second timeout occurs (whereupon it will send DLE-ACK0 or the wait-a-bit sequence) or $CKLEN sets BFPOST (whereupon $BSCA will again check for a received wait-a-bit).

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.

$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 three-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, a two-byte field in the caller's FB reserved for the use of $DCOM. If that field is non-zero, 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 non-zero 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 workarea and transmits it to HASP by calling subroutine $CKLEN.

When called, $CMPR examines the status of its local workarea. If the workarea is busy, $CMPR has been called by some other processor and has in turn called $CKLEN; $CKLEN is non-process exiting until it can find sufficient bytes in a MULTI-LEAVING buffer to allocate to $CMPR. In this case, $CMPR non-process exits until its workarea becomes free.

When the workarea 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, $CMPR moves the compressed record, shows its workarea 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 queuing), $CKLEN attempts to get another buffer to satisfy the caller's request; if no buffer is free, it non-process 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 OR's the two byte field FBFCS into the two-byte field $FCS and compares the resulting $FCS with BCXFCS, the FCS last transmitted to HASP. If $FCS is not equal to BCXFCS, $FREEBUF posts $BSCA.

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 four-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 120 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 one, and register two as of the time ABEND gained control; the remainder of core is intact.

$LOG - HASP Error Recording Subroutine

$LOG is a re-entrant 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 four-byte coded entry addressed by register one 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 - Initialization 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 others 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 repped 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 lengtl 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 re-executed. 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 two-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 re-executed.

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 consists 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.

(The remainder of this page intentionally left blank.)

5.0 HASP CONTROL SERVICE PROGRAMS

This section contains detailed internal information about each of the HASP Control Service Programs and is intended primarily for use by systems programmers.

5.1 HASP DISPATCHER

5.1.1 HASP Dispatcher - General Information

The HASP Dispatcher is responsible for the allocation of the CPU time used by the HASP Task to each of the HASP Processors.

5.1.2 HASP Dispatcher - Program Logic

The HASP Dispatcher receives control each time the HASP task is dispatched by the Operating System and utilizes an ordered queue of Processor Control Elements (PCE's) to distribute the CPU time among the HASP Processors. The Event Wait Field (EWF) in each PCE (see Figure 8.2.1) is a two 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 on and returns control to the HASP Dispatcher to allow other processors to be dispatched. The Processor (or Service Routine) will not be given control again until some other system function issues a $POST to its EWF for the event $WAITed on.

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 on and corresponds 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 EWF's 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 (additional details regarding $WAIT/$POST events may be found in Section 9.8).

When scanning the PCE chain, the HASP Dispatcher, upon discovering a zero EWF (no events being $WAITed on), will enter the code controlled by the PCE immediately below the prior $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 $POST 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 $POST 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 the processor active count to determine if any jobs are being processed. If an active job is in the system (active count ≠ 0) an OS WAIT state is entered to wait for some external event (I/O interrupt, etc.) to activate HASP again. This WAIT allows use of the CPU by other tasks in the system. If no jobs are active, the message

"ALL AVAILABLE FUNCTIONS COMPLETE"

is sent to all operator consoles and HASP is placed into the WAIT state.

When scanning the PCE chain, the Dispatcher detects the special case of a PCE which is not dispatchable (PCEEWF is not zero) but is $WAITing only on the OROL bit. This situation is created when, while the PCE was $WAITing on other event(s), the Overlay Area being used by the PCE is preempted by the Overlay Roll Processor for other use (see Section 4.20). Subsequently, the other event(s) being $WAITed on are $POSTed allowing the Dispatcher to detect the "OROL only". The Processor in such a condition is not entered but is made to call Overlay Service. The actions performed are identical to those described for $LINK Service in Section 5.16.2, beginning with the fourth paragraph describing search of Overlay Areas. The Processor will be entered by Overlay Service if the requested routine is in memory, or will be $WAITed on OLAY allowing the Dispatcher to continue its PCE scan.

5.2 INPUT/OUTPUT SUPERVISOR

5.2.1 Input/Output Supervisor - General Description

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 remain "device independent" through the wide range and number of HASP direct-access devices. In addition, $EXCP also provides all required I/O appendages for OS IOS and for the $POSTing of I/O completions to each processor.

5.2.2 Input/Output Supervisor - Program Logic

The only interface between the HASP Input/Output Supervisor and the using processors is the Device Control Table element (DCT), which is passed via the $EXCP macro-instruction when I/O is requested. (Additional information concerning the DCT and the $EXCP macro may be found in Sections 8.5 and 9.5 respectively.) Upon entry to $EXCP the address of the buffer to be used is obtained from the DCT and the IOB (appended to 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.

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 appropriate channel end appendage in $EXCP to be entered upon termination of the I/O. Since these appendages are entered asynchronously with HASP operation, the buffer associated with the completed I/O is scheduled for synchronous HASP processing by the Asynchronous Input/Output Processor. The HASP task is POSTed, and immediate return is made to IOS. (The action taken by the Asynchronous Input/Output Processor is explained in Section 4.8.)

A separate channel end appendage is provided for remote terminal operations. This appendage correlates the channel end conditions with the commands executed and provides special processing of conditions unique to the teleprocessing.

5.3 JOB QUEUE MANAGER

5.3.1 Job Queue Manager - General Information

Jobs being processed or awaiting processing by a HASP phase are represented in an ordered queue by a Job Queue Element (see Figure 8.6.1).

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 (see Section 9.3). 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.

5.3.2 $QADD Routine - Program Logic

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 onto the SPOOL1 disk, 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 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 SPOOL1 disk. The condition code is set non-zero and control is returned to the caller. Upon return, register "R0" contains the address of the associated Job Information Table Entry.

5.3.3 $QREM Routine - Program Logic

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 non-zero to indicate that the Queue Element was successfully removed.

5.3.4 $QGET Routine - Program Logic

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, print, punch, or purge) which is not in hold status and 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 non-zero, 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 such 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.

5.3.5 $QPUT Routine - Program Logic

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 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 SPOOL1 disk. If the QUEPURGE bit is on in the queue element (indicating that the job has been deleted), the job queue element is placed in the punch queue by moving the punch queue type into the queue element's QUETYPE field. If the QUEPURGE bit is not on, the job queue element is placed in the queue indicated by register "R0" upon entry to this routine. The QENTBY bit is turned off to indicate that this queue entry has been returned, the condition code is set non-zero, and control is returned to the caller. Upon return, register "R1" contains the address of the Job Queue Entry just returned and register "R0" contains the address of the associated Job Information Table Entry.

5.3.6 $QLOC Routine - Program Logic

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 non-zero, 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.

5.3.7 $QSIZ Routine - Program Logic

The $QSIZ routine is entered to obtain the number of Job Queue Elements in a given queue type, route, class, and forms. 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 non-zero, the condition code is set non-zero, 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 set to zero to indicate that no jobs are available in the specified job queue.

5.4 BUFFER MANAGER

5.4.1 Buffer Manager - General Description

The Buffer Management routines are responsible for the allocation of the dynamic memory area (Buffer Pool) of HASP. Fixed-size buffers in this area are allocated and de-allocated to HASP Processors and Routines via the $GETBUF and $FREEBUF macro-instructions (see Section 9.1).

5.4.2 Buffer Manager - Program Logic

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 assure that the buffer is within the buffer chain. If it is not in the chain, the catastrophic error routine is entered; otherwise, control is returned to the $GETBUF routine. The condition code is set non-zero 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, the buffer to be freed is inserted back into the appropriate free buffer chain (depending upon whether the buffer is a HASP Buffer or an RJE buffer), and the IOBSTART field is updated with the address of the buffer's channel program: IOBCCW1 (see Figure 8.3). The HASP Dispatcher's Event Control Field is $POSTed to show that a buffer is available and control is returned to the caller.

5.5 UNIT ALLOCATOR

5.5.1 Unit Allocator - General Description

The Unit Allocation routines are responsible for the allocation and de-allocation of the Input/Output units which have been assigned to HASP. Device Control Tables (DCTs) are allocated and de-allocated to HASP Processors and Routines via the $GETUNIT and $FREUNIT macro-instructions (see Section 9.2).

5.5.2 Unit Allocator - Program Logic

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 "not available" and control is returned to the caller with the condition code set non-zero. The address of the DCT is returned in register "R1".

The $FREUNIT routine first examines the "Active Buffer Count" field of the DCT (see Figure 8.5) to see if there are any buffers involved in active I/O with the associated unit. If the "Active Buffer Count" is non-zero, the Processor is placed in a HASP $WAIT state until this count is reduced to zero. When the count is zero, the DCT is made available and control is returned to the caller.

5.6 INTERVAL TIMER SUPERVISOR

5.6.1 Interval Timer Supervisor - General Description

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 upon expiration of the interval. This routine uses the standard OS/360 timer features (STIMER & TTIMER) but has the additional capability to simultaneously monitor an unlimited number of intervals.

5.6.2 Interval Timer Supervisor -Program Logic

All uses of the Interval Timer Supervisor are through the HASP macro-instructions $STIMER and $TTIMER which are described in Section 9.6. Each user of $STIMER is required to provide a 12-byte (three-word) HASP Timer Queue Element (TQE), passed via parameter register "R1" (see Section 8.10). $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). Upon being entered with a new interval request, $STIMER first 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 re-activated 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 first reduces the intervals of all queued TQEs by the size of the just-elapsed interval, then $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 re-activated with the interval from the first TQE on queue and return is made to the caller. NOTE: A $TTIMER for a TQE which is not active has no effect and a zero value is returned in register "R0" as the time remaining.

5.7 $WTO PROCESSING ROUTINE

5.7.1 $WTO Processing Routine — General Description

This routine services the $WTO macro-instruction (see Section 9.5) by queuing the associated message for the Operator Console Input/Output Processor.

5.7.2 $WTO Processing Routine — Program Logic

This routine tests for a free message buffer. If none are available, it causes the requesting processor to be placed in a $WAIT condition until a message buffer is released. Otherwise it links to the Console Buffering Routine to process the message.

5.8 DIRECT ACCESS STORAGE ALLOCATOR

5.8.1 Direct Access Storage Allocator - General Information

This routine allocates tracks for the SPOOL volumes that were on-line at IPL time. The track information is stored in the Job Control Table (JCT) and is also returned to the caller in register "R1". The track allocation algorithm is designed to reduce seek time as much as possible.

5.8.2 Direct Access Storage Allocator - Program Logic

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

Track groups on the SPOOL volumes are allocated whenever the JCT has not previously acquired any tracks or whenever all the tracks in the current track group which is allocated to the JCT have been acquired. If the JCT has already been allocated 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 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 in the following format: MTTR, where M is the module number (one byte), TT is the track number relative to cylinder 0 track 0 (two bytes), and R is the record number (one byte). The JCT 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 track group bit map.

When a new track group has to be acquired, seek time is reduced by searching for the nearest track group + or - 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 - one track group from the last-used track group, then + one from the last-used track group and this progression continues until an

available track group is found or the + eight track group is searched. If an available track group is found, the JCT 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 track group bit map is also updated. If a track group is not available within + or - eight of the last-used track group, another search routine is entered which inspects each byte of the track group maps, starting with the first byte. This search will continue until an available track group is found or until all of the active track group bit maps have been searched. If an available track group is found, the JCT 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 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.

5.8.3 Direct Access Storage Purge Routine - Program Logic

This routine frees all of the SPOOL volume tracks that the job has acquired and informs the system that these tracks are available to be re-acquired.

The track group bit map in the job's Job Control 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 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.

5.9 DISASTROUS ERROR HANDLER

5.9.1 Disastrous Error Handler - General Description

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.

5.9.2 Disastrous Error Handler - Program Logic

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
```

5.10 CATASTROPHIC ERROR HANDLER

5.10.1 Catastrophic Error Handler - General Description

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 one instruction disabled loop. The error codes and their meanings are listed in the HASP Operator's Guide (see Section 11). For more information, refer to Section 9.10.1.

5.10.2 Catastrophic Error Handler - Program Logic

When this routine is entered, register "R0" contains the address of a four byte field containing the three character error code left justified. After the system is disabled, the four byte error code field is moved into the operator message. This message is then written on the operator's console defined by the HASPGEN parameter "$PRICONA":

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

After this message is typed, all registers are restored so that they will be intact, and a one instruction loop is executed.

5.11 TRACE EFFECTOR

5.11.1 Trace Effector — General Description

The Trace Program is a debug facility used in HASP which is completely independent of the OS trace facility. This program will insert the contents of the general purpose registers into a special trace table (assembled into the HASP module) each time it is called and thereby aid in the determination of HASP problems.

5.11.2 Trace Effector — Program Logic

The Trace Program is called by any Routine or Processor in HASP by the insertion of a $TRACE macro-instruction (see Section 9.9.1). If the HASPGEN parameter "&TRACE" is set non-zero, the macro-instruction will expand into an instruction which will cause a unique specification program interrupt. All program interrupts are fielded by the HASP Trace Program and the instruction which 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 sixteen word trace entry is inserted into the HASP trace table.

The sixteen word trace entry has the following format:

First Byte.....$TRACE count

First Word.....$TRACE storage location

Second Word.....Register 0

Third Word.....Register 1

Fourth Word.....Register 2

Fifth Word.....Register 3

Sixth Word.....Register 4

Seventh Word.....Register 5

Eighth Word.....Register 6

Ninth Word.....Register 7

Tenth Word.....Register 8

Eleventh Word.....Register 9

Twelfth Word.....Register 10

Thirteenth Word.....Register 12

Fourteenth Word.....Register 13

Fifteenth Word.....Register 14

Sixteenth Word.....Register 15

After the trace table entry has been inserted and the pointers 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

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 symbolic location "$TRACETB" in HASP identifies a three-word table with the following format: the first word is the address of the last entry which 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.

5.12 WTO/WTOR PROCESSING ROUTINE

5.12.1 WTO/WTOR Processing Routine — General Description

The function of this routine is to process all OS WTO's and WTOR's. If a console buffer is not available for the message the requesting task is placed in an OS WAIT state until a buffer becomes available to process the request. This routine is not included if the HASP interface to OS Console Support is generated (&NUMCON=0, see Appendix 12.15).

5.12.2 WTO/WTOR Processing Routine — Program Logic

The WTO/WTOR Processing Routine is entered from the OS Execution Control Processor whenever an SVC 35 or SVC 36 is issued. Upon entering, a test is made to determine if a free console buffer is available. If none are available the user is made to WAIT using the first word of his calling sequence as an ECB. The information necessary to later dispatch the task is saved in an entry in the $WTORQUE (see figure 5.12.1).

If an unused buffer exists register zero is set with the parameters required by the Console Buffering Routine (see figure 5.13.1), register one is set with the message address, and if a job number exists for the job, register 10 (JCT) is set with the JCT address. If the SVC represents

a WTOR, a reply number is established and a WTOR Reply Element (see figure 5.12.2) is set up to service the subsequent operator reply.

A link is then made to the Console Buffering Routine to initialize the buffer with the message. Upon return, HASP is POSTed and the processing routine exits to the OS dispatcher via register 14.

Figure 5.12.1 -- WTO/WTOR TASK WAIT ELEMENT

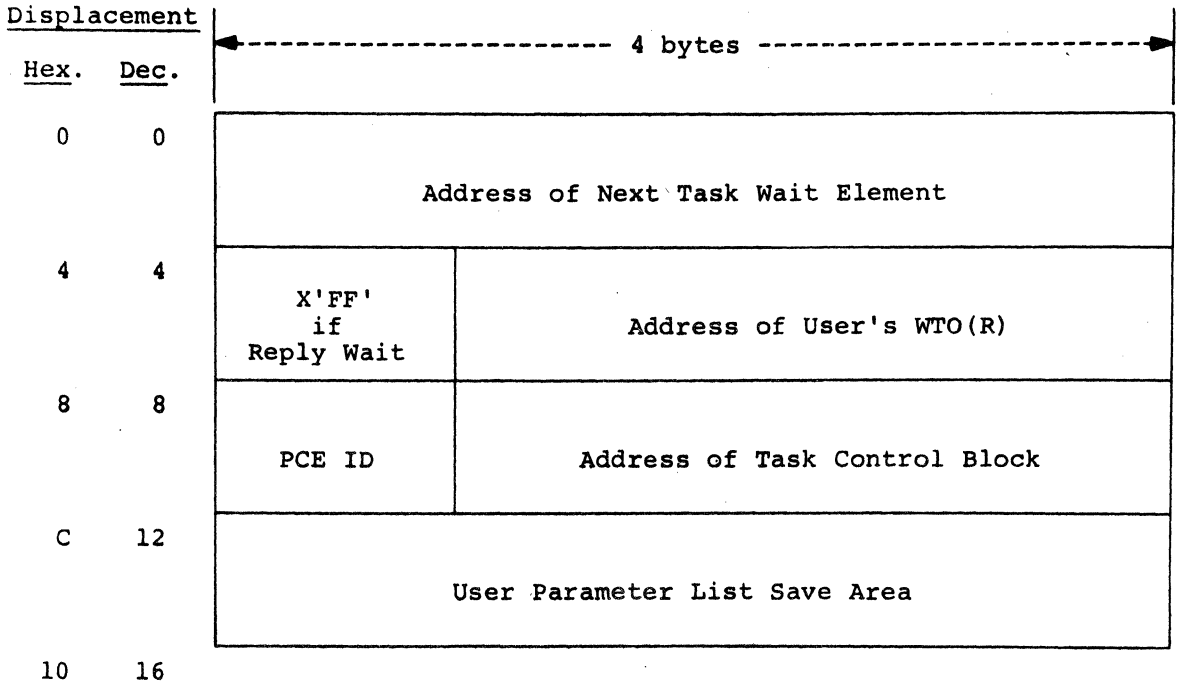
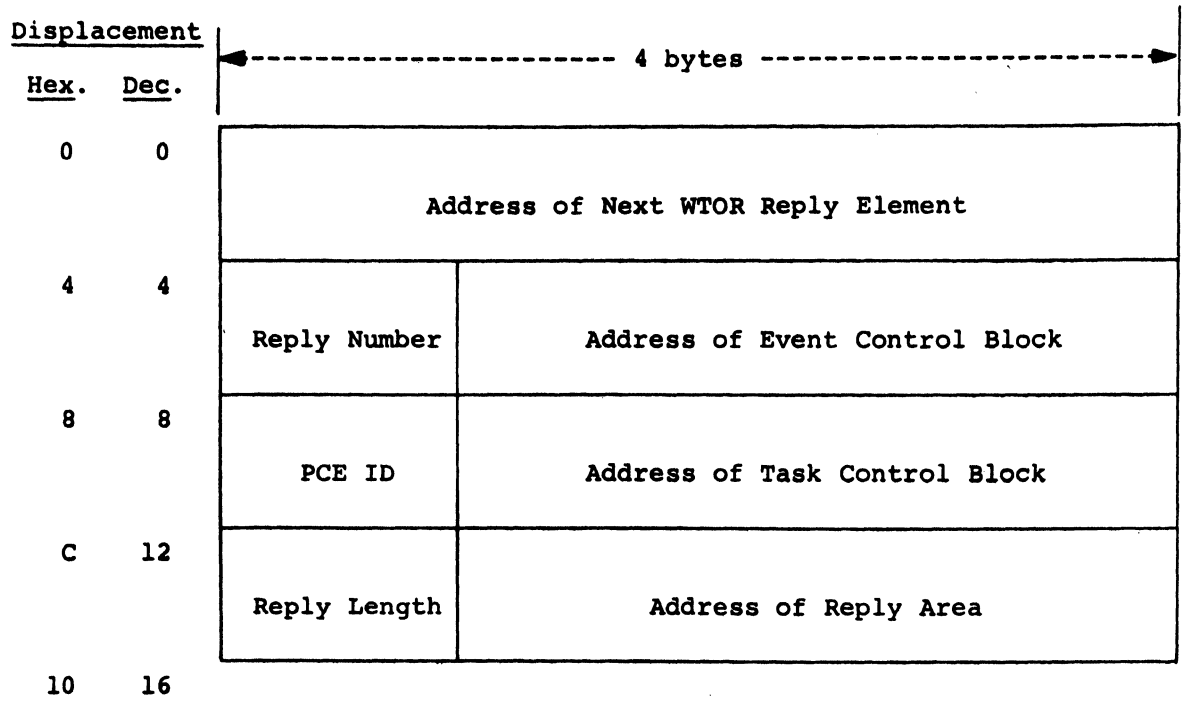


Figure 5.12.2 -- WTOR REPLY ELEMENT



5.13 CONSOLE BUFFERING AND QUEUING ROUTINES

The following routines are responsible for the queuing and de-queuing of all console and log messages.

5.13.1 CONSOLE BUFFERING ROUTINE - PROGRAM LOGIC

The Console Buffering Routine is used to prepare a message buffer with the information required to process a console message. At entrance registers zero and one contain the information shown in figure 5.13.1.

The routine makes use of three tables comprised of one-byte entries. The bits in each byte specify the physical consoles which are to be used for the respective entry. In the first ($WCONTBL) each byte corresponds to one of eight consoles. The bytes are ORed for each specified symbolic console to set the physical byte for a write operation.

A second table ($WCLASTB) has an entry for each of the sixteen possible message classes. The appropriate byte is ANDed with the physical consoles byte to screen out consoles with the class set too high.

In addition to setting the console routing byte, the Console Buffering Routine supplies the other information shown in figure 8.4.1. Prior to returning to the caller, the routine places the message in the log queue (non-HASP messages with a job number), or in the queue of messages to be processed by the Console Input/Output Processor (all other output messages and all reads).

5.13.2 CONSOLE QUEUING ROUTINE - PROGRAM LOGIC

This routine places a console buffer into a queue of messages, according to priority, to be processed by the Operator Console Input/Output Processor and $POSTs that processor.

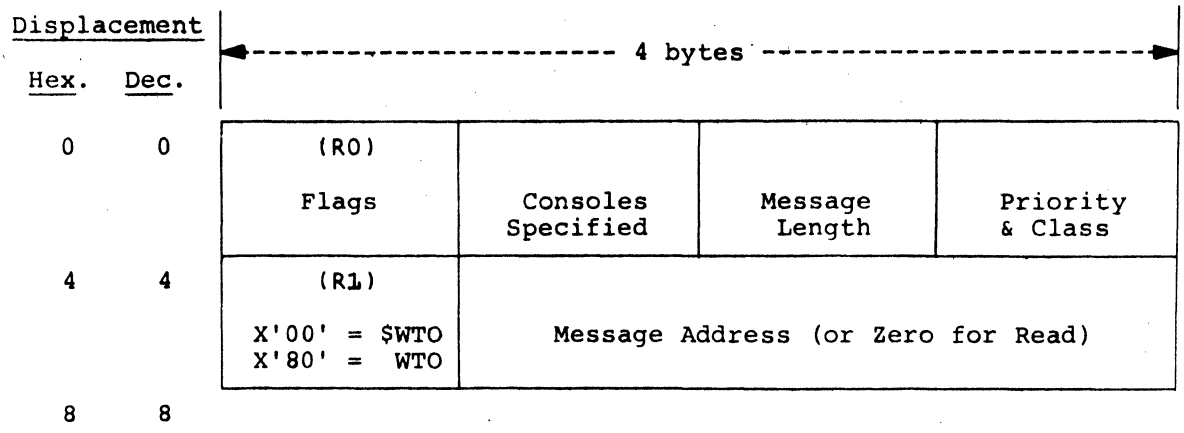
5.13.3 LOG QUEUING ROUTINE - PROGRAM LOGIC

This routine places a console buffer at the end of the queue of messages to be processed by the HASP Log Processor and $POSTs that processor.

5.13.4 CONSOLE BUFFER FREEING ROUTINE - PROGRAM LOGIC

This routine places the console buffer in the free queue. The Attention Processor's PCE is examined to determine if the Attention Processor is $WAITing for a console buffer, and if it is, the Attention Processor is $POSTed and exit is made. If the Attention Processor is not $WAITing, the $WTORQUE is tested and the first task found is POSTed. If no tasks are waiting, the HASP Event Control Field is $POSTed and exit is made.

Figure 5.13.1 -- CONSOLE BUFFERING ROUTINE PARAMETER REGISTERS



5.14 INPUT/OUTPUT ERROR LOGGING ROUTINE

5.14.1 Input/Output Error Logging Routine -- General Description

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.

5.14.2 Input/Output Error Logging Routine — Program Logic

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 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 format of the message describing a direct-access error is as follows:

I/O ERROR ON SPOOLn uuu,cc,ssss,iiii,bbcchhr

where:

n	— identifies the SPOOL disk in error
uuu	— unit address of disk
cc	— channel command code being executed
ssss	— channel status code
iiii	— unit sense information
bbcchhr	— track address as follows:
	bb — bin (always zero)
	cc — cylinder
	hh — head
	r — record

The format of the message describing a line error is as follows:

I/O ERROR ON LINEm uuu,cc,ssss,iirr,ttee

where:

m	— line number
uuu	— unit address of line
cc	— channel command code being executed
ssss	— channel status code
ii	— unit sense information

where:

rr	— STR - sense information
	BSC - terminal response
tt	— internal sequence and command code
ee	— STR - always blank
	BSC - expected response

5.15 REMOTE TERMINAL ACCESS METHOD (RTAM)

5.15.1 Remote Terminal Access Method -- General Description

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 synchronizes very closely with RTAM through a series of subroutines, the more important ones, of which, are briefly described below.

5.15.2 Remote Terminal Access Method -- Program Logic

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. The primary subroutines are discussed in Section 5.15.3 below.

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.

5.15.3 Remote Terminal Access Method -- Subroutines

This section describes the primary subroutines used by the Remote Terminal Access Method and the MULTI-LEAVING Line Manager.

MSIGNON -- Sign-On Card Processor

This subroutine is passed the address of a /*SIGNON card in register "R1". If the line from which the Sign-On Card was read was defined to be a "leased" line, the Sign-On Card is ignored and the subroutine returns immediately. If the line is a "dial" line, the MABORT and MDISCON subroutines are called to disconnect any other remote which may have been attached to

this line. The password is then checked and if not valid, an error message is issued and the subroutine returns. If the password is valid the specified Remote Terminal's DCT's 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 and a confirmation message is issued.

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 upon this value and returns. Figure 5.15.1 depicts the various CCW sequences which can be constructed by the subroutine.

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 insuring maximum line throughput. The subroutine first establishes the status of every processor currently active on the MULTI-LEAVING line. Then, based upon 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 either transmits an ACK0 to the terminal, transmits a text buffer to the terminal, or initiates a one-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 2770/2780 parity check (ACK0-ACK1) conversion.

Figure 5.15.1 — HASP Remote Terminal CCW Sequences

STR Hardware Terminal Prepare Sequence (code=4)

CCW	COMMAND	DATA ADDRESS	FLAGS	INTERNAL CODE	BYTE COUNT
IOBCCW1	DISABLE	0	60	40	1
IOBCCW2	SET MODE	LCBMCB	60	41	2
IOBCCW3	ENABLE	0	60	42	1
IOBCCW4	TEST SYNCH	0	60	43	15
IOBCCW5	SEND INQUIRY	0	20	4A	6

STR CPU Terminal Prepare Sequence (code=5)

CCW	COMMAND	DATA ADDRESS	FLAGS	INTERNAL CODE	BYTE COUNT
IOBCCW1	DISABLE	0	60	50	1
IOBCCW2	SET MODE	LCBMCB	60	51	2
IOBCCW3	ENABLE	0	60	52	1
IOBCCW4	TEST SYNCH	0	60	53	15
IOBCCW5	SEND EOT	0	60	5B	6
IOBCCW6	PREPARE	0	60	57	1
IOBCCW7	READ	TPBUFST	20	54	&TPBFSIZ

Figure 5.15.1 (continued) — HASP Remote Terminal CCW Sequences

STR Read Sequence (code=0: Hardware; code=1: CPU)

CCW	COMMAND	DATA ADDRESS	FLAGS	INTERNAL CODE	BYTE COUNT
IOBCCW1	TEST SYNCH	0	60	03/13	15
IOBCCW2	PREPARE	0	60	07/17	1
IOBCCW3	READ	TPBUFST	20	04/14	&TPBFSIZ
IOBCCW4	STEP COUNT	0	60	00/10	1
IOBCCW5	ERROR	0	60	00/10	1
IOBCCW6	TIC	IOBCCW3	00	00/10	0

STR Write Sequence (code=2: Hardware; code=3: CPU)

CCW	COMMAND	DATA ADDRESS	FLAGS	INTERNAL CODE	BYTE COUNT
IOBCCW1	TEST SYNCH	0	60	23/33	15
IOBCCW2	SEND INQUIRY	0	60	2A/3A	6
IOBCCW3	WRITE	TPBUFST	20	28/38	*-*

Figure 5.15.1 (continued) — 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	LCBRCB	20	C6	2

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

Figure 5.15.1 (continued) — HASP Remote Terminal CCW Sequences

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	A8	*-*
IOBCCW7	WRITE	METBSEQ	60	A8	2
IOBCCW8	READ	LCBRCB	20	A5	2

5.16 OVERLAY SERVICE ROUTINES

5.16.1 Overlay Service - General Description

These routines, together with the Overlay Roll Processor described in Section 4.20, 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, described in Sections 10.2.2 and 6.3, 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.

Actions necessary to initialize HASP Overlay Service are contained in module HASPINIT and are described in Section 6.1.2.

See Sections 8.3.3, 9.7, and 12.14 for descriptions of Overlay Area(s) format, macros mentioned above, and coding rules relating to use of overlay routines.

5.16.2 $LINK Service - Program Logic

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, whose 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, 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 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 (see 5.16.9). 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 (see 5.16.8) 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 (see 5.16.9).

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, as described in Section 4.20, or by the OEXIT subroutine, as described in 5.16.7.

5.16.3 $LOAD Service - Program Logic

$LOAD shares almost all logic with $LINK (see 5.16.2). 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 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 with the routine in an actual or theoretical area, addressable by BASE3 as with $LINK.

5.16.4 $XCTL Service - Program Logic

$XCTL logic shares almost all logic with $LINK (see 5.16.2). 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 (see 5.16.7) to remove the calling Processor's PCE from the chain of users of the current overlay routine.

5.16.5 $RETURN Service - Program Logic

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 (see 5.16.7) to remove caller's PCE from the chain of users of the current overlay routine.

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

5.16.6 $DELETE Service - Program Logic

$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.

5.16.7 OEXIT Subroutine - Program Logic

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 (see 5.16.2) is inspected. If PCE(s) are waiting, the top priority group of one or more requesting the same overlay routine is de-queued, the address of the first such PCE is placed in register "R1", and OEXIT simply falls through to the OLOD subroutine (5.16.8), which eventually returns to the caller of OEXIT.

5.16.8 OLOD Subroutine - Program Logic

This subroutine is used by service routines for $LINK, $LOAD, $XCTL; by the Overlay Roll Processor (see Section 4.20); and indirectly by users of the OEXIT subroutine (5.16.7). 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 which 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 (see 5.8) 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. See Section 6.1.2 for initialization of the Overlay DCT and data set.

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 (see 5.16.9). OLOD then returns to its caller or caller of OEXIT.

5.16.9 Overlay $ASYNC Exit - Program Logic

This routine is entered when under control of the Asynchronous Input/Output Processor ($ASYNC) PCE (see Section 4.8) an overlay read operation (started by OLOD subroutine, see 5.16.8) 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 re-entry address ("R15", now stored in PCER15) is absolutized 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 (on OACEPCE chain) re-entry 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.

If &OREPSIZ is set to zero, this Exit returns to $ASYNC. Otherwise, the Overlay REP storage area is examined to see if any REPs were read during HASP Initialization (see 6.4) which may apply to

this overlay routine. 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 memory 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.

(The remainder of this page intentionally left blank.)

6.0 MISCELLANEOUS

This section contains detailed internal information about miscellaneous routines imbedded in or involved with the HASP System and is intended primarily for use by systems programmers.

6.1 HASP INITIALIZATION

6.1.1 HASP Initialization - General Description

The purpose of HASP initialization is to initialize for HASP job processing. Initialization builds the required control blocks and makes modifications to the Operating System nucleus which allows HASP to monitor the execution of jobs.

HASP 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 which are entered after a "cold" start will be processed. A "cold" start does not have any requirements as to configuration except as defined in the HASP generation parameters. A "warm" start is a restart. Checkpointed information is read from the SPOOL1 volume and the queued jobs and data from the last processing are recovered. This type of start requires, as a minimum, that the SPOOL volumes that were used during the previous execution be on-line. Extra SPOOL volumes, up to a total of &NUMDA volumes, may be added.

6.1.2 HASP Initialization - Program Logic

Initialization begins with the issuing of the HASP Supervisor Call which turns control over to the HASP Initialization SVC Routine. On return the HASP task will be in the Supervisor State with protect key of zero. Register 1 points to a list of resolved nucleus addresses and a return point for resetting HASP to problem state. These addresses are moved into the HASP Communication Table (HCT) for later use by the system.

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 used for HASP SPOOLING. The HASP Initialization WTOR is then displayed via OS WTOR facilities. Initialization then waits for the operator to respond with the desired options. The options are then 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 JOB Section of the HASP Operator's Guide.)

The HASP REP routine (described in Section 6.4) is entered for optional alteration of the resident portions of the Operating System or HASP (resident or overlay control sections).

Preparation Of Overlay Service

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

The overlay data set is described by a DD card having ddname of 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.

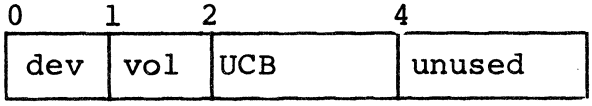
The number of tracks/cylinder and extent are used to compute a beginning absolute TT of the overlay data set, which is stored in the Overlay DCT for later use by the OLOD subroutine (see Section 5.16.8).

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 sixth volume serial character are saved in a temporary workarea 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 six-byte entries (NSPOOLL1) describing the direct-access devices upon which Spool disks are mounted, of which each entry appears as follows:



where:

dev is the low-order byte of the direct-access device type;

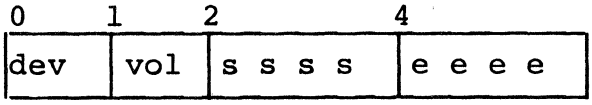
vol is the low-order byte of the volume serial number; and

UCB is 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 track of this data set (the first two tracks if &JITSIZE is not zero); 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, the track group map, printer checkpoint information, miscellaneous status information (including direct access checkpoint information) and, optionally, the job information table (JIT). The direct access checkpoint information, $DACKPT, consists of &NUMDA six-byte entries of the following form:



where:

dev is the low-order byte of the direct-access device type;

vol is the low-order byte of the volume serial number;

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

eeee is 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 insures 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, or 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 core 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.

After relocation, NGWARM scans the job queue to check the busy bit of each active entry and to reset certain flags. If a busy bit is on, NGWARM turns it off and issues a WTO to inform the operator that the job was reading, executing, printing, or punching. Additionally, if the job was reading, NGWARM uses HASP queue management routine $QREM to delete the job's queue entry.

At the end of the job queue, NGWARM gives control to NGEXIT, which 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; and gives control to NINITWTO.

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 core in which to build a formatting channel program and data, builds them, and formats each track by calling NGEXCP, which merely issues an EXCP and 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 then 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 above.

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.) It checks for SPOOL1 and quiesces HASP if SPOOL1 is not found. Then NGDEBSET processes the Spool volumes.

For each volume, NGDEBSET calculates number of records per track using information from the device characteristics table IECZDTAB in the OS nucleus and the formula given with the DEVTYPE macroinstruction in the OS System Programmer's Guide. Then it sets up certain information in an entry of the Table of Extent Data (TED).

Then, 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 (see 5.16.8) 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

The OS UCBs are scanned for unit record devices. Devices which are on-line on a DUAL Processor Model 65 system, have OS scheduled I/O activity, or answer positively to a TIO instruction are considered real devices. Otherwise the devices are considered pseudo devices.

PSEUDO DEVICE INITIALIZATION - Pseudo devices are initialized by flagging the UCB for later identification by the HASP Execution Processor SVC 0 intercept routines and are varied on-line.

Pseudo 2540 reader, 1442 special forms punch, and 1443 special forms printer devices are especially noted and counts are maintained for the HASP Execution Processor Device Allocation Routine. Pseudo 1403 Printer UCS feature is removed from the UCB. Pseudo 2520 devices are identified and matched with an internal reader INTRDR Device Control Table which is initialized for processing.

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, 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 and (as appropriate) HASP console UCB attention index values are set to four allowing HASP to recognize the readying of the readers or the pressing of the enter key(s). (At least one HASP console device is reserved for a 1052 type of device.) If more real unit record devices of each particular type are found than available DCTs, an error message is displayed and the additional devices are ignored.

Control is then passed to the Remote Job Entry or console initialization routines as appropriate via a $XCTL macro.

Remote Job Entry Initialization - HASPIOVR

LINE INITIALIZATION - The OS UCBs are scanned for Synchronous Communication Adapter devices. The UCBs found are first matched with one or more DCT and corresponding line descriptions (LINEmm HASP Generation Parameters). Any DCT with a line description which specifically designates the UCB will be initialized for the UCB. If no line description designates the UCB, tests are made to determine if the adapter is physically on-line and, if so, a DCT with a line description with "***" specified 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 HASP Generation 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 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 console (local) 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 insure that the space required is within the allocated space. Control is given to the console (local) initialization routine by $XCTL.

Console Initialization - HASPIOVB

OS CONSOLE INITIALIZATION - Information is extracted from the OS UCM and the Console processor is made ready for interfacing with OS.

HASP CONSOLE INITIALIZATION - The Console DCTs are initialized for HASP console support. Each DCT that was matched with a UCB by the unit record initialization routine is initialized for I/O processing. The corresponding authorization for each console is converted to console restrictions and set into the DCT. The operator command $S console is simulated.

Control is passed to the intercept initialization routine via $XCTL.

Intercepts Initialization - HASPIOVC

The following intercepts are made in accordance with the type of the HOST Operating System MVT or MFT and the HASP generation options as follows:

SVC 0 - EXCP interface used for control of user I/O
SVC 6 - LINK interface used to recognize events within OS
SVC 7 - XCTL interface used to recognize events within OS and to interface with OS console support
SVC 35 - WTO interface for console support
SVC 36 - WTL interface for write to log support.

Start Initiator, reader, and writer (optional) commands are issued which start the procedures contained on SYS1.PROCLIB. The reader will be directed to the pseudo device &RDR and the writer will be directed (if started) to the pseudo device &WTR.

In the event the HASP writer is selected in lieu of the OS writer, the HASP writer module "HASPWTR" is attached. If OS console support is selected, the HASP communications task is attached, via the attaching of module "HASPBR1" which enters the console processor.

Control is passed to the HASP buffer building routine via $XCTL.

Buffer Build - HASPIOVD

An OS variable GETMAIN is issued to obtain storage for the alternate buffer pool from the hierarchy as designated by the &BUFHICH generation parameter. The actual amount of core is reduced by the amount of reserved core (&RESCORE*1024) and used to determine the number of buffers which may be created in the alternate buffer pool along with the actual amount of storage needed. Extra core 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 &MINBUF ≥ &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.

The operator is then asked to ENTER HASP REQUESTS (optional) and the ROOT segment of HASP INITIALIZATION is entered via the $RETURN macro.

Activation Of Normal Processing

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.

6.2 HASP INITIALIZATION SVC ROUTINE

6.2.1 HASP Initialization SVC Routine - General Description

This program is a Type-I SVC routine which resides in the Operating System Nucleus and provides the following basic functions:

1. For HASP:

   - To give HASP a zero storage protection key.
   - To place HASP in supervisor state.
   - To return the address of key symbols in the nucleus which are required for HASP processing.
   - To guard against recursive entries in order to prohibit multiple copies of HASP from being initiated.
   - To provide 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.

2. For the HASP Reader/Interpreter Appendage:

   - To place the HASP JCL Exit routine in supervisor state.
   - To return the left half of the PSW which was in use when the SVC was invoked.

3. For the non-HASP program:

   - To give an indication to any other program as to whether HASP is currently active or not.

6.2.2 HASP Initialization SVC Routine - Program Logic

This program is a Type-I OS SVC routine. It must be link-edited with the nucleus to resolve the external address constants required for HASP processing.

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", a test is made to determine if HASP has been invoked. If not, then this switch is set to indicate that HASP is now active and the left half of the PSW is saved for the "reset entry".

If HASP has been invoked, the protect key of the caller is interrogated. If this protect key is non-zero, the caller is ABENDed with an appropriate ABEND code.

The SVC OLD PSW is modified so that the return to HASP will place HASP in the supervisor state and give HASP a zero storage protection key. 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. At this time register 0 contains the left half of the PSW which was in use when the SVC was invoked.

One of the addresses in the nucleus address table is the address of the SVC reset routine. When this routine is entered, it resets the switch to indicate that HASP is no longer active. It then returns to the user by loading a PSW constructed by concatenating the left half of the original PSW with register 14.

6.3 HASP OVERLAY BUILD UTILITY

6.3.1 HASP OVERLAY BUILD - GENERAL DESCRIPTION

The purpose of this program is to process the object deck output from the ten primary HASP assemblies. 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 which allow changing the status of any overlayable CSECT from actual overlay to permanently resident and vice-versa.

The use of this program to install HASP (control cards, listings produced, etc.) is described in section 10.2.2.3, which should be read as background to this description. Overlay Services and Roll logic, and Overlay Programming Rules are described in sections 5.16, 4.20, and 12.14 respectively.

6.3.2 HASP OVERLAY BUILD - PROGRAM LOGIC

On initial entry, the time is sampled. A truncated "time-like" value is saved. This value will be placed into 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 encountered in the object decks. A listing header line is printed at the end of control card processing.

All objects 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/360 Loader PLM, Y28-6714, Figures 30-34; 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 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, and if 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 "O", 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 constructed as ESD items are encountered, under control of SYM table contents. It is a table of words, in order by ESID number. TXT 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 memory, 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 non-overlay CSECTs or overlays being made permanently resident is written un-altered 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 non-overlay CSECT, it is left un-altered. 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 in memory. If the Q constant exists in a non-overlay 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 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 1024 bytes in unmodified HASP). A listing line is printed for each overlay CSECT defined in the most recent deck, with its 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 signalled. 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 memory 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.

6.4 HASP REP ROUTINE

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

6.4.1 REP Card Format

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 digits) 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 memory 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 interpretted 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.

6.4.2 REP Routine - Program Logic

REP cards, as described in Section 6.4.1, are read from the card reader, whose address is given by the HASPGEN parameter $REPRDR, immediately after the operator replys 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. 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 memory address 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 non-blank, 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 memory address 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 non-zero assembly origin of the overlay CSECT is subtracted from and the loaded memory 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 memory 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 preceeding card. If they are not equal, the card is ignored. Otherwise, the card is considered

to be a continuation of the preceeding card and the patch value is applied at the next higher memory address or saved as appropriate.

If no area was reserved to save patch information for application to non-resident 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.

6.5 HASP ACCOUNTING ROUTINE

6.5.1 HASP Accounting Routine - General Description

The Accounting Routine accumulates statistics for each job at the completion of the Punch phase and produces the HASP Account Card (see Section 11) which is punched by the Punch Processor. This feature is optional and may be deleted at HASPGEN time.

6.5.2 HASP Accounting Routine - Program Logic

The HASP Accounting Routine is a separately assembled overlay segment which gains control at the end of the Punch phase. Its function is to construct an accounting card such that the Punch Processer can punch this card upon return.

Upon entry, the following registers contain the following information:

Register 1 - Address of the HASP Job Queue Entry Priority Byte.

Register 2 - Address of the Accounting Card Image Area.

Register 10 - Address of the HASP Job Control Table.

Register 14 - Return Address.

All registers must be saved and restored before return to HASP.

This routine blanks out the Accounting Card Image Area and then extracts information from the HASP Job Control Table and HASP Job Queue Entry and constructs the Accounting Card Image in the Accounting Card Image Area. Special consideration is made for the clock passing midnight. In such cases the elapsed time is negative and a correction factor (24 hours) must be added.

The accounting card which is normally punched when this routine returns to the punch processor may be deleted by setting the condition code to zero before returning.

6.6 HASP DUMP ROUTINES

HASP provides two dump routines which are optionally included as debugging aids. The HASP Dump Routine for printer formatted dumps and the HASP High Speed Dump to Tape Routine are discussed in the following sections.

6.6.1 HASP Dump Routine - General Description

The $Dump routine is available as a debugging aid (effective only if &DEBUG=YES and will, when the console PSW RESTART key is depressed, dump memory according to specified limits.

6.6.2 HASP Dump Routine - Program Logic

This routine gains control via the PSW RESTART key on the console. Upon activation of the key, a specially formatted HASP PSW is loaded from location HEX'0'. The format is HEX'0004000F' for the first word; where 0004 is the mask that allows only a machine check interrupt, and 000E is the address of the printer that is referenced in the routine. The second word will contain the address of the $Dump routine of HASP.

Once activated, $Dump will reference the low core address of HEX'30' for its beginning limit and HEX'34' for its ending limit. These limits default to values that will dump all of the memory unless the operator changes the limits prior to pushing the PSW RESTART key. If a change is desired, the limits should be entered at their respective locations in the following format: HEX'00XXXXXX'. Also, if an operator should care to change the limits while $Dump is activated, the routine will immediately note a change, will immediately stop the previous dump, and will start dumping memory within the new limits. It should be noted at this point that a machine check will destroy the limit values within their position in core. To avoid undetected machine checks, however, the dump program is, at all times, enabled for machine check interrupts.

The routine also allows the operator to route the printing to a printer with an address other than HEX'00E'. A change of the printer address in the HASP preformatted PSW, prior to activation of $Dump, will accomplish this.

At the normal end of the routine, the system will be placed in a "wait" state via the LPSW command. At this point in time, the registers will have been restored back to their values prior to $Dump and the default limits of $Dump will have been returned to their respective values.

6.6.3 HASP HIGH SPEED DUMP TO TAPE ROUTINE - GENERAL DESCRIPTION

The High Speed Dump to Tape Routine, available as an optional debugging aid, dumps all of main storage to tape for post processing by the IBM System/360 Operating System Service Aids program IMDPRDMP or an equivalent processor.

6.6.4 HASP HIGH SPEED DUMP TO TAPE ROUTINE - PROGRAM LOGIC

ENTRY TO IDMTAPE - If the system programmer sets the HASPGEN parameter &DMPTAPE to the address of an attached magnetic tape drive the High Speed Dump to Tape Routine (IDMTAPE) will be created to write on the specified drive when entered. Initialization will display on the operator's console the message "SET RESTART PSW TO 0004000000aaaaaa FOR TAPE DUMP" where "aaaaaa" is the address of the entry point IDMTAPE. This message not only verifies that the routine is present; it is sufficient information for the operator to manually activate the dump. Via the REP processing routine or via manual key entries the system programmer or operator is able to insert code within the system to detect errors and cause dumps by program entry to the routine simulating the loading of the requested PSW. (Example: set program new PSW as directed to dump on the first program check.)

IDMTAPE assumes that the device generated is an appropriate tape drive to use, that the tape is at load point ready for writing, and that the recording mode status is correctly set. If these conditions are not true unpredictable results will occur.

CHANGING THE TAPE ADDRESS - The halfword located in storage at IDMTAPE-4 contains the address of the tape drive upon which the routine will write when entered. This address may be altered manually to any other tape drive address as appropriate prior to executing the routine.

PROGRAM LOGIC - The general registers and selected fixed storage areas are saved in location 84 hexadecimal to be compatible with the OS Service Aids dump program.

HASP control section locator elements are moved into low storage adjacent to the fixed area information. These elements contain the last four characters of the "HASPxxxx" control section names of basic assembly modules. Following each CSECT identification is the address of the beginning of the identified CSECT. (HASP control sections may then be easily located in the dump after post processing.) Location 80 hexadecimal

is set to the negative of address 2048. Location 80 hexadecimal for 4 bytes is written followed by 2048 bytes of storage (first part of which contains saved data).

Each succeeding record is written by adding the address 2048 to the address in location 80 (hex), storing the memory protect key in the high byte, and writing 2052 bytes of storage (four from location 80 (hex) and 2048 from the designated address). When the address is greater than zero the program new PSW is set to provide an end of storage exit which, when entered, will cause the writing of EOF, a rewind unload, and the loading of a wait state PSW.

7.0 HASPGEN AND RMTGEN PARAMETERS

This section describes the parameters used to specify the HASP System, HASP MULTI-LEAVING Remote Terminal Programs, and the System/360 Model 20 STR Remote Terminal Program.

Generation of the HASP System is called HASPGEN, and generation of the HASP MULTI-LEAVING Remote Terminal Programs and the System/360 Model 20 STR Program for HASP Remote Job Entry is called RMTGEN. Both generation processes are described in Section 10.

7.1 HASPGEN PARAMETERS

Generation of a HASP System involves specification of certain parameters, called HASPGEN parameters. With these parameters, the installation system programmer specifies the characteristics of the System/360 or System/370 with which he will use HASP and the optional HASP features he wishes to be included in the generated HASP System.

The following pages describe the HASPGEN parameters. For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation and refer to related HASPGEN parameters.

The HASPGEN parameters are given in alphabetical order (neglecting the first character if it is & or $) except for parameter $$x, which appears last.

&ACCTNG

Explanation: Variable symbol &ACCTNG specifies the HASP job accounting option. If it is specified as YES, HASP will call the HASP accounting routine and punch a HASP accounting card for each job processed by HASP. The specification must be either YES or NO.

Default: &ACCTNG=YES

Notes:
1. The HASP accounting routine and the HASP accounting card are discussed in other sections of this manual.
2. If &NUMPUNS=0, parameter &ACCTNG should be set to NO.

&AUTORDR

Explanation: Variable symbol &AUTORDR specifies the inclusion or exclusion of code in HASP to recognize automatically when a physical card reader available to HASP becomes ready. The specification must be either YES or NO.

Default: &AUTORDR=YES

Notes:

1. If &AUTORDR=NO, HASP's physical card readers remain in the INACTIVE state when they become ready; the operator must issue a $SRDRn command to cause HASP to begin reading cards from READERn.

&BSCCPU

Explanation: Variable symbol &BSCCPU specifies inclusion or exclusion in the HASP Remote Terminal Access Method of support for HASP MULTI-LEAVING Remote Job Entry.

Default: &BSCCPU=NO

&BSC2770

Explanation: Variable symbol &BSC2770 specifies inclusion or exclusion in the HASP Remote Terminal Access Method of Remote Job Entry support for the 2770 Data Communication System. The specification must be either YES or NO.

Default: &BSC2770=NO

&BSC2780

Explanation: Variable symbol &BSC2780 specifies inclusion or exclusion in the HASP Remote Terminal Access Method of Remote Job Entry support for the 2780 Data Transmission Terminal. The specification must be either YES or NO.

Default: &BSC2780=NO

&BSHPRSU

Explanation: Variable symbol &BSHPRSU specifies inclusion or exclusion of the HASP Remote Job Entry Printer Interrupt feature for binary synchronous hardware terminals. If this feature is included, the Remote Terminal operator may interrupt printing to transmit jobs or HASP commands to HASP. The specification must be either YES or NO.

Default: &BSHPRSU=YES

Notes:

1. If &BSHPRSU=YES, HASP will recognize certain control characters from the binary synchronous hardware terminal which indicate that the printer has stopped. The HASP Remote Terminal Operator's Manual for hardware terminals contains more information.

&BSVBOPT

Explanation: Variable symbol &BSVBOPT specifies inclusion or exclusion in the HASP Remote Terminal Access Method of code to recognize an EM (End of Media) punch in card images transmitted nontransparently by the 2780 Data Transmission Terminal. The specification must be either YES or NO.

Default: &BSVBOPT=NO

&BUFHICH

Explanation: Variable symbol &BUFHICH specifies the storage hierarchy for which HASP initialization will issue a GETMAIN in an attempt to get more than &NUMBUF buffers for HASP. The specification must be either 0 or 1.

Default: &BUFHICH=1

Notes:
1. &BUFHICH has meaning only with OS storage hierarchy support.
2. See HASPGEN parameters &NUMBUF, &MINBUF and &RESCORE for additional information concerning the use of this parameter.

&BUFSIZE

Explanation: Variable symbol &BUFSIZE specifies the size in bytes of each HASP buffer. If the value specified is not a multiple of eight, HASPGEN will adjust it upward to a multiple of eight. The specification must be an integer not larger than the track size of any SPOOL device and not smaller than the number given by

300+2*&NUMDA*&NUMTGV/8+5*S+8*F

where S = maximum allowable number of SYSOUT specifications per job, including special forms requests

F = maximum allowable number of special forms requests per job.

Default: &BUFSIZE=688

Notes:

1. The above formula is the approximate size of a HASP control block called the JCT. &BUFSIZE is the data length of each physical record on the portion of a SPOOL volume used by HASP, and JCTs are written on SPOOL volumes. A JCT's initial size is about 300 + 2*&NUMDA*&NUMTGV/8; it increases in size as a job is being executed. When HASP recognizes a job step change, the JCT is increased in size by five bytes for each non-special-forms SYSOUT data set, or by thirteen bytes for each special-forms SYSOUT data set, that was written upon since the previously-recognized step change. But if an increase of five (or thirteen) bytes would cause the JCT size to become greater than &BUFSIZE, HASP instead writes to operator the message

   JCT OVERFLOW--OUTPUT LOST

   for the SYSOUT data set, and its output is lost.
2. The default &BUFSIZE of 688 is optimized for the 2314 track size. If &NUMTGV and &NUMDA are left at their default values of 400 and 2, the &BUFSIZE default allows a maximum of 37 (no special forms) and a minimum of 14 (all special forms) SYSOUT data sets per job. A good value of &BUFSIZE optimized for the 3330 would be 736.

$CKPTIME

Explanation: Ordinary symbol $CKPTIME specifies the interval, in seconds, at which certain HASP information will be checkpointed for warm start purposes.

Default: $CKPTIME=60

Notes:
1. The time interval specified is a maximum. Checkpoints are also taken when a job changes its status in the HASP job queue.
2. The section of this manual describing the HASP checkpoint processor describes the checkpoint information.

&CLS(n)

Explanation: Subscripted variable symbols &CLS(n) specify HASP job classes. The nth HASP logical partition may select for OS execution a job from the HASP job queue only if the job's class (specified by the user in the CLASS=parameter of the JOB card, or defaulted to A) is one of the characters specified in the &CLS(n) parameter or specified by the operator in the set command, $T Imm,list (where &PID(n)=mm). Each specification must be a 1- to 8-character string of valid HASP job classes. The same HASP job class may be specified in two or more specifications.

Default:
```
&CLS(1)=A
&CLS(2)=BA
&CLS(3)=CBA
&CLS(4)=DCBA
&CLS(5)=EDCBA
&CLS(6)=FEDCBA
&CLS(7)=GFEDCBA
&CLS(8)=HGFEDCBA
&CLS(9)=IHGFEDCB
&CLS(10)=JIHGFEDC
&CLS(11)=KJIHGFED
&CLS(12)=LKJIHGFE
&CLS(13)=MLKJIHGF
&CLS(14)=NMLKJIHG
&CLS(15)=ONMLKJIH
```

Notes:
1. Only the first &MAXPART specifications, &CLS(1) through &CLS(&MAXPART), will be used.
2. If &MAXCLAS is specified less than 8, only the first &MAXCLAS characters of each specification &CLS(n) will be used.

&CONAUTH

Explanation: Variable symbol &CONAUTH specifies the HASP command authorization of each of the eight possible HASP physical consoles when HASP console support is included in the generated HASP system. The specification must be a string of up to eight numeric digits, each of which may range from 0 to 7 and is the sum of the desired authorizations for its respective console (leftmost digit for CONSOLE 1, next digit for CONSOLE2, etc.) as follows:

1 - System Control (including OS) Commands
2 - Device Control Commands
4 - Job Control Commands.

Default: &CONAUTH=77777777

Notes:
1. Any HASP console's command authorization may be changed from a console with System Control Command authority.
2. If &NUMCONS=0, parameter &CONAUTH is not used.
3. All HASP consoles are authorized for the issuance of Display Commands.

&DEBUG

Explanation: Variable symbol &DEBUG specifies inclusion or exclusion of debugging code in the generated HASP system. The specification must be either YES or NO.

Default: &DEBUG=NO

Notes:
1. The &DEBUG option is independent of the &TRACE option.
2. If &DEBUG is specified as YES, HASP includes, in addition to other debugging code, a core dump rouine.

$DELAYCT

Explanation: Ordinary symbol $DELAYCT specifies a delay factor to be applied by the HASP Remote Terminal Access Method when transmitting to a Multi-Leaving System/360 Model 20 Sub-Model 2 Remote Terminal over a high-speed (19,200 baud or greater) teleprocessing line, to avoid the possibility of certain line errors. The specification must be an integer greater than zero. Recommended values for some central CPUs are:

```
Model 91 - 4000
Model 85 - 3000
Model 75 -  500
Model 65 -  256
Model 50 -  100
Model 40 -    1
```

Values for other CPUs may be interpolated based on CPU speed.

Default: $DELAYCT=256

&DMPTAPE

Explanation: Variable symbol &DMPTAPE specifies a unit address to be used with the HASP dump program. The specification must be a valid unit address. If the specification is not 000, the address is assumed to be that of a tape drive and the generated HASP system will include a dump-to-tape program for that tape drive. The tape produced by this program may be printed using FE Service Aid IMDPRDMP.

Default: &DMPTAPE=000

Notes:

1. This parameter does not affect inclusion of the HASP dump-to-printer program, which is always assembled if &DEBUG=YES. To use the dump-to-tape program the operator must set location 0 as indicated in an operator message produced by HASP initialization, make the tape drive ready for writing, and push PSW RESTART.
2. This facility is provided as an aid to the system programmer and may not operate correctly in all environments or with other than the IMDPRDMP program provided with Release 19 of OS. No maintenance will be provided for this facility. Installations unable to utilize this support as distributed should utilize the service aid IMDSADMP to produce dump tapes.

$ESTIME

Explanation: Ordinary symbol $ESTIME specifies the default estimated execution time, in minutes, for a job. The specification must be an integer greater than zero.

Default: $ESTIME=2

Notes:
1. If a user does not specify in the accounting field of his job card a value for estimated execution time, the value $ESTIME is used.
2. All timings performed by HASP are in real time. The timing for estimated execution time begins when HASP allows its OS Reader/ Interpreter to start reading the job.

$ESTLNCT

Explanation: Ordinary symbol $ESTLNCT specifies the default estimated print line count, in thousands of lines, for a job. The specification must be an integer greater than zero.

Default: $ESTLNCT=2

Notes:

1. If a user does not specify in the accounting field of his job card a value for estimated print line count, the value $ESTLNCT is used.

$ESTPUN

Explanation: Ordinary symbol $ESTPUN specifies the default estimated punched card count, in cards, for a job. The specification must be an integer greater than zero.

Default: $ESTPUN=100

Notes:

1. If a user does not specify in the accounting field of his job card a value for estimated card count, the value $ESTPUN is used.

&INITSVC

Explanation: Variable symbol &INITSVC specifies the SVC number of the HASP SVC. The specification must be an integer between 200 and 255, inclusive.

Default: &INITSVC=255

Notes:

1. The specification for &INITSVC must correspond to a use at SYSGEN time of the OS SYSGEN macro-instruction SVCTABLE. The HASP SVC is a type-1 SVC and must be included in the OS Nucleus before HASP can be used. After HASPGEN has completed, the object deck for the HASP SVC is member HASPSVC of partitioned data set SYS1.HASPOBJ. For further discussion, see the section on installing HASP.

&JITSIZE

Explanation: Variable symbol &JITSIZE specifies the number of bytes per entry of the HASP job information table. The specification must be an integer greater than or equal to 8, or equal to 0, but never so large that the value

&MAXJOBS*&JITSIZE

is greater than the track size of the device upon which SPOOL1 is mounted.

If &JITSIZE=0, the HASP command $D'jobname' will be inoperative.

Default: &JITSIZE=0

Notes:

1. The recommended specifications for &JITSIZE are 0 and 8. If &JITSIZE is set greater than 8, the additional space generated in the job information table will not be used by HASP.
2. If &JITSIZE is specified greater than zero, the job information table will be checkpointed on SPOOL1 whenever it changes.

$LINECT

Explanation: Ordinary symbol $LINECT specifies the default maximum number of lines to be printed per page of a job's printed output.

Default: $LINECT=61

Notes:
1. If a user does not specify in the accounting field of his job card a value for line count, the value $LINECT is used.

LINEmm

Explanation: Ordinary symbols LINEmm specify the characteristics of teleprocessing lines to be used by HASP Remote Job Entry. Lines must be defined consecutively, starting with LINE01. Each specification must be a 5-character string of the form

LINEmm=aaalc

where the letters represent the following:

Code Letters	Range	Description
mm	01-99	Line Number
aaa	000-FFF	STR or BSC Adapter Address (See Note 2)
l	0-5	Line Description as follows:
		STR Lines
		0 = Interface A - 2 wire Half-Duplex 1 = Interface A - 4 wire Half-Duplex 2 = Interface A - Full-Duplex 3 = Interface B - 2 wire Half-Duplex 4 = Interface B - 4 wire Half-Duplex 5 = Interface B - Full-Duplex
l	0-5	BSC Lines
		0 = Interface A - Half-Duplex (1200-9600 baud) 1 = Interface A - Full-Duplex (1200-9600 baud) 2 = Interface A - Full Duplex (19.2-230.4 k-baud) 3 = Interface B - Half-Duplex (1200-9600 baud) 4 = Interface B - Full-Duplex (1200-9600 baud) 5 = Interface B - Full-Duplex (19.2-230.4 k-baud)
c		Clock/Code as follows:
	0-3	STR Lines
		0 = Internal Clock X 1 = Internal Clock Y 2 = Internal Clock Z 3 = External Clock

Code Letters	Range	Description
	0-7	BSC Lines

0 = Code A - EBCDIC - No Transparency
1 = Code A - EBCDIC - Transparency
2 = Code A - USASCII - No Transparency
3 = Code A - USASCII - Transparency
4 = Code B - EBCDIC - No Transparency
5 = Code B - EBCDIC - Transparency
6 = Code B - USASCII - No Transparency
7 = Code B - USASCII - Transparency

Default: LINEmm=***01

Notes:

1. Parameter &NUMLNES must specify the number of specifications LINEmm to be included in the generated HASP system.
2. The unit address aaa may be specified as ***. HASP initialization will assign unit addresses to lines whose unit addresses are specified as *** by scanning the OS UCBs. A teleprocessing UCB whose device type field specifies a 2701 STR adapter, a 2701 BSC adapter, or a 2703 BSC adapter will be recognized as a UCB defining a line. If the unit address of such a UCB is not specified explicitly in any of the first &NUMLNES line definitions LINEmm, HASP initialization will assign the UCB to the first line number whose unit address is specified *** and will change the *** to the EBCDIC address specified in the UCB, unless no line definition remains whose unit address is ***, or if (except for M65MP and a 2-CPU multiprocessor) a TIO shows the line not operational, or if (for M65MP and a 2-CPU multiprocessor) the UCB is marked off-line; in that case HASP will not use the line.
3. If a line specification LINEmm designates USASCII, that line cannot be used with any but 2770 and 2780 USASCII terminals. HASP will translate each record it receives into EBCDIC, and each record it transmits into USASCII before transmission.

&LOGOPT

Explanation: Variable symbol &LOGOPT specifies inclusion or exclusion of code to support the HASP System Log feature. The specification should be either YES or NO.

Default: &LOGOPT=YES

Notes:

1. The HASP System Log is a listing included in each user's output of console messages that were produced during processing of the job and (unless &NUMCONS=0) of replies to WTORs issued during processing of the job.

&MAXCLAS

Explanation: Variable symbol &MAXCLAS specifies the maximum number of job classes which may be specified via the HASP command $T In,list for a HASP logical partition. The specification must be an integer from 1 to 64, inclusive.

Default: &MAXCLAS=8

Notes:
1. If &MAXCLAS is specified as less than 8, then no more than &MAXCLAS characters may be specified for each of the parameters &CLS(n).

&MAXJOBS

Explanation: Variable symbol &MAXJOBS specifies the maximum number of jobs that can be in the HASP System at any given time. The specification must be an integer greater than zero.

Default: &MAXJOBS=100

Notes:

1. This variable does not affect the range of HASP job numbers, which is 1 to 999.
2. This variable strongly influences the size of the HASP checkpoint record(s). The size of the first checkpoint record is

   ```
   16*(&MAXJOBS+&NUMPRTS+&NUMTPPR)+
   &NUMDA*((&NUMTGV+7)/8)+40.
   ```

   The size of the second checkpoint record is

   ```
   &MAXJOBS*&JITSIZE.
   ```

   If either checkpoint is longer than the track size of the device on which SPOOL1 is mounted, HASP will not warm start correctly.

&MAXPART

Explanation: Variable symbol &MAXPART specifies, for both MFT and MVT, the number of HASP logical partitions to be defined. The specification must be an integer between 1 and 15, inclusive.

Default: &MAXPART=&MAXXEQS

Notes:
1. The nth logical partition is further defined by the specifications &PRI(n), &OSC(n), and &CLS(n).

&MAXXEQS

Explanation: Variable symbol &MAXXEQS specifies the maximum number of jobs which may concurrently be active in the HASP Execution phase. The specification must be an integer greater than zero.

Default: &MAXXEQS=3

Notes:
1. See also &MAXPART, the variable which determines the number of HASP logical partitions.

&MINBUF

Explanation: This variable is provided to allow installations, which depend on the dynamic buffer construction feature of HASP, to detect the condition where sufficient buffers for proper operation cannot be obtained. The specification should be an integer value representing the minimum number of buffers determined as necessary for the installation (see &NUMBUF).

Default: &MINBUF= 3*&MAXXEQS+2*&NUMRDRS
+&NUMINRS+2*&NUMPRTS+&NUMPUNS
+&NUMTPBF

Notes:

1. HASP will automatically attempt to utilize, via a variable GETMAIN, any free space in its region or partition (hierarchy indicated by &BUFHICH) as additional buffers. If the number of buffers so obtained when added to the variable &NUMBUF is less than the value of &MINBUF the warning message
&MINBUF BUFFERS NOT AVAILABLE
will be issued during HASP initialization and processing will continue.
2. Since the changing of HASPGEN options, local modifications and/or OS changes can affect the number of HASP buffers, proper setting of this variable can prevent a possible undetected performance degradation.
3. See the description of HASPGEN parameters &NUMBUF, &BUFHICH and &RESCORE for related information.

&MLBFSIZ

Explanation: Variable symbol &MLBFSIZ specifies the size in bytes of each HASP Multi-Leaving buffer. The specification for &MLBFSIZ must be a positive integer no larger than &TPBFSIZ.

Default: &MLBFSIZ=&TPBFSIZ

Notes:
1. The value specified for &MLBFSIZ automatically becomes the Multi-Leaving buffer size in each HASP Multi-Leaving Remote Terminal program.
2. The defaults given for &TPBFSIZ and &MLBFSIZ are judged to be practical minimum values and to give suitable performance for medium-speed teleprocessing lines.

&MONINTV

Explanation: Variable symbol &MONINTV specifies the interval in seconds at which the HASP Execution Task Monitor will examine CPU utilization characteristics and, if necessary, modify dynamically the order in the TCB chain, of all HASP-controlled job step tasks which fit Execution Task Monitor criteria. The specification should be an integer between 0 and 10 inclusive. If &MONINTV is specified as zero, the Execution Task Monitor is excluded from the generated HASP system.

Default: &MONINTV=5

Notes:
1. See also parameters &XZPRTY (for MVT), &XZMFTL, and &XZMFTH.

&NOPRCCW

Explanation: Variable symbol &NOPRCCW specifies the maximum number of channel command words per channel program for local printers.

Default: &NOPRCCW=15

&NOPUCCW

Explanation: Variable symbol &NOPUCCW specifies the maximum number of channel command words per channel program for local punches.

Default: &NOPUCCW=10

&NUMBUF

Explanation: This variable symbol indicates the number of INPUT/OUTPUT buffers to be included in the HASP load module and should normally be set by each installation, according to the formulae below, to reflect the total number of buffers required for proper operation of HASP. However, since HASP will automatically utilize free space in its region or partition to dynamically construct additional buffers, there are circumstances when &NUMBUF may be set to a value less than the actual number of buffers required for proper HASP operation. In this case, it is assumed that sufficient additional buffers will be dynamically obtained from free storage in the HASP region/partition to provide an adequate total number of buffers (see &MINBUF and &RESCORE). This facility could be used, for example, to allow additional buffers to reside in a storage hierarchy different from that of the HASP load module (see &BUFHICH) or to provide a HASP whose size (and performance and function) can be controlled by the setting of the region or partition size.

Default: &NUMBUF=15

Notes:

1. In order to utilize all the dynamic storage contained in the HASP load module for the initialization process, the value of &NUMBUF must never be less than the value
   1+6000/(80+&BUFSIZE)
2. Since all HASP buffers are maintained in a dynamic pool until required by an active function, installation should determine the appropriate number of buffers for HASP based on predicted simultaneity of the various functions required at the installation. The following indicates the number of buffers required for each logical function. A defined function which is inactive requires no buffers.

Each local input function	--2	
Each internal reader	--1	
Each Remote Input function	--1	
Each local print function	--2	(1 if $PRTBOPT=1)
Each remote print function	--1	(2 if $RPRBOPT=2)
Each local punch function	--1	(2 if $PUNBOPT=2)
Each remote punch function	--1	(2 if $RPUBOPT=2)
Each OS job execution	--2	(minimum value)

For performance reasons, additional buffers must be available to sustain periods of high SYSIN/SYSOUT activity by jobs being processed by OS. It is therefore recommended that additional buffers (beyond the requirements indicated above) be included corresponding to the value: 1+&MAXXEQS.

SEVERE PERFORMANCE AND/OR DEVICE DEGRADATION CAN OCCUR IN A SYSTEM CONTAINING INSUFFICIENT BUFFERS TO PERFORM THE REQUIRED FUNCTIONS.

3. To avoid a complete system failure caused by a buffer "lock-out" condition, the number of available buffers must never be less than the value

```
      &MAXXEQS+&NUMRDRS+&NUMTPES+1
     +&NUMPRTS*($PRTBOPT-1)
     +&NUMPUNS*($PUNBOPT-1)
     +&NUMTPPR*($RPRBOPT-1)
     +&NUMTPPU*($RPUBOPT-1)
```

&NUMCONS

Explanation: Variable symbol &NUMCONS specifies the type of console support to be provided by HASP. Two options are available: console support controlled totally by HASP or a HASP interface to the standard OS Console processors.

The specification &NUMCONS=n (n an integer between one and eight) causes HASP to support directly as many as n consoles. The devices which may be used as consoles are 1052, 1053, 3210, 3215, 2260 and 1443. The 1053s and 2260's must be attached locally via a 2848.

The specification &NUMCONS=0 causes HASP to interface with the OS console support, including the MCS option. All devices available with the OS console routines are supported through this interface.

Default: &NUMCONS=0

Notes:

1. If &NUMCONS=0 is specified, then all HASP functions with the exception of those listed below are provided.

   a. Non-HASP Messages, e.g., problem program WTOs, will appear on the console(s) without time tags or HASP Job numbers. A copy of the message which includes the Job number and time tag is placed in the HASP System Log.

   b. WTORs issued by the problem program will be included in the HASP Log without the OS assigned reply identification number.

   c. Replies to outstanding WTORs are not included in the HASP log.

2. If OS Multiple Console Support or M65MP or the Time Sharing Option have been SYSGENed and are to be used with HASP, then &NUMCONS should be specified as zero.

3. If &NUMCONS greater than zero is specified and if HASP initialization finds more than &NUMCONS devices of the type supported then the message

   MAXIMUM OF &NUMCONS CONSOLE(S) EXCEEDED

is issued and HASP uses as consoles the devices with the lowest unit addresses starting with the first 1052, 3210 or 3215 and continuing with the next &NUMCONS-1 devices.

4. 2260 support (&NUMCONS greater than zero) is dependent on additional specifications via the variables &SIZ2260 and &SPD2260.

5. If &NUMCONS is specified greater than zero, HASP will intercept and process all WTOs and WTORs, ignoring all MCS information. In particular, HASP will ignore all routing codes. Thus, for example, a WTO with ROUTCDE=11 will be written on HASP consoles and on the HASP System Log but not in an OS System Message Block.

6. See parameter &CONAUTH, &WTLOPT and &LOGOPT for additional information.

&NUMDA

Explanation: Variable symbol &NUMDA specifies the maximum number of direct-access volumes which may be mounted concurrently as SPOOL volumes. The specification must be an integer greater than zero.

Default: &NUMDA=2

Notes:

1. All direct-access devices except 2321s are eligible for use as SPOOL devices.
2. Specifying &NUMDA greater than the default may require increasing the value of &BUFSIZE.
3. If HASP initialization finds mounted more than &NUMDA direct-access volumes whose volume serials begin with the characters SPOOL, it will write to operator the message
   MAXIMUM OF &NUMDA SPOOL VOLUME(S) EXCEEDED
   and HASP will quiesce.
4. This variable influences the size of the HASP checkpoint record; see Note 2 of variable &MAXJOBS.
5. An associated variable is &NUMTGV.

&NUMDDT

Explanation: Variable symbol &NUMDDT specifies the number of Data Definition Tables (DDTs) to be assembled into HASP. The specification should be an integer between 3 and 256, and equal to

2 + &MAXXEQS + A + B + C + D + E

where

A = number of pseudo-2540 readers defined at SYSGEN time
B = number of pseudo-2540 punches defined at SYSGEN time
C = number of pseudo-1403 printers defined at SYSGEN time
D = number of pseudo-1442 punches defined at SYSGEN time
E = number of pseudo-1443 printers defined at SYSGEN time

Default: &NUMDDT=20

Notes:

1. Pseudo-units for &RDR and &WTR need not be counted in &NUMDDT.

&NUMINRS

Explanation: Variable symbol &NUMINRS specifies the number of 2520 pseudo-punches to be used by the generated HASP System as internal readers.

Default: &NUMINRS=0

Notes:
1. If &NUMINRS is specified as or defaulted to zero, code to support the HASP internal reader feature will be deleted from the generated system.
2. If more than &NUMINRS 2520 pseudo-punches have been specified at SYSGEN time, only the first &NUMINRS 2520 pseudo-punches can be used. It is permissible to specify &NUMINRS greater than the number of 2520 pseudo-punches specified at SYSGEN time.
3. The count of 2520 pseudo-punches is not included in HASPGEN variable &NUMDDT.

&NUMLNES

Explanation: Variable symbol &NUMLNES specifies the largest teleprocessing line identification number (mm in LINEmm) and thus the number of line definitions which are to be used by the generated HASP System. The specification must be an integer between 0 and 99 inclusive. The specification for &NUMLNES automatically becomes the specification for &NUMRJE, unless &NUMRJE is specified explicitly.

Default: &NUMLNES=0

Notes:
1. See also the HASPGEN variable LINEmm.
2. If &NUMLNES is set to or left at zero, all other Remote Job Entry parameters should be left at their default values.

&NUMOACE

Explanation: Variable symbol &NUMOACE specifies the number of overlay areas to be provided for the standard HASP Overlay feature. The specification must be an integer greater than zero.

Default: &NUMOACE=1

Notes:
1. It is judged that more than two overlay areas will benefit only a system with high performance orientation (a very fast CPU or a work load consisting of a large number of short jobs).
2. See also parameter &OLAYLEV.

&NUMPRTS

Explanation: Variable symbol &NUMPRTS specifies the maximum number of physical printers HASP may use to print the printed output of jobs. HASP supports 1403 and 3211 printers. The specification must be an integer greater than zero.

Default: &NUMPRTS=2

Notes:

1. If HASP initialization finds more than &NUMPRTS 1403 and 3211 printers, it writes to operator the message
   MAXIMUM OF &NUMPRTS PRINTER(S) EXCEEDED
   and continues normally, using as printers only the &NUMPRTS printers with lowest unit addresses.
2. Regardless of the number specified for &NUMPRTS, HASP will use only those 1403 and 3211 printers which are operational (as shown by a TIO) or on-line (for M65MP only) when HASP is started.
3. Handling of special forms by printer, the optional 1403 UCS buffer, and the 3211 UCS and Forms Control buffers is explained as part of the $T operator command.
4. This variable influences the size of the HASP checkpoint record; see Note 2 of variable &MAXJOBS.

&NUMPUNS

Explanation: Variable symbol &NUMPUNS specifies the maximum number of physical punches which will be used by HASP to punch the punched output of jobs. HASP supports 2540, 2520, and 1442 card punches. The specification must be an integer greater than or equal to zero.

Default: &NUMPUNS=1

Notes:

1. If HASP initialization finds more than &NUMPUNS 2540, 2520 and 1442 punches, it writes to operator the message
   MAXIMUM OF &NUMPUNS PUNCH(S) EXCEEDED
   and continues normally, using as printers only the &NUMPUNS punches with lowest unit addresses.
2. Regardless of the number specified for &NUMPUNS, HASP will use only those 2540, 2520 and 1442 punches which are operational (as shown by a TIO) or on-line (for M65MP only) when HASP is started.
3. If &NUMPUNS=0, parameter &ACCTNG should be set to NO.

&NUMRDRS

Explanation: Variable symbol &NUMRDRS specifies the maximum number of physical card readers HASP may use to read OS job streams. HASP supports 2540 and 2501 card readers. The specification must be an integer greater than zero.

Default: &NUMRDRS=1

Notes:

1. If HASP initialization finds more than &NUMRDRS 2540 and 2501 card readers, it writes to operator the message

   MAXIMUM OF &NUMRDRS READER(S) EXCEEDED

   and continues normally, using as readers only the &NUMRDRS readers with lowest unit addresses.
2. Regardless of the number specified for &NUMRDRS, HASP will use only those 2540 and 2501 readers which are operational (as shown by a TIO) or on-line (for M65MP only) when HASP is started.

&NUMRJE

Explanation: Variable symbol &NUMRJE specifies the largest remote terminal identification number (nn in RMTnn) and thus the number of remote terminal definitions which are to be used by the generated HASP System. The specification must be an integer between 0 and 99 inclusive.

Default: &NUMRJE=&NUMLNES

Notes:
1. See also the HASPGEN variable RMTnn.
2. If this variable is not specified and if &NUMLNES is specified as an integer greater than zero, the first &NUMLNES remote terminal definitions RMTnn are used by the generated HASP System, whether they are specified explicitly or by default.

&NUMTGV

Explanation: Variable symbol &NUMTGV specifies the number of units (track groups) into which each SPOOL volume will be divided for HASP allocation purposes. The specification must be a positive integer no greater than the number of tracks on the SPOOL device with the fewest tracks.

Default: &NUMTGV=400

Notes:

1. The user should decide upon the number of tracks he requires in a track group and then divide by that number the total number of tracks (except alternate tracks) on a typical SPOOL device type at the installation. For example, to obtain a track group size of five tracks on a 2311, the division would yield a quotient of 400; the user would specify &NUMTGV=400. If the same installation occasionally used a 2314 as a SPOOL device, the track group size for the 2314 would automatically become ten tracks.
2. Specifying a large &NUMTGV may require increasing the value of &BUFSIZE.
3. For each SPOOL volume it finds, HASP initialization calculates number of tracks per group by dividing the total number of tracks on the volume by &NUMTGV. It then marks unavailable all track groups which lie partially or wholly outside the first extent of data set SYS1.HASPACE on that volume. HASP initialization also computes the number of HASP buffers of length &BUFSIZE which will fit on a track of the SPOOL volume for each SPOOL volume mounted.

&NUMTPBF

Explanation: Variable symbol &NUMTPBF specifies the number of HASP Teleprocessing buffers for HASP Remote Job Entry to be assembled into the generated HASP system. The specification must be an integer greater than or equal to zero.

Default: &NUMTPBF=&NUMLNES

Notes:

1. Each signed-on HASP Multi-Leaving terminal requires at least two HASP Teleprocessing buffers; each other signed-on terminal requires at least one buffer. If &NUMTPBF is specified too small, HASP RJE may not work correctly.
2. See also parameters &TPBFSIZ and &MLBFSIZ.

&NUMTPES

Explanation: Variable symbol &NUMTPES specifies the maximum number of tape drives HASP may use simultaneously to read OS job streams. The specification must be an integer greater than or equal to zero.

Default: &NUMTPES=1

Notes:

1. If &NUMTPES=0 is specified, code required to support tapes as readers is omitted from the generated HASP System.
2. Since the operator specifies a unit address when issuing the start command to a tape drive (e.g., $S TPE1,182), there is rarely a need to specify for &NUMTPES a number greater than one.
3. Tapes should be equivalent to the JCL specification LABEL=(1,NL),DCB=(RECFM=FB,LRECL=80, BLKSIZE=nnnnn) where nnnnn is not greater than (10*&BUFSIZE)/11. For seven-track tape, use the additional DCB specification DEN=2,TRTCH=C.

&NUMTPPR

Explanation: Variable symbol &NUMTPPR specifies the maximum number of HASP Remote Terminal (including Multi-Leaving) printed-output streams that can simultaneously be active. The specification must be an integer greater than or equal to zero.

Default: &NUMTPPR=&NUMLNES

Notes:
1. If any remote terminal is to receive printed output, &NUMTPPR must not be zero.

&NUMTPPU

Explanation: Variable symbol &NUMTPPU specifies the maximum number of HASP Remote Terminal (including Multi-Leaving) punched-output streams that can simultaneously be active. The specification must be an integer greater than or equal to zero.

Default: &NUMTPPU=&NUMLNES

Notes:

1. If any remote terminal is to receive punched output, &NUMTPPU must not be zero.

&NUMTPRD

Explanation: Variable symbol &NUMTPRD specifies the maximum number of HASP Remote Terminal (including Multi-Leaving) input streams that can simultaneously be active. The specification must be an integer greater than or equal to zero.

Default: &NUMTPRD=&NUMLNES

Notes:

1. If any remote terminal is to read cards (including the /*SIGNON and /*SIGNOFF control cards) &NUMTPRD must not be zero.

&NUMWTOQ

Explanation: Variable symbol &NUMWTOQ specifies the number of message buffers to be provided in HASP. The specification must be an integer greater than two.

Default: &NUMWTOQ=15

Notes:

1. If &NUMCONS is specified greater than zero, additional message buffers are needed.
2. If Remote Job Entry is used, more message buffers are needed. This is especially true with console support for MULTI-LEAVING terminals.
3. Serious system degradation can be caused by specifying too few message buffers.
4. During periods of high console activity, when no message buffers are available, certain non-critical HASP messages will be discarded rather than delaying the associated process. These include certain RJE oriented messages (such as communication line error messages), execution time/line/card excession messages (continued excession will be noted when a message buffer becomes available), and certain I/O error messages on HASP-controlled devices. Additionally, when no message buffers are available in a system in which &NUMCONS is greater than zero, messages from the OS error task are queued only to a depth of one which can result in the loss of some of these messages.

&OLAYLEV

Explanation: Variable symbol &OLAYLEV specifies a HASP overlay level to be used for assembly of the various HASP control sections. Any potential overlay code defined (by the $OVERLAY macro) with a residence factor greater than &OLAYLEV will be assembled as resident code rather than overlay code. The specification for &OLAYLEV must be an integer between 0 and 15, inclusive.

Default: &OLAYLEV=15

Notes:

1. HASP uses only residence factors 4, 8, 12 and (for HASP initialization only) 0.
2. If &OLAYLEV=15, all potential overlay code will be assembled as overlay code.
3. If &OLAYLEV=0, all potential overlay code except that in HASP initialization will be assembled as resident code. HASP main storage requirements will be increased by approximately 24K over the case &OLAYLEV=15.
4. Regardless of the setting of &OLAYLEV, the installation systems programmer may use control cards for the HASP Overlay Builder after the HASPGEN process is complete to specify that a particular section of potential overlay code be made either resident code or overlay code.

&OREPSIZ

Explanation: Variable symbol &OREPSIZ specifies the size in bytes of a table in HASP to be used to hold REP data for true overlay code. The REPs associated with a particular section of true overlay code will be applied to that code every time it is brought into main storage from the HASP overlay library. The specification for &OREPSIZ must be either 0 or an integer not less than 10.

Default: &OREPSIZ=0

Notes:

1. Each entry in the HASP Overlay REP table consists of 8+n bytes ($2 \leq n \leq 256$) where n is the number of contiguous bytes to be changed in a section of overlay code.
2. The table is used only if the operator specifies to HASP initialization that REPs are to be used and if some of the REPs are for sections of true overlay code.
3. If the HASP Overlay REP table is too small to handle all true overlay REPs, HASP initialization writes to operator the message
   OVERLAY REPPING ERROR
   and HASP quiesces.
4. See also Section 6.4 of this manual.

Explanation: Subscripted variable symbols &OSC(n) specify OS job classes. A job selected by HASP logical partition n will be submitted to OS with the job class &OSC(n). Each specification must be a single unique letter between A and O, inclusive. No two specifications may be the same.

Default:
```
&OSC(1)=A
&OSC(2)=B
&OSC(3)=C
&OSC(4)=D
&OSC(5)=E
&OSC(6)=F
&OSC(7)=G
&OSC(8)=H
&OSC(9)=I
&OSC(10)=J
&OSC(11)=K
&OSC(12)=L
&OSC(13)=M
&OSC(14)=N
&OSC(15)=O
```

Notes:

1. Only the first &MAXPART specifications, &OSC(1) through &OSC(&MAXPART) will be used.
2. In an MVT system, HASP initialization issues the &MAXPART commands
```
S INIT.HOSINIT&OSC(1),,,&OSC(1)


S INIT.HOSINIT&OSC(&MAXPART),,,&OSC(&MAXPART).
```
3. In an MFT system, HASP initialization issues the single command
```
S INIT.ALL;
```
thus the classes of the MFT partitions to be controlled by HASP must have already been defined. Each such partition must be defined with only one job class; that job class must match one and only one of the &MAXPART job classes &OSC(1), &OSC(&MAXPART).

Explanation: Variable symbol &OSINOPT specifies inclusion or exclusion of the HASP OS Input Spooling option. The specification must be either YES or NO. If &OSINOPT=YES and a DD * (or DD DATA) statement specifies the DCB keyword, HASP will pass the DD statement and the data following it to the OS Reader/Interpreter; OS will perform input spooling. If &OSINOPT=NO, and for every DD * (or DD DATA) statement that does not specify the DCB= keyword, HASP will SPOOL the input data as usual.

Default: &OSINOPT=NO

&OUTPOPT

Explanation: Variable symbol &OUTPOPT specifies the action to be taken when a job exceeds its estimated print lines or punched cards of output. The specification must be one of the integers 0, 1 or 2. For &OUTPOPT=2, output excession causes the job to be cancelled with a dump. For &OUTPOPT=1, output excession causes the job to be cancelled without a dump. For &OUTPOPT=0, output excession does not cause the job to be cancelled.

Default: &OUTPOPT=0

Notes:

1. Regardless of the specification for &OUTPOPT, output excession causes messages to be written to the operator. See also Notes 1 and 2 of &OUTXS.
2. If &OUTPOPT=2 is specified, users should use SYSUDUMP or SYSABEND DD cards if a core dump is desired on output excession.

$OUTXS

Explanation: Ordinary symbol $OUTXS specifies the interval, in print lines/punched cards, at which messages will be written to the operator informing him that a job's print line count or punch card count has been exceeded. The specification must be an integer greater than zero.

Default: $OUTXS=2000

Notes:
1. The first print line excession message will be written to the operator when the job's estimated print line count has been exceeded.
2. The first punched card excession message will be written to the operator when the job's estimated punched card count has been exceeded.

&PID(n)

Explanation: Subscripted variable symbols &PID(n) specify the identifiers to be used with the HASP logical partitions. Each specification must be a unique 1- or 2-character string.

Default:
```
&PID(1)=1
&PID(2)=2
&PID(3)=3
&PID(4)=4
&PID(5)=5
&PID(6)=6
&PID(7)=7
&PID(8)=8
&PID(9)=9
&PID(10)=10
&PID(11)=11
&PID(12)=12
&PID(13)=13
&PID(14)=14
&PID(15)=15
```

Notes:
1. Only the first &MAXPART specifications, &PID(1) through &PID(&MAXPART), will be used.
2. The identifiers &PID(n) are used in messages to and commands from the operator. For example, when an operator uses the set command $T Imm,list he is referring not to logical partition mm but to logical partition n, where &PID(n)=mm.

&PRI(n)

Explanation: Subscripted variable symbols &PRI(n) specify OS job priorities. A job selected by HASP logical partition n will be submitted to OS with the job priority &PRI(n). Each specification must be an integer between 1 and 15, inclusive.

Default:
```
&PRI(1)=7
&PRI(2)=7
&PRI(3)=7
&PRI(4)=7
&PRI(5)=7
&PRI(6)=7
&PRI(7)=7
&PRI(8)=7
&PRI(9)=7
&PRI(10)=7
&PRI(11)=7
&PRI(12)=7
&PRI(13)=7
&PRI(14)=7
&PRI(15)=7
```

Notes:

1. The defaults are all the same as &XZPRTY. This allows the HASP Execution Task Monitor to regulate all job steps under the control of HASP. See also parameters &XZPRTY and &MONINTV.
2. These parameters have no effect in MFT.
3. The priorities defined by &PRI(n) affect only OS execution. The priority of a job in the HASP Job Queue is determined by parameters &RPRT(m), &RPRI(m), &XLIN(m), and &XPRI(m).
4. Only the first &MAXPART specifications, &PRI(1) through &PRI(&MAXPART), will be used.

$PRICONA

Explanation: Ordinary symbol $PRICONA specifies the unit address of a 1052, 3210, 3215, 1443, or 1403 to which HASP will issue a SIO in the event of a catastrophic error. The message HASP writes to this device is:

HASP CATASTROPHIC ERROR. CODE=xxx.

The specification must be a valid unit address.

Default: $PRICONA=01F

Notes:

1. When HASP is operating with M65MP in a 2-CPU multiprocessor environment, the device specified by $PRICONA must be operational to both CPUs when a HASP catastrophic error occurs.

$PRIDCT

Explanation: Ordinary symbol $PRIDCT specifies the number of print lines to appear on each HASP job separator page for local printers. The specification must be an integer greater than or equal to zero. If the specification is zero, no separator page will be produced on local printers.

Default: $PRIDCT=61

Notes:
1. The equivalent HASPGEN parameter for remote terminal printers is $TPIDCT.

&PRIHIGH

Explanation: Variable symbol &PRIHIGH specifies a HASP priority to be associated with the HASP Priority Aging feature. A job will not be priority-aged if its HASP priority is (or becomes) greater than or equal to &PRIHIGH. The specification must be an integer between 0 and 15, inclusive.

Default: &PRIHIGH=10

Notes:
1. If &PRIRATE=0, parameter &PRIHIGH is not used.
2. See also parameters &PRIRATE and &PRILOW.

&PRILOW

Explanation: Variable symbol &PRILOW specifies a HASP priority to be associated with the HASP Priority Aging feature. A job will not be priority-aged by HASP unless its HASP priority is initially at least &PRILOW. The specification must be an integer between 0 and 15, inclusive.

Default: &PRILOW=5

Notes:
1. If &PRIRATE=0, parameter &PRILOW is not used.
2. See also parameters &PRIRATE and &PRIHIGH.

&PRIRATE

Explanation: Variable symbol &PRIRATE specifies the amount by which a job's HASP priority will be incremented in 24 hours by the HASP Priority Aging feature. For example if &PRIRATE=3 then a job's priority will be incremented by one for every eight hours it remains in the system. But a job's priority will not be incremented unless it is at least &PRILOW; nor will a job's priority be incremented above &PRIHIGH. The specification must be an integer greater than or equal to zero. If zero is specified, Priority Aging is excluded from the generated HASP system.

Default: &PRIRATE=0

Notes:
1. If &PRIRATE=0, parameters &PRILOW and &PRIHIGH are not used.
2. See also parameters &RPRT(n), &RPRI(n), &XLIN(n), and &XPRI(n).
3. If a job's priority is specified on the /*PRIORITY control card, the job will be priority-aged if its priority is eligible.

$PRTBOPT

Explanation: Ordinary symbol $PRTOPT specifies the printer buffering option to be used for local HASP printers. The specification must be either 1 (for single buffering) or 2 (for double buffering).

Default: $PRTBOPT=2

&PRTRANS

Explanation: Variable symbol &PRTRANS specifies translation for lines of print. The specification must be either YES or NO.

Default: &PRTRANS=YES

Notes:
1. If &PRTRANS is specified as YES, each line to be printed by HASP is first translated. Translation changes lower-case letters to upper-case letters and characters invalid on a PN train to blanks.
2. If any print train is to be used on any HASP-controlled printer which has characters not equivalent to those on a PN train or a P11 train, &PRTRANS must be specified as NO.

&PRTUCS

Explanation: Variable symbol &PRTUCS specifies the name of the print chain or print train which HASP initially assumes is mounted on every local 1403 printer SYSGENed with the UCS feature, and on every local 3211 printer. The UCS identifier can be modified by the operator individually by printer. The specification should be either AN, HN, PN, QN, RN, UN, A11, H11, P11, U11, or 0.

Default: &PRTUCS=0

Notes:
1. A specification of zero causes HASP to bypass the UCS loading procedure on all local printers until the UCS type of each printer is specified by the operator.
2. Only the first character of &PRTUCS is interrogated. That is to say, an AN specification is equivalent to an A11 specification, an H11 specification is equivalent to an HN specification, etc.
3. If a UCS specification is encountered which is not valid for the type of printer being addressed, the UCS loading procedure will be bypassed.
4. The UN and U11 specifications are provided for installation use to support other type of print chains.

$PUNBOPT

Explanation: Ordinary symbol $PUNBOPT specifies the punch buffering option to be used for local HASP punches. The specification must be either 1 (for single buffering) or 2 (for double buffering).

Default: $PUNBOPT=1

&RDR

Explanation: Variable symbol &RDR specifies the unit address of a pseudo-2540 reader to be used with HOSRDR to supply jobs to OS. The specification must be a valid unit address which has been specified at SYSGEN time as a pseudo-2540 reader.

Default: &RDR=0FC

&RDRPART

Explanation: Variable symbol &RDRPART specifies for MFT systems the identifier field of an OS START command issued by HASP initialization for the OS reader/interpreter HOSRDR. The complete START command is

```
S  HOSRDR.&RDRPART,&RDR.
```

The specification must be a valid identifier field for an OS START reader command, as described in the OS Operator's Guide.

Default: &RDRPART=S

Notes:

1. If HASP initialization detects an MVT system, this parameter is not used.

$REPRDR

Explanation: Ordinary symbol $REPRDR specifies the unit address of a physical 2540 or 2501 card reader from which HASP initialization will read REP cards if requested by the operator. The specification must be a valid unit address.

Default: $REPRDR=00C

Notes:
1. When REP cards are to be read and HASP is operating with M65MP in a 2-CPU multiprocessor environment, the device specified by $REPRDR must be operational to both CPUs.

$REPWTR

Explanation: Ordinary symbol $REPWTR specifies the unit address of a physical 1403 or 1443 on which each REP card read is to be printed, if printing of REP cards is requested by the operator. The specification must be a valid unit address.

Default: $REPWTR=00E

Notes:
1. When REP cards are to be printed and HASP is operating with M65MP in a 2-CPU multiprocessor environment, the device specified by $REPWTR must be operational to both CPUs.

&RESCORE

Explanation: Variable symbol &RESCORE specifies a storage size, in multiples of 1024 bytes. HASP will always issue a GETMAIN for additional storage for HASP buffers; all such storage but &RESCORE*1024 bytes will be used for buffers.

The specification must be an integer greater than or equal to zero.

Default: &RESCORE=0

&RJOBOPT

Explanation: Variable symbol &RJOBOPT specifies whether or not an illegal HASP JOB card is to prevent execution of the associated job. The specification must be either YES or NO.

Default: &RJOBOPT=NO

Notes:

1. If &RJOBOPT=YES and HASP reads a JOB card whose accounting field does not match the specifications required by HASP, the job is flushed by HASP and an appropriate message is written to the operator.

RMTnn

Explanation: Ordinary symbols RMTnn specify the characteristics of remote terminals to be used with HASP Remote Job Entry. Terminals must be defined consecutively, starting with RMT01. Each specification must be a 14-character string of the form

RMTnn=mmooppiillwtdf

where the letters represent the following:

Code Letters	Range	Description
nn	01-99	Remote Number
mm	01-99	Line Number (**indicates /*SIGNON assignment)
oo	00-99	Print Routing (Remote Number)
pp	00-99	Punch Routing (Remote Number)
ii	00-15	Priority Increment for this Remote
ll	00-15	Priority Limit for this Remote
w	0-6	Printer Width as follows: 0 = 80 characters 1 = 100 characters 2 = 120 characters 3 = 132 characters 4 = 144 characters 5 = 150 characters 6 = 96 characters
t	0-6	Terminal Type as follows: 0 = 1009, 2770 1 = 1978, 2780, 7702 2 = System 360/20 Sub-model 2 3 = System 360/20 Sub-model 5 4 = System 360/25, 30, 40, 50, etc. 5 = 1130 6 = System/3
d	0-4	Data Format as follows: 0 = Unblocked fixed length 1 = Blocked fixed length 2 = Unblocked variable length 3 = Blocked variable length (Note - this should be used for all 1978, 2770, and 2780 terminals.)

4 = Programmable Interface (Note - this should be used for all STR 20 and BSC MULTI-LEAVING interfaces.)

f Terminal Features as follows:

0-3 2770 Terminal Features

f	Buffer Expansion	Transparency	Notes
0	No	No	w must be 0, 1, 2 or 6
1	No	Yes	w must be 0, 1 2 or 6
2	Yes	No	
3	Yes	Yes	

0-7 2780 Terminal Features

f	Horizontal Format Control	Multiple Record Feature	Transparency
0	No	No	No
1	No	No	Yes
2	No	Yes	No
3	No	Yes	Yes
4	Yes	No	No
5	Yes	No	Yes
6	Yes	Yes	No
7	Yes	Yes	Yes

0-3 MULTI-LEAVING Terminal Features

f	Console Support	Transparency
0	No	No
1	No	Yes
2	Yes	No
3	Yes	Yes

Default: RMTnn=**nn0000153131

Notes:
1. Parameter &NUMRJE must specify the number of specifications RMTnn to be included in the generated HASP system.

2. No two specifications RMTnn may specify the same line number (mm). If ** is specified instead of a line number for mm, the associated remote terminal may connect to HASP via any suitable line. HASP will logically connect the terminal with the line when it recognizes the /*SIGNON control card. If line number is specified explicitly, the associated terminal need not use a /*SIGNON card.
3. The line number specification mm refers to line specification LINEmm, which in turn specifies the unit address of the line.
4. For print and punch routing, a specification of 00 causes output from jobs submitted at the Remote Terminal to be printed/punched locally, unless re-routed.
5. Priority increment is the value to be added to the priority of a job submitted from the Remote Terminal.
6. Priority limit is the maximum value of priority for any job submitted from the Remote Terminal.
7. If any MULTI-LEAVING workstation is to utilize more than one reader, printer or punch, see Section 12.16 for additional information.

$RPRBOPT

Explanation: Ordinary symbol $RPRBOPT specifies the printer buffering option to be used for all printers at HASP Remote Terminals. The specification must be either 1 (for single buffering) or 2 (for double buffering).

Default: $RPRBOPT=1

Notes:
1. The specification refers to HASP regular buffers, not to HASP Teleprocessing buffers.

&RPRI(n)

Explanation: Subscripted variable symbols &RPRI(n) specify tentative job priorities corresponding to intervals defined by subscripted variable symbols &RPRT(n). If a user specifies in the accounting field of his job card an estimated execution time of t minutes, the job's tentative priority will be &RPRI(n) where

&RPRT(n-1)<t≤&RPRT(n).

Each specification must be an integer between 0 and 15 inclusive.

Default:
&RPRI(1)=9
&RPRI(2)=8
&RPRI(3)=7
&RPRI(4)=6
&RPRI(5)=5
&RPRI(6)=4
&RPRI(7)=3
&RPRI(8)=2
&RPRI(9)=1

Notes:

1. The values &RPRI(n) will normally be in opposite order from the subscripts n.
2. See also Notes 1 and 2 for &RPRT(n).

&RPRT(n)

Explanation: Subscripted variable symbols &RPRT(n) specify estimated execution times in minutes. If a user specifies in the accounting field of his job card an estimated execution time of t minutes, and if t satisfies the relation

&RPRT(n-1)<t≤&RPRT(n)

then &RPRI(n) will be the tentative priority of the job. If t is less than &RPRT(1) or greater than &RPRT(9), value &RPRI(1) or zero will be the tentative priority of the job. Each specification must be an integer between 1 and X'FFFFFF'/60 inclusive.

Defaults:
```
&RPRT(1)=2
&RPRT(2)=5
&RPRT(3)=15
&RPRT(4)=X'FFFFFF'/60
&RPRT(5)=X'FFFFFF'/60
&RPRT(6)=X'FFFFFF'/60
&RPRT(7)=X'FFFFFF'/60
&RPRT(8)=X'FFFFFF'/60
&RPRT(9)=X'FFFFFF'/60
```

Notes:

1. Priority &RPRI(n) is overridden by HASP control card /*PRIORITY.
2. The tentative priority defined above is adjusted according to &XLIN(m) and the estimated print lines specified in the accounting field of the user's JOB card. If the user estimated p print lines, and if p satisfies the relation

   &XLIN(m)<p≤&XLIN(m+1)

   then the tentative priority is reduced by m.
3. The values &RPRT(n) should be in the same order as the subscripts n.

$RPUBOPT

Explanation: Ordinary symbol $RPUBOPT specifies the printer buffering option to be used for all punches at HASP Remote Terminals. The specification must be either 1 (for single buffering) or 2 (for double buffering).

Default: $RPUBOPT=1

Notes:
1. The specification refers to HASP regular buffers, not to HASP Teleprocessing buffers.

&RQENUM

Explanation: Variable symbol &RQENUM specifies the number of WTOR reply buffers to be provided in the generated HASP system. The specification must be an integer greater than zero.

Default: &RQENUM=5

Notes:
1. This parameter is ignored if &NUMCONS=0.
2. If &NUMCONS is not specified as zero, no more than &RQENUM replies can be outstanding at any time.

&SIZ2260

Explanation: Variable symbol &SIZ2260 specifies screen width in characters for local 2260s (attached via 2848) to be used as HASP operator consoles. The specification must be either 0, 40, or 80. If 0 is specified, support for local 2260s and local 1053s (attached via 2848) will be excluded from the generated HASP system.

Default: &SIZ2260=0

Notes:
1. This parameter is not used if &NUMCONS=0.
2. See also parameter &SPD2260.

&SPD2260

Explanation: Variable symbol &SPD2260 specifies roll rate in hundredths of a second for local 2260s (attached via 2848) to be used as HASP operator consoles. If new messages are pending for display on a HASP 2260 console, they will be displayed (and old messages will be deleted, if the screen is full) at the rate of one message every &SPD2260/100 seconds.

Default: &SPD2260=50

Notes:
1. This parameter is not used if &NUMCONS=0 or if &SIZ2260=0.

&SPOLMSG

Explanation: Variable symbol &SPOLMSG specifies the number of physical records in the first extent of SYS1.HASPACE on SPOOL1 which are to be reserved for holding operator and HASP messages for HASP Remote Terminals. Each physical record is capable of holding one or more messages for a single remote terminal. Messages are held if they are directed to:

- any terminal not signed on;
- any signed-on hardware terminal which is currently processing an input or output stream;
- any signed-on computer terminal that is not a Multi-Leaving terminal with a console.

If a message is to be held but no space is available to hold it, the message is thrown away without operator notification.

The specification for &SPOLMSG must be an integer greater than or equal to zero. If &SPOLMSG is specified as zero, no messages will be sent to hardware terminals.

Default: &SPOLMSG=10*&NUMRJE

Notes:

1. Only the $DM command can generate messages to a terminal not signed on.
2. For signed-on terminals, messages are generated for job-on-reader, by $DM, and as responses to commands from the terminal.
3. Each message to a terminal (except to a Multi-Leaving remote defined with a console) is held until it can be printed, or until HASP is restarted.

&STRCPU

Explanation: Variable symbol &STRCPU specifies inclusion or exclusion in the HASP Remote Terminal Access Method of Remote Job Entry support for the System/360 Model 20 with a Synchronous Transmit-Receive (STR) adapter and the associated HASP Remote Terminal program. The specification must be either YES or NO.

Default: &STRCPU=NO

&STR1978

Explanation: Variable symbol &STR1978 specifies inclusion or exclusion in the HASP Remote Terminal Access Method of Remote Job Entry support for Synchronous Transmit-Receive (STR) hardware terminals such as the 1978. The specification must be either YES or NO.

Default: &STR1978=NO

&TIMEOPT

Explanation: Variable symbol &TIMEOPT specifies the action to be taken when a job's estimated execution time is exceeded. The specification must be one of the integers 0, 1, 2 or 4. For &TIMEOPT=4, the job's time limits will not be monitored. For &TIMEOPT=2, time excession causes the job to be cancelled with a dump. For &TIMEOPT=1, time excession causes the job to be cancelled without a dump. For &TIMEOPT=2, &TIMEOPT=1, or &TIMEOPT=0, time excession causes messages to be written to the operator.

Default: &TIMEOPT=4

Notes:

1. See also Notes 1 and 2 of &ESTIME, which apply for &TIMEOPT=0, &TIMEOPT=1, and &TIMEOPT=2.

$TIMEXS

Explanation: Ordinary symbol $TIMEXS specifies the interval, in minutes, at which messages will be written to the operator informing him that a job's execution time is exceeded. The specification must be an integer greater than zero.

Default: $TIMEXS=1

Notes:
1. The first time excession message is written to the operator when the job's estimated execution time has been exceeded.
2. If &TIMEOPT is specified greater than 2, $TIMEXS is not used.
3. See also Note 2 of $ESTIME.

&TPBFSIZ

Explanation: Variable symbol &TPBFSIZ specifies the size in bytes of each HASP Teleprocessing buffer. The specification must be an integer not less than 328 if &BSC2770=NO and &BSC2780=NO, and otherwise not less than 400.

Default: &TPBFSIZ=400

Notes:

1. The value of &TPBFSIZ is the maximum size of any HASP Teleprocessing buffer. See also parameter &MLBFSIZ, which may never be specified larger than &TPBFSIZ.
2. The HASP Remote Terminal program for the System/360 Model 20 with an STR communications adapter (HRTPSM20) uses a teleprocessing buffer size of &TPBFSIZ; all other HASP Remote Terminal programs are Multi-Leaving programs, and use &MLBFSIZ.
3. The parameter &TPBFSIZ is specified only once, at HASPGEN time; it is conveyed automatically to the requisite Remote Terminal programs by HASPGEN.
4. See also the notes for &MLBFSIZ.

$TPIDCT

Explanation: Ordinary symbol $TPIDCT specifies the number of print lines to appear on each HASP job separator page for jobs whose printed output is directed to any HASP Remote Terminal. The specification must be an integer greater than or equal to zero. If the specification is zero, no separator page will be produced on remote printers.

Default: $TPIDCT=6

Notes:
1. The equivalent HASPGEN parameter for local printers is $PRIDCT.

&TRACE

Explanation: Variable symbol &TRACE specifies inclusion or exclusion of a facility for event-tracing and statistics-gathering in the generated HASP system. It also specifies the number of entries to be generated in the HASP trace table. The specification must be an integer greater than or equal to zero.

Default: &TRACE=0

Notes:

1. Inclusion of the HASP Trace facility causes the OS program interrupt exit (SPIE) mechanism to work incorrectly. For this reason, the HASP Trace should not be included in any generated HASP system designed for normal production.
2. The &TRACE option is independent of the &DEBUG option.

&USASCII

Explanation: Variable symbol &USASCII specifies inclusion or exclusion in the HASP Remote Terminal Access Method of the capability to use USASCII line control characters as well as EBCDIC line control characters. If any line specification LINEmm for a BSC line has value c set to 2, 3, 6 or 7, &USASCII should be set to YES; otherwise, &USASCII should be set to NO.

Default: &USASCII=NO

$WAITIME

Explanation: Ordinary symbol $WAITIME specifies a time interval, in seconds. For hardware terminals, the HASP Remote Terminal Access Method will wait $WAITIME seconds at the completion of processing of any input stream, printed output stream, or punched output stream, to allow the operator time to alter the normal sequence of Remote Job Entry operations. For example, the operator may wish to transmit another job to HASP after a previous job has finished printing rather than wait till the previous job has finished punching.

The specification for $WAITIME must be an integer greater than zero.

Default: $WAITIME=1

&WCLSREQ

Explanation: Variable symbol &WCLSREQ specifies optional requeueing for OS output classes specified by &WTRCLAS. The values assigned &WCLSREQ are effective only if &WTRPART=*.

If &WTRPART=*, then the HASP writer subtask (load module HASPWTR) processes jobs queued in the OS output queues defined by &WTRCLAS. At the end of processing a job whose output class is the nth character of &WTRCLAS, HASPWTR examines the nth character of &WCLSREQ. If the nth character of &WCLSREQ is *, HASPWTR deletes the job from the OS Job Queue. But if the nth character of &WCLSREQ is an OS Output class, HASPWTR requeues the job in the OS Output queue specified by the nth character of &WCLSREQ (which must be different from any class specified in &WTRCLAS).

The specification must be a string of one to eight characters each of which is either * or a unique valid OS output class different from any specified in &WTRCLAS. If more characters are specified than were specified for &WTRCLAS, the excess characters are unused.

Default: &WCLSREQ=********

Notes:

1. The output requeueing option is useful for providing an extra copy of a job's system messages to, for example, a conversational programming terminal.
2. A requeued job is not referenced by HASP, but must be accessed by a standard OS Output Writer or other suitable means.
3. A requeued job may contain a mixture of system messages and sysout data sets of the same class, if the sysout data sets were spooled by OS (see HASPGEN parameter $$x). The module HASPWTR does not process the sysout data sets, but requeues the entire job containing them in the new class specified by &WCLSREQ. The system messages and sysout data sets are then available to a standard OS Output Writer which is processing the new class.

4. Any DD statements in the system messages of a requeued job, which are originally coded as DD* or DD DATA and are not subject to OS spooling (see HASPGEN parameter &OSINOPT), are available to a Writer processing a &WCLSREQ class as DD$ and DD CATA respectively. They are printed as DD$ and DD CATA unless the Writer is programmed to change them to their original form.

Explanation: Variable symbol &WTLOPT specifies inclusion or exclusion of code to cause HASP to intercept the WTL SVC (SVC 36) and to add to a job's output any log messages associated with it. The messages will be written on the HASP System Log for the job. The specification for &WTLOPT must be either YES or NO.

Default: &WTLOPT=NO

Notes:
1. If &WTLOPT is set to YES and &NUMCONS is set non-zero, no messages will be recorded on the OS log data sets and the OS WRITELOG command must not be used.
2. If &WTLOPT is set to YES and &LOGOPT is set to NO, all WTL messages will be thrown away.

&WTR

Explanation: Variable symbol &WTR specifies the unit address of a pseudo-1403 printer to be used by a writer to retrieve from the OS Job Queue System Message Blocks (SMBs) for jobs controlled by HASP. The specification must be a valid unit address which has been specified at SYSGEN time as a pseudo-1403 printer.

Default: &WTR=0FE

Notes:

1. The unit address assigned to this parameter must not be assigned a symbolic unit name at SYSGEN time, as described for other pseudo-1403 printers.

&WTRCLAS

Explanation: Variable symbol &WTRCLAS specifies the OS System Output classes to be processed by HASP. The output writer started by HASP initialization (and selected by the &WTRPART Parameter) is intended to process only those System Message Blocks (SMBs) created by OS jobs submitted to and controlled by HASP. If other OS writers are to be used concurrently with the writer started by HASP, none of them may process any of the output classes specified in &WTRCLAS.

The specification for &WTRCLAS must be one to eight unique characters that are valid OS output classes.

Default: &WTRCLAS=HA

Notes:

1. HASP examines the MSGCLASS parameter of every JOB card it sends to OS. If MSGCLASS is not specified or is not one of the classes specified by &WTRCLAS, HASP adds the MSGCLASS parameter to the JOB card, using as a class the leftmost character of &WTRCLAS.
2. If a job submitted to OS by HASP has certain errors on the JOB card, OS will fail the job and change its MSGCLASS to A. It is therefore recommended that class A be specified in &WTRCLAS. If class A is not specified and such an error happens, HASP may not operate correctly.
3. See also HASPGEN parameter $$x.

&WTRPART

Explanation: Variable symbol &WTRPART specifies the method HASP will use to retrieve from the OS Job Queue System Message Blocks for jobs controlled by HASP.

For &WTRPART=*, HASP initialization creates a subtask to interface directly between HASP and the OS Job Queue.

If &WTRPART is not specified as *, HASP initialization starts an OS writer (using procedure HOSWTR) to interface between HASP and the OS Job Queue. In particular, for MFT systems HASP initialization issues the OS command

```
S  HOSWTR.&WTRPART,&WTR,,&WTRCLAS
```

The specification for &WTRPART must be either * or, for MVT, any other character string of one to eight characters or, for MFT, a valid identifier for an OS START writer command, as described in the OS Operator's Guide.

Default: &WTRPART=*

Notes:

1. For an OS MFT system, the default specification requires that, during SYSGEN, the SUPRVSOR macro include ATTACH in the OPTIONS=keyword.
2. If &WTRPART is not specified as *, it is recommended for an MFT system that HOSWTR be assigned the partition immediately below HASP in priority. That is, if HASP were to be assigned PO, &WTRPART would be specified as P1.

&XBATCHC

Explanation: Variable symbol &XBATCHC specifies a list of job classes to be used with the HASP Execution Batch Scheduling feature. The specified classes are excluded from running jobs outside of Execution Batch Scheduling. The specification for &XBATCHC is a string of one to eight characters (letters and numbers) which specify valid unique HASP job classes. If &XBATCHC is left at its default, the generated HASP system will not include Execution Batch Scheduling.

Default: &XBATCHC=[null string]

Notes:
1. For further information, see the section of this manual on the Execution Batch Scheduling feature.
2. If &XBATCHC is not specified, then &XBATCHN is not used.

&XBATCHN

Explanation: Variable symbol &XBATCHN specifies the first five characters of the name of each OS job to be started internally by HASP when required for the execution of a user "job" under the HASP Execution Batch Scheduling feature. The specification must be a five-character string of which the first character is alphabetic or national and the remaining four are alphameric or national.

Default: &XBATCHN=$$$$$

Notes:
1. For further information, see the section of this manual on the Execution Batch Scheduling feature.
2. If &XBATCHC is specified, then HASP will reject all user submitted jobs whose jobnames start with the five characters &XBATCHN.

&XLIN(n)

Explanation: Subscripted variable symbols &XLIN(n) specify estimated line counts. If a user specifies in the accounting field of his job card an estimated line count of l=p/1000, then the tentative job priority computed on the basis of his estimated execution time will be reduced by n, where

&XLIN(n)<p≤&XLIN(n+1).

Each specification must be an integer between 1 and 16,777,215. &XLIN(9) must be 16,777,215.

Default:
```
&XLIN(1)=2000
&XLIN(2)=5000
&XLIN(3)=15000
&XLIN(4)=X'FFFFFF'
&XLIN(5)=X'FFFFFF'
&XLIN(6)=X'FFFFFF'
&XLIN(7)=X'FFFFFF'
&XLIN(8)=X'FFFFFF'
&XLIN(9)=X'FFFFFF'
```

Notes:

1. The values &XLIN(n) must be in the same order as the subscript n.
2. See also Note 2 for &RPRT(n).
3. These values are not used if the job uses a /*PRIORITY HASP control card.
4. See also the description of &XPRI(n), used with &XLIN(n) to determine a job's printing priority.

&XPRI(n)

Explanation: Subscripted variable symbols &XPRI(n) specify job priorities for printing which correspond to intervals defined by subscripted variable symbols &XLIN(n). If a user does not supply a /*PRIORITY control card with his job, the job's priority is recomputed after execution based upon the actual number of print lines it produced. If the job produced p print lines then its priority for printing and punching will become &XPRI(n), where n is the smallest number for which

$p \leq$ &XLIN(n).

Each specification must be an integer between 0 and 15.

Default:
```
&XPRI(1)=9
&XPRI(2)=8
&XPRI(3)=7
&XPRI(4)=6
&XPRI(5)=5
&XPRI(6)=4
&XPRI(7)=3
&XPRI(8)=2
&XPRI(9)=1
```

&XZMFTH

Explanation: Variable symbol &XZMFTH specifies the dispatching priority of the highest-priority MFT task to be included in the group of tasks analyzed by the HASP Execution Task Monitor. Each MFT HASP-controlled job step task without subtasks whose dispatching priority falls within the range &XZMFTL through &XZMFTH is examined by the HASP Execution Task Monitor every &MONINTV seconds. In order to balance the CPU utilization characteristics of these tasks, the Execution Task Monitor resets the dispatching priority of each of them to &XZMFTL and, if necessary, changes their order on the TCB ready chain. The specification for &XZMFTH must be one or two hexadecimal characters ranging from 00 to FF.

Default: &XZMFTH=FF

Notes:
1. If &XZMFTH is specified as 00, Execution Task Monitor support for MFT is excluded from the generated HASP system.
2. If &XZMFTH is specified as 00 and &XZPRTY is specified as 0-1, then &MONINTV must be specified as 0.

&XZMFTL

Explanation: Variable symbol &XZMFTL specifies the dispatching priority of the lowest-priority MFT task to be included in the group of tasks analyzed by the HASP Execution Task Monitor. Each MFT HASP-controlled job step task without subtasks whose dispatching priority falls within the range &XZMFTL through &XZMFTH is examined by the HASP Execution Task Monitor every &MONINTV seconds. In order to balance the CPU utilization characteristics of these tasks, the Execution Task Monitor resets the dispatching priority of each of them to &XZMFTL and, if necessary, changes their order on the TCB ready chain. The specification for &XZMFTL must be one or two hexadecimal characters ranging from 00 to FF.

Default: &XZMFTL=00

Notes:

1. If &XZMFTH is specified as 00, &XZMFTL is not used and Execution Task Monitor support for MFT is excluded from the generated HASP system.

&XZPRTY

Explanation: Variable symbol &XZPRTY specifies a priority value used by the HASP Execution Task Monitor for MVT systems. The Execution Task Monitor periodically analyzes each MVT HASP-controlled job step task, without subtasks, whose dispatching priority is &XZPRTY*16+11. In order to balance the CPU utilization characteristics of these tasks, the Execution Task Monitor may re-order these tasks on the TCB ready chain. The specification for &XZPRTY must be an integer between 0 and 15 inclusive, or the expression "0-1". The latter value should be used when the Execution Task Monitor is to operate for MFT but not for MVT.

Default: &XZPRTY=7

Notes:
1. If &MONINTV=0, the value of &XZPRTY is not used.
2. If &XZPRTY is specified as 0-1 and &XZMFTH is specified as 0, then &MONINTV must be set to 0.

$$x

Explanation: Ordinary symbol $$x specifies the destination for an output data set designated in the user's JCL as SYSOUT=x. The specification for each of these ordinary symbols must be one of the characters A, B, 1, 2, or *.

For $$x=A, associated SYSOUT data sets will be printed with the user's job.

For $$x=B, associated SYSOUT data sets will be punched with the user's job.

For $$x=1, associated SYSOUT data sets will be added to the HASP special forms queue, to be printed with other SYSOUT data sets requiring the same forms.

For $$x=2, associated SYSOUT data sets will be added to the HASP special forms queue, to be punched with other SYSOUT data sets requiring the same forms.

For $$x=*, associated SYSOUT data sets will be processed entirely by OS. In this case, HASP will add the specification UNIT=SYSDA to the JCL, unless the user has himself specified UNIT=information.

Default:

$$A=A	$$S=A
$$B=B	$$T=A
$$C=A	$$U=A
$$D=A	$$V=A
$$E=A	$$W=A
$$F=A	$$X=A
$$G=A	$$Y=A
$$H=A	$$Z=A
$$I=A	$$1=A
$$J=1	$$2=A
$$K=2	$$3=A
$$L=A	$$4=A
$$M=A	$$5=A
$$N=A	$$6=A
$$O=A	$$7=A
$$P=A	$$8=A
$$Q=A	$$9=A
$$R=A	$$0=A

Notes:

1. For any output class x, regardless of the value specified for $$x, a four-digit special forms number can be coded as the third positional parameter of the SYSOUT=keyword. The specification is converted to a packed number (unless $$x=*); that is, forms number 0001 is the same as forms number 01.
2. For an output class x for which $$x=*, the SYSOUT=parameter may be coded as described in the OS Job Control Language Reference manual.
3. A user SYSOUT specification which includes the second positional parameter (program name) will be processed entirely by OS, regardless of whether the associated $$ parameter was specified as *.
4. If a given output class x is one of the classes assigned to &WTRCLAS, it must not be used in a SYSOUT specification to be processed by OS (caused if $$x=*, or if the second parameter of SYSOUT is used), unless that class x is subject to requeueing as described under the parameter &WCLSREQ.

7.2 RMTGEN PARAMETERS FOR SYSTEM/360 MODEL 20 STR

This section describes the parameters used in assembly of the System/360 Model 20 STR Remote Terminal Program for HASP Remote Job Entry. The parameters are used during RMTGEN to specify hardware configuration and software options.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The parameters are listed in alphabetical order.

&CCT

Explanation: Variable symbol &CCT specifies the degree of text compression to be provided in the program. For text transmission to HASP, the program will compress strings of &CCT or more identical characters. The specification must be an integer between 3 and 80 inclusive.

Default: &CCT=4

Notes:
1. Low values of &CCT cause creation of highly compact records, increasing effective line speed at the expense of CPU

&CORESIZ

Explanation: Variable symbol &CORESIZ specifies the amount of main storage available to the program in K bytes. The specification must be an integer greater than 0.

Default: &CORESIZ=8.

&NUMBUFS

Explanation: Variable symbol &NUMBUFS specifies the maximum number of buffers to be used by the program. The specification must be an integer greater than or equal to 2.

Default: &NUMBUFS=10

Notes:
1. The length of each buffer is given by HASPGEN parameter &TPBFSIZ.

&PUNCH

Explanation: Variable symbol &PUNCH specifies inclusion (&PUNCH=1) or exclusion (&PUNCH=0) of support for a card punch attached to the Model 20. The specification must be either 0 or 1.

Default: &PUNCH=1

Notes:
1. The program supports 1442, 2520, and 2560 card punches interchangeably.

7.3 RMTGEN PARAMETERS FOR SYSTEM/360 MODEL 20 BSC

This section describes the parameters used in assembly of the System/360 Model 20 BSC Remote Terminal Program for HASP MULTI-LEAVING Remote Job Entry. The parameters are used during RMTGEN to specify hardware configuration and software options.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The parameters are listed in alphabetical order.

&CCT

Explanation: Variable symbol &CCT specifies for all text compression but trailing blank compression the minimum number of characters to be compressed. A duplicate character string of fewer than &CCT characters will be treated as a string of non-duplicate characters for compression purposes. The specification must be an integer between 3 and 31, inclusive.

Default: &CCT=4

Notes:
1. See also &CMPTYPE. The value of &CCT is not used if &CMPTYPE=1.
2. A smaller value of &CCT increases efficiency of communication line usage at the expense of compute time required for compression.

&CMPTYPE

Explanation: Variable symbol &CMPTYPE specifies the type of compression to be applied to all text transmitted from the Model 20 to the central computer. The specification must be either 1, 2, or 3. The value 1 specifies trailing blank compression; 2 specifies compression of leading, embedded, and trailing blanks, and 3 specifies compression of all duplicate character strings.

Default: &CMPTYPE=2

Notes:
1. See also &CCT.

&CORESIZ

Explanation: Variable symbol &CORESIZ specifies the size of Model 20 main storage in Kbytes (1 Kbyte = 1024 bytes). The specification must be an integer between 8 and 32 inclusive.

Default: &CORESIZ=8

&ERRMSGN

Explanation: Variable symbol &ERRMSGN specifies the number of four-byte entries to be assembled in the System/360 Remote Terminal as an error message log table. The specification must be an integer not less than 8.

Default: &ERRMSGN=8

&LINESPD

Explanation: Variable symbol &LINESPD specifies the speed, in baud, of the communication line to be used between the Model 20 and the central computer. The specification must be a positive integer.

Default: &LINESPD=2000

&NUMBUFS

Explanation: Variable symbol &NUMBUFS specifies number of teleprocessing buffers to be constructed by the Model 20 Remote Terminal program. The specification must be an integer no less than given by the formula

2*X+1

where

X=1 if either a 2520 or a 2560 is to be used as both a reader and a punch, or
0 otherwise.

Default: &NUMBUFS=8

Notes:

1. The length of each buffer is &MLBFSIZ+5 bytes (rounded up to the next full word); the value of HASPGEN parameter &MLBFSIZ is automatically propagated to RMTGEN.
2. If &NUMBUFS specifies more buffers than can be built in available storage, the Remote Terminal program will build as many buffers as it can.
3. It is recommended that at least two buffers be furnished for each output device and for the communication adapter.

&NUMTANK

Explanation: Variable symbol &NUMTANK specifies the number of decompression buffers ("decompression tanks") to be assembled in the Model 20 Remote Terminal program. The specification should be an integer not less than 2.

Default: &NUMTANK=8

Notes:
1. The length of each decompression tank is &PRTSIZE+6.
2. It is recommended that at least two tanks each be provided for the printer and the punch.
3. For an 8K Model 20, specification of &NUMTANK greater than 8 may cause the Remote Terminal program to assemble larger than X'1F00' bytes (8K-256); the resultant program will fail to load.

&PDEV(1)

Explanation: Subscripted variable symbol &PDEV(1) specifies device type for the Model 20 printer. The specification must be either 1403 or 2203.

Default: &PDEV(1)=2203

&PRTCONS

Explanation: Variable symbol &PRTCONS specifies degree of use of the printer as an output console. The specification must be either 0, 1, or 2.

For &PRTCONS=0, the printer will never be used as an output console.

For &PRTCONS=1, the printer will be used as an output console. If the Model 20 receives an operator message while the printer is printing a job, the message will be held in a decompression tank. If all but two of the decompression tanks contain messages, the job's printing will be stopped, a page ejected, the messages printed, another page will be ejected, and the job will resume printing.

For &PRTCONS=2, the printer will be used as an output console only if it is not printing a job. Otherwise, operator messages received from the central computer will be thrown away.

Default: &PRTCONS=0

Notes:
1. If &WDEV(1) is not specified as zero, &PRTCONS is not used and the printer will never be used as an output console.
2. If &PRTCONS=1 or &PRTCONS=2, console support must be indicated for this Remote Terminal at HASPGEN time. See HASPGEN parameter RMTnn.

&PRTSIZE

Explanation: Variable symbol &PRTSIZE specifies the length in bytes of the text portion of each decompression tank. Each tank must be long enough to hold a maximum-length output record to either the printer, the punch, or the operator console. The specification must be an integer that is the largest of 80 (if &UDEV(1) is not zero), 120 (if &WDEV(1) is not zero), and the line width of the printer.

Default: &PRTSIZE=120

&RADR(1)

Explanation: Subscripted variable symbol &RADR(1) specifies the unit address of the Model 20 card reader. The specification must correspond to the specification for &RDEV(1) as follows:

&RDEV(1)	&RADR(1)
2501	1
2520	2
2560	2

Default: &RADR(1)=1

&RDEV(1)

Explanation: Subscripted variable symbol &RDEV(1) specifies device type for the Model 20 card reader. The specification must be either 2501, 2520, or 2560.

Default: &RDEV(1)=2501

Notes:
1. See also &RADR(1)

&SUBMOD

Explanation: Variable symbol &SUBMOD specifies the Submodel number of the System/360 Model 20 for the specified Remote Terminal. The specification must be a valid System/360 Model 20 Submodel number.

Default: &SUBMOD=2

&UADR(1)

Explanation: Subscripted variable symbol &UADR(1) specifies the unit address of the Model 20 card punch. The specification must correspond to the specification for &UDEV(1) as follows:

&UDEV(1)	&UADR(1)
1442	3
2520	2
2560	2
0	not used

Default: &UADR(1)=3

&UDEV(1)

Explanation: Subscripted variable symbol &UDEV(1) specifies device type for the Model 20 card punch. The specification must be either 1442, 2520, 2560, or 0. Specification 0 is used when the Model 20 does not include a card punch.

Default: &UDEV(1)=1442

Notes:
1. See also &UADR(1), unless &UDEV(1)=0.

&WDEV(1)

Explanation: Subscripted variable symbol &WDEV(1) specifies device type for the Model 20 console. The specification must be either 2152 (if a console is present) or 0 (if no console is present).

Default: &WDEV(1)=0

Notes:
1. If &WDEV(1)=2152, console support must be indicated for this Remote Terminal at HASPGEN time. See HASPGEN parameter RMTnn.

&WTOSIZE

Explanation: Variable symbol &WTOSIZE specifies the maximum length in bytes of a HASP operator command to be transmitted from the Model 20 to the central computer. The specification must be a positive integer.

Default: &WTOSIZE=120

Notes:
1. If &WDEV(1)=0, this parameter is not used.

&XPARENT

Explanation: Variable symbol &XPARENT specifies presence or absence of the text transparency feature. If the Binary Synchronous Communication Adapters at both the Model 20 and the central computer have the text transparency feature, YES should be specified; otherwise NO should be specified.

Default: &XPARENT=YES

7.4 RMTGEN PARAMETERS FOR SYSTEM/360 (EXCEPT MODEL 20) BSC

This section describes the parameters used in assembly of the System/360 BSC Remote Terminal Program for HASP MULTI-LEAVING Remote Job Entry. The parameters are used during RMTGEN to specify hardware configuration and software options.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The parameters are listed in alphabetical order.

&ADAPT

Explanation: Variable symbol &ADAPT specifies the unit address of the Binary Synchronous Communication Adapter to be used by the System/360 Remote Terminal to communicate with HASP at the central computer. The specification must be a valid unit address.

Default: &ADAPT=020

&CCT

Explanation: Variable symbol &CCT specifies for all text compression but trailing blank compression the minimum number of characters to be compressed. A duplicate character string of fewer than &CCT characters will be treated as a string of non-duplicate characters for compression purposes. The specification must be an integer between 3 and 31, inclusive.

Default: &CCT=4

Notes:

1. See also &CMPTYPE. The value of &CCT is not used if &CMPTYPE=1.
2. A smaller value of &CCT increases efficiency of communication line usage at the expense of compute time required for compression.

&CMPTYPE

Explanation: Variable symbol &CMPTYPE specifies type of compression to be applied to all text transmitted from the System/360 Remote Terminal to the central computer. The specification must be either 1, 2, or 3. The value 1 specifies trailing blank compression; 2 specifies compression of leading, embedded, and trailing blanks, and 3 specifies compression of all duplicate character strings.

Default: &CMPTYPE=2

Notes:
1. See also &CCT.

&CORESIZ

Explanation: Variable symbol &CORESIZ specifies the size of main storage for the System/360 Remote Terminal in Kbytes (1 Kbyte = 1024 bytes). The specification must be an integer between 8 and 32 inclusive. If the System/360 is larger than 32 Kbytes, &CORESIZ must be specified as 32.

Default: &CORESIZ=32

&ERRMSGN

Explanation: Variable symbol &ERRMSGN specifies the number of four-byte entries to be assembled in the Model 20 Remote Terminal program as an error message log table. The specification must be an integer not less than 8.

Default: &ERRMSGN=8.

&LINESPD

Explanation: Variable symbol &LINESPD specifies the speed, in baud, of the communication line to be used between the System/360 Remote Terminal and the central computer. The specification must be a positive integer.

Default: &LINESPD=2000

&MACHINE

Explanation: Variable symbol &MACHINE specifies the model number of the System/360 to be used as a HASP Remote Terminal. The specification must be a valid System/360 model number for a System/360 which includes the standard instruction set and the decimal instruction set.

Default: &MACHINE=30

&NUMBUFS

Explanation: Variable symbol &NUMBUFS specifies number of teleprocessing buffers to be constructed by the System/360 Remote Terminal program. The specification must be an integer no less than given by the formula

$$2*X+1$$

where

X=1 if either a 2520 or a 1442 is to be used as both a reader and a punch, or
0 otherwise.

Default: &NUMBUFS=8

Notes:

1. The length of each buffer is &MLBFSIZ+5 bytes (rounded up to a multiple of 4); the value of HASPGEN parameter &MLBFSIZ is automatically propagated to RMTGEN.
2. If &NUMBUFS specifies more buffers than can be built in available storage, the Remote Terminal program will build as many buffers as it can.
3. It is recommended that at least two buffers be furnished for each output device and for the communication adapter.

&NUMTANK

Explanation: Variable symbol &NUMTANK specifies the number of decompression buffers ("decompression tanks") to be assembled in the System/360 Remote Terminal program. The specification should be an integer not less than 2.

Default: &NUMTANK=8

Notes:
1. The length of each decompression tank is &PRTSIZE+6.
2. It is recommended that at least two tanks be provided for each printer and each punch (3 for a 2540 punch).

&PADR(n)

Explanation: Subscripted variable symbols &PADR(n) specify unit addresses for the printers defined by &PDEV(n). For each &PDEV(n) not specified as zero, the corresponding symbol &PADR(n) must specify the device's 3-character hexadecimal unit address.

Default: &PADR(1)=00E
&PADR(2)=00F
&PADR(3)=FFF
&PADR(4)=FFF
&PADR(5)=FFF
&PADR(6)=FFF
&PADR(7)=FFF

&PDEV(n)

Explanation: Subscripted variable symbols &PDEV(n) specify the existence and device types of the Remote Terminal printers. Each specification must be either 1403, 1443, or 0. A specification of 0 indicates that the associated printer does not exist.

Default: &PDEV(1)=1403
&PDEV(2)=0
&PDEV(3)=0
&PDEV(4)=0
&PDEV(5)=0
&PDEV(6)=0
&PDEV(7)=0

Notes:

1. If &PDEV(n) is specified as a device type, then &UDEV(8-n) must be specified as zero.
2. If &PDEV(n+1) is specified as a device type, then &PDEV(n) must be specified as a device type.
3. If more than one printer is specified, a Device Control Table (DCT) for each additional printer must be added to the HASP System.

&PRTSIZE

Explanation: Variable symbol &PRTSIZE specifies the length in bytes of the text portion of each decompression tank. Each tank must be long enough to hold a maximum-length output record to either a printer, a punch, or the operator console. The specification must be an integer that is the larger of 120 and the line width of the widest printer.

Default: &PRTSIZE=132

&RADR(n)

Explanation: Subscripted variable symbols &RADR(n) specify unit addresses for the readers defined by &RDEV(n). For each &RDEV(n) not specified as zero, the corresponding symbol &RADR(n) must specify the device's 3-character hexadecimal unit address.

Default: &RADR(1)=00C
&RADR(2)=FFF
&RADR(3)=FFF
&RADR(4)=FFF
&RADR(5)=FFF
&RADR(6)=FFF
&RADR(7)=FFF

&RDEV(n)

Explanation: Subscripted variable symbols &RDEV(n) specify the existence and device types of the Remote Terminal readers. Each specification must be either 2540, 2501, 2520, 1442, or 0. A specification of 0 indicates that the associated reader does not exist.

Default:
```
&RDEV(1)=2540
&RDEV(2)=0
&RDEV(3)=0
&RDEV(4)=0
&RDEV(5)=0
&RDEV(6)=0
&RDEV(7)=0
```

Notes:
1. If &RDEV(n+1) is specified as a device type, then &RDEV(n) must be specified as a device type.
2. If more than one reader is specified, a Device Control Table (DCT) for each additional reader must be added to the HASP System.

&UADR(n)

Explanation: Subscripted variable symbols &UADR(n) specify unit addresses for the punches defined by &UDEV(n). For each &UDEV(n) not specified as zero, the corresponding symbol &UADR(n) must specify the device's 3-character hexadecimal unit address.

Default: &UADR(1)=00D
&UADR(2)=FFF
&UADR(3)=FFF
&UARD(4)=FFF
&UARD(5)=FFF
&UARD(6)=FFF
&UARD(7)=FFF

&UDEV(n)

Explanation: Subscripted variable symbols &UDEV(n) specify the existence and device types of the Remote Terminal punches. Each specification must be either 2540, 2520, 1442, or 0. A specification of 0 indicates that the associated punch does not exist.

Default: &UDEV(1)=2540
&UDEV(2)=0
&UDEV(3)=0
&UDEV(4)=0
&UDEV(5)=0
&UDEV(6)=0
&UDEV(7)=0

Notes:
1. If &UDEV(n) is specified as a device type, then &PDEV(8-n) must be specified as zero.
2. If &UDEV(n+1) is specified as a device type, then &UDEV(n) must be specified as a device type.
3. If more than one punch is specified, a Device Control Table (DCT) for each additional punch must be added to the HASP System.

&WADR(1)

Explanation: Subscripted variable symbol &WADR(1) specifies the unit address of the 1052 operator console on the System/360 Remote Terminal. The specification must be a 3-character hexadecimal unit address.

Default: &WADR(1)=01F

&WTOSIZE

Explanation: Variable symbol &WTOSIZE specifies the maximum length in bytes of a HASP operator command to be transmitted from the System/360 Remote Terminal to the central computer. The specification must be a positive integer.

Default: &WTOSIZE=120

&XPARENT

Explanation: Variable symbol &XPARENT specifies presence or absence of the text transparency feature. If the Binary Synchronous Communication Adapters at both the System/360 Remote Terminal and the central computer have the text transparency feature, YES should be specified; otherwise NO should be specified.

Default: &XPARENT=YES

7.5 RMTGEN PARAMETERS FOR 1130

This section describes the parameters used in assembly of the 1130 Remote Terminal Program for HASP MULTI-LEAVING Remote Job Entry. The parameters are used during RMTGEN to specify hardware configuration and software options.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The parameters are listed in alphabetical order.

&CLOCK

Explanation: The variable symbol &CLOCK is used to specify the type of communication adapter clocking available on the 1130 to be used by the workstation program. The specification of &CLOCK=0 is interpreted to mean that data set clocking is being used. The value &CLOCK=1 specifies internal (1130) clocking.

Default: &CLOCK=0

Notes:

1. The rate of insertion of the synchronous idle sequence in the transmitted data is determined by the variables &CLOCK, &LINESPD and &TRANPRN. The relationship of these variables to the insertion rate is:

&CLOCK	&TRANPRN	INSERTION EVERY:	
0	0	&LINESPD/8	characters
0	1	&LINESPD/8	characters
1	0	70	characters
1	1	&LINESPD/8	characters

2. The equation used for the insertion rate is:

   (&LINESPD/8)*T

   where T is 1.00 second which is the nominal 2701 timer value.

&CMPTYPE

Explanation: The variable symbol &CMPTYPE is used to specify the compression technique that is to be applied to the data transmitted to the central HASP system. The choices for &CMPTYPE are:

&CMPTYPE=0	for no compression of duplicate characters or truncation of trailing blanks.
&CMPTYPE=1	for trailing blank truncation only.
&CMPTYPE=2	for full compression: trailing blank truncation and encoding of duplicate characters.

Default: &CMPTYPE=2

Notes:

1. The process of compressing input data offers optimum performance with respect to efficient line utilization. However, the factors of line speed, CPU availability, buffer size, line turn-around time, nature of the data to be compressed, etc., are variables which contribute to the overall operation of the workstation program. Since compression and truncation require considerable CPU time, the user may decide, on the basis of the other variables, to respecify the compression technique.

&DELAY

Explanation: The variable symbol &DELAY is used to define the number of intervals of time that RTP1130 will delay in transmitting a "handshaking" sequence (DLE-ACK0) to the central HASP site. The hardware program timer clock is used to measure the delay and is assumed to be set to a nominal value of .35 seconds.

Default: &DELAY=3

Notes:
1. &DELAY=3 results in a delay of 1.05 seconds, assuming a timer interval of .35 seconds.
2. The purpose of the delay when "handshaking" is to minimize CPU processing at the central HASP computer when no data is being transmitted.
3. The value of &DELAY must not be set to such a large increment that the delay will be greater than the timeout period of the central site 2701/2703.

&FULLIST

Explanation: The variable symbol &FULLIST is used to specify the type of assembly listing which is produced by the OS/360 assembler during the RMTGEN process. If the value of &FULLIST is set to 0, then the assembly listing produced will be according to the PRINT NOGEN stipulation of the assembler. If the value of &FULLIST is set to 1, the listing will be produced according to the PRINT GEN stipulation.

Default: &FULLIST=1

Notes:

1. Since most of the code in RTP1130 and RTPLOAD is created by Macro instructions, the specification of &FULLIST=0 will essentially produce a source listing (cross referenced) without the 1130 assembled instructions. Error messages will not appear on the listing.

&LINESPD

Explanation: The variable symbol &LINESPD is used to specify the baud rate for the communication line interface to the workstation program. The value should correspond to the selected setting of the baud rate switch on the 1130 SCA control panel: 1200,2000,...,etc.

Default: &LINESPD=2000

Notes:

1. The rate of insertion of the synchronous idle sequence (DLE-SYN or SYN-SYN) in the transmitted data is determined by the variables &CLOCK, &LINESPD and &TRANPRN. See note 1 of &CLOCK description.

&MACHSIZ

Explanation: Variable symbol &MACHSIZ specifies the amount of 1130 core to be used by RTP1130. The value of &MACHSIZ is in units of 1130 words.

Default: &MACHSIZ=8192

Notes:
1. The value of &MACHSIZ is interpreted to mean that "&MACHSIZ" number of words, starting at location 0, are available for the workstation program consisting of RTPBOOT, RTPLOAD and RTP1130.
2. The same variable symbol must be defined for RTPLOAD and should have the same value.
3. The value of &MACHSIZ may be less than the actual available storage but must not be greater.

&PN1442

Explanation: The variable symbol &PN1442 is used to define a 1442 punch. If the variable is set to 1, then RTP1130 will include support for punched card output produced by jobs at the Central HASP site. If the variable is set to 0, no support for the 1442 punch will be provided. See &RD1442 for the definition of a reader function on the 1442.

Default: &PN1442=1

&PRFOTLW

Explanation: The value of the variable symbol &PRFOTLW is used to define the line width of the 1403 printer specified by &PR1403. The choices are 120 or 132 character lines.

Default: &PRFOTLW=120

Notes:

1. The definition of the line width for all printers on a particular remote is a HASPGEN requirement. See HASPGEN parameter RMTnn.

&PR1132

Explanation: The variable symbol &PR1132 is used to define an 1132 printer. If the variable is set to 1, then RTP1130 will include support for the 1132 to print job output. If the variable is set to 0, no support will be included in RTP1130 for the 1132.

Default: &PR1132=0

&PR1403

Explanation: The variable symbol &PR1403 is used to define a 1403 printer for use as an output device. If the value of &PR1403 is 1, then the 1403 function will be included in RTP1130. If the value is 0, the function is deleted from RTP1130.

Default: &PR1403=1

Notes:
1. See &PRFOTLW for specifying the line width of the 1403.

&RD1442

Explanation: The variable symbol &RD1442 is used to define a 1442 as a card reader. If the variable is set to 1, then RTP1130 will be assembled with all necessary control blocks and support routines to provide job input from the 1442. If the variable is set to 0, no support for the 1442 reader will be provided in RTP1130. See &PN1442 for a definition of the punch function on the 1442.

Default: &RD1442=1

Notes:

1. If the variable &RD1442 is set to 1 and a 1442 reader does not exist then the operation of the workstation program may be unpredictable.

&RD2501

Explanation: The variable symbol &RD2501 is used to define a 2501 card reader. If the variable is set to 1, then RTP1130 will be assembled with all necessary control block and subroutines to support the 2501 as a job input device. If the variable is set to 0, no support for the 2501 will be included in RTP1130.

Default: &RD2501=0

Notes:

1. If the variable &RD2501 is set to 1 and a 2501 does not exist then the operation of the work-station program will be unpredictable and usually unproductive.

&RTPLORG

Explanation: The variable symbol &RTPLORG defines the origin in 1130 storage of the program loader RTPLOAD which is used to load RTP1130.

Default: &RTPLORG=2*(&MACHSIZ-1024)

Notes:

1. The value of the above expression, assuming &MACHSIZ=8192, is 14336 (which is twice the actual 1130 storage address because the value is used in an ORG operation and must be in terms of bytes not 1130 words.
2. The RTPLOAD program must origin in the storage available between the end of RTP1130 (beginning of buffer pool) and the end of defined (&MACHSIZ) storage MINUS the length of RTPLOAD. The default value of &RTPLORG allows for an RTPLOAD of 1024 words in size.

&TRANPRN

Explanation: The variable symbol &TRANPRN is used to define the simulation of the Binary Synchronous Transparency feature. If the value of &TRANPRN is set to 1, then RTP1130 will simulate the transparency feature in the same manner as the 2701 SDA-II adapter equipped with the transparency feature. If the variable is set to 0, no simulation will occur and therefore data which contains transparent characters cannot be properly processed by RTP1130.

Default: &TRANPRN=1

Notes:

1. If &TRANPRN=0 is specified, the conversion of card code data is monitored and all BSC control characters are converted to hexadecimal 0. This prevents mispunched data from causing an infinite error retry if the central site does not have transparency.
2. See &LINESPD and &CLOCK for additional influence of &TRANPRN.
3. If &TRANPRN=1, the generated Remote Terminal program will communicate only with a 2701 or 2703 adapter which has the text transparency feature.

7.6 RMTGEN PARAMETERS FOR 1130 LOADER

This section describes the parameters used in assembly of RTPLOAD, the 1130 Loader Program. RTPLOAD is used to load the 1130 Remote Terminal Program. RTPLOAD's three parameters specify machine size, loader origin, and an assembler list option.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The RMTGEN processes produce the object decks for RTPLOAD and RTP1130. The bootstrap loader (RTPBOOT) cannot be produced on a System 360 and must be punched by keypunch as indicated in Section 4.14.3.

The parameters are listed in alphabetical order.

&FULLLIST

Explanation: The variable symbol &FULLLIST is used to specify the type of assembly listing which is produced by the OS/360 assembler during the RMTGEN process. If the value of &FULLLIST is set to 0, then the assembly listing produced will be according to the PRINT NOGEN stipulation of the assembler. If the value of &FULLLIST is set to 1, the listing will be produced according to the PRINT GEN stipulation.

Default: &FULLLIST=1

Notes:

1. Since most of the code in RTP1130 and RTPLOAD is created by Macro instructions, the specification of &FULLLIST=0 will essentially produce a source listing (cross referenced) without the 1130 assembled instructions. Error messages will not appear on the listing.

&MACHSIZ

Explanation: Variable symbol &MACHSIZ specifies the amount of 1130 core to be used by RTPLOAD. The value of &MACHSIZ is in units of 1130 words.

Default: &MACHSIZ=8192

Notes:
1. The value of &MACHSIZ is interpreted to mean that "&MACHSIZ" number of words, starting at location 0, are available for the workstation program consisting of RTPBOOT, RTPLOAD and RTP1130.
2. The same variable symbol must be defined for RTP1130 and should have the same value.
3. The value of &MACHSIZ may be less than the actual available storage but must not be greater.

&RTPLORG

Explanation: The variable symbol &RTPLORG defines the origin in 1130 storage of the program loader RTPLOAD which is used to load RTP1130.

Default: &RTPLORG=2*(&MACHSIZ-1024)

Notes:

1. The value of the above expression, assuming &MACHSIZ=8192, is 14336 (which is twice the actual 1130 storage address because the value is used in an ORG operation and must be in terms of bytes not 1130 words.
2. The RTPLOAD program must origin in the storage available between the end of RTP1130 (beginning of buffer pool) and the end of defined (&MACHSIZ) storage MINUS the length of RTPLOAD. The default value of &RTPLORG allows for an RTPLOAD of 1024 words in size.

7.7 RMTGEN PARAMETERS FOR SYSTEM/3

This section describes the parameters used in assembly of the System/3 Remote Terminal Program for HASP MULTI-LEAVING Remote Job Entry. The parameters are used during RMTGEN to specify hardware configuration and software options.

For each parameter there is an explanation, the default value, and frequently notes which expand upon the explanation.

The parameters are listed in alphabetical order.

&COMP

Explanation: Variable symbol &COMP specifies degree of text compression to be provided for all text transmitted from the System/3 to HASP. The specification must be either 0, 1, or 2.

For &COMP=0, neither compression nor truncation is performed.

For &COMP=1, trailing blanks are truncated from each logical record before it is transmitted.

For &COMP=2, compression takes place after truncation. Strings of from two to 31 blanks are compressed to a single byte; strings of from three to 31 duplicate characters are compressed to two bytes.

Default: &COMP=2

&DEBUG

Explanation: Variable symbol &DEBUG specifies inclusion or exclusion of certain validity tests and a core dump program in the System/3 Remote Terminal Program. The specification must be either 0 or 1.

Default: &DEBUG=0

&DIAL, &DIAL1

Explanation: Variable symbols &DIAL and &DIAL1 specify the telephone number to be used during the initialization process. The values will be included on the default /*SIGNON card assembled into the System/3 Remote Terminal Program and preceded by the keyword DIAL (unless the parameters are left at their defaults). Each specification is a string of from one to eight decimal digits. If the telephone number is eight or fewer digits long, it should be specified by &DIAL. If the telephone number is longer than eight digits, its leftmost eight digits should be specified by &DIAL and the remaining digits by &DIAL1.

Default: &DIAL=[null string]
&DIAL1=[null string]

&MACHSIZ

Explanation: Variable symbol &MACHSIZ specifies the size of System/3 Core storage. The specification should be either 8192, 12288, 16384, 24576, or 32768 for core storage sizes of 8K, 12K, 16K, 24K or 32K respectively.

Default: &MACHSIZ=8192

&PASSWD

Explanation: Variable symbol &PASSWD specifies a password to be used during the SIGNON process. The value will be included on the default /*SIGNON card assembled into the System/3 Remote Terminal Program. The specification must be a character string of from one to eight characters. If blanks are desired, no specification may be made.

Default: &PASSWD=[null string]

&PC(n)

Explanation: Subscripted variable symbols &PC(n) specify skip information for the 5203 printer. The value to which &PC(n) is set will be the print line number to which paper will be skipped when the System/3 Remote Terminal Program simulates the 1403 command "Skip to Channel n". Each specification must be an integer between 0 and &S3FORML, inclusive. A specification of 0 causes no forms movement.

Default:
```
&PC(1)=1
&PC(2)=0
&PC(3)=0
&PC(4)=0
&PC(5)=0
&PC(6)=0
&PC(7)=0
&PC(8)=0
&PC(9)=0
&PC(10)=0
&PC(11)=0
&PC(12)=&S3FORML-5
```

&PRTCONS

Explanation: Variable symbol &PRTCONS specifies utilization of the 5203 printer as an operator's output console. The specification must be 0, 1, or 2.

For &PRTCONS=0, the 5203 printer will never be used as an operator's output console.

For &PRTCONS=1, the System/3 Remote Terminal Program will attempt to hold operator messages from HASP until a job has completed printing. However, if two or more MULTI-LEAVING buffers are received containing HASP operator messages, the 5203 will eject a page (skip to channel 1), print the HASP operator messages, eject another page, and resume printing its job.

For &PRTCONS=2, the System/3 Remote Terminal Program will throw away all operator messages while the 5203 is printing a job. While the 5203 is dormant , it will print any received messages.

Default: &PRTCONS=2

Notes:
1. If &S35471=1, the value of &PRTCONS is ignored and assumed to be zero.

&S3FORML

Explanation: Variable symbol &S3FORML specifies the number of print lines on a page of the continuous forms which will be used in the 5203 printer. The specification must be an integer not less than 6.

Default: &S3FORML=66

&S3NPUNS

Explanation: Variable symbol &S3NPUNS specifies the maximum number of jobs that can be punching simultaneously at the System/3 Remote Terminal. The specification must be 1, 2, or 3. (A value of 3 allows simultaneous operation of both 5424 hoppers and the 1442 hopper as punches.)

Default: &S3NPUNS=1

Notes:

1. If &S3NPUNS is set to 2 or 3, extra device control tables must be added for the appropriate remote to the HASP System at HASPGEN time.

&S3NRDRS

Explanation: Variable symbol &S3NRDRS specifies the maximum number of job streams that can be reading simultaneously from the System/3 Remote Terminal. The specification must be 1, 2, or 3. (A value of 3 allows simultaneous operation of both 5424 hoppers and the 1442 hopper as readers.)

Default: &S3NRDRS=1

Notes:

1. If &S3NRDRS is set to 2 or 3, extra device control tables must be added for the appropriate remote to the HASP System at HASPGEN time.

&S3OBJDK

Explanation: Variable symbol &S3OBJDK specifies inclusion of a facility to punch Operating System object decks. Text transparency should be present. The specification should be 0 or 1.

If &S3OBJDK=1, each card of an OS object deck will be expanded and punched into two 96-column cards. These cards will be recognized when later read by a System/3 Remote Terminal program for which &S3OBJDK=1, and for each two 96-column cards read an OS object deck card image will be transmitted.

Default: &S3OBJDK=0

&S3SIP

Explanation: Variable symbol &S3SIP specifies usage of those bytes of System/3 core storage between X'100' and X'1FF', inclusive. The specification must be either 0 or 1. For &S3SIP=1, the System/3 Remote Terminal Program will not use the bytes; their values will be preserved for the use of the System/3 Card System Initialization Program.

Default: &S3SIP=0

&S3TRACE

Explanation: Variable symbol &S3TRACE specifies the number of four-byte entries in the System/3 Remote Terminal Program's internal error message table. The specification must be an integer greater than 1.

Default: &S3TRACE=10

&S3XPAR

Explanation: Variable symbol &S3XPAR specifies presence or absence of the EBCDIC text transparency feature. The specification should be 1 if both the central computer's communications adapter and the System/3 BSCA have the EBCDIC text transparency feature; otherwise the specification should be 0.

Default: &S3XPAR=0

&S31442

Explanation: Variable symbol &S31442 specifies inclusion or exclusion of support for the 1442 Card Reader-Punch (RPQ). The specification must be 1 for inclusion and 0 for exclusion of 1442 support.

Default: &S31442=0

Notes:

1. If &S31442=1, the resultant System/3 Remote Terminal Program requires that a 1442 be present on the System/3.

&S35471

Explanation: Variable symbol &S35471 specifies presence or absence of a 5471 Printer-Keyboard on the System/3. The 5471 will be used as an operator's input/output console. The specification must be 1 if a 5471 is present; otherwise it must be 0.

Default: &S35471=0

&S35475

Explanation: Variable symbol &S35475 specifies presence or absence of a 5475 Data Entry Keyboard on the System/3. The 5475 will be used as an operator's input console. The specification must be 1 if a 5475 is present; otherwise it must be 0.

Default: &S35475=0

Notes:
1. If &S35471=1, this parameter is ignored.

&S396COL

Explanation: Variable symbol &S396COL specifies inclusion or exclusion of the System/3 load-mode punch option. The specification must be either 0 or 1. If &S396COL is specified, the resultant System/3 Remote Terminal Program will be capable of receiving correctly the punched output of a System/3 RMTGEN.

Default: &S396COL=0

(The remainder of this page intentionally left blank.)

8.0 HASP CONTROL TABLE FORMATS

This sections contains block diagrams which depict the formats of the HASP Control Tables which are not described in other sections of this manual.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT

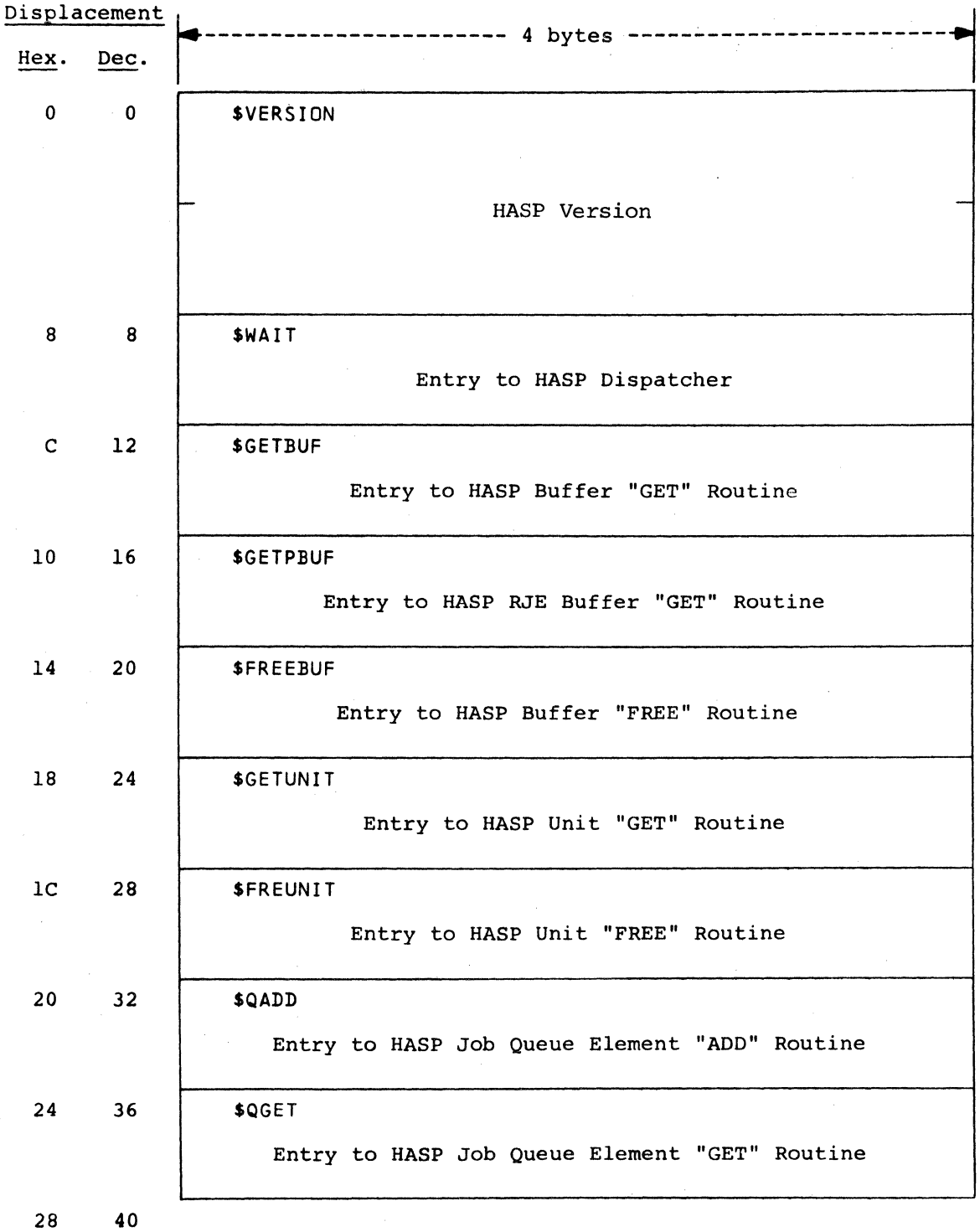


Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	4 bytes
28	40	$QPUT Entry to HASP Job Queue Element "PUT" Routine
2C	44	$QREM Entry to HASP Job Queue Element "REMOVE" Routine
30	48	$QSIZ Entry to HASP Job Queue "SIZE" Routine
34	52	$QLOC Entry to HASP Job Queue Element "LOCATE" Routine
38	56	$QJITLOC Entry to HASP JIT Element "LOCATE" Routine
3C	60	$TRACK Entry to HASP Track Allocation Routine
40	64	$PURGER Entry to HASP Track Purge Routine
44	68	$EXCP Entry to HASP Input/Output Supervisor
48	72	$EXTPOPE Entry to HASP RTAM Open Routine
4C	76	$EXTPGET Entry to HASP RTAM Get Routine
50	80	

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	4 bytes
50	80	$EXTPPUT Entry to HASP RTAM Put Routine
54	84	$EXTPCLO Entry to HASP RTAM Close Routine
58	88	$RESTORE Entry to HASP RTAM Restore Routine
5C	92	$ODEL Entry to HASP Overlay $DELETE Routine
60	96	$ORET Entry to HASP Overlay $RETURN Routine
64	100	$OLINK Entry to HASP Overlay $LINK Routine
68	104	$OXCTL Entry to HASP Overlay $XCTL Routine
6C	108	$OLOAD Entry to HASP Overlay $LOAD Routine
70	112	$WTO Entry to HASP Write-to-Operator Routine
74	116	$FREEMSG Entry to HASP Console Message Buffer Free Routine
78	120	

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	4 bytes			
78	120	$STIMER Entry to HASP Set Interval Timer Routine			
7C	124	$TTIMER Entry to HASP Test Interval Timer Routine			
80	128	$IOERROR Entry to HASP Input/Output Error Logging Routine			
84	132	$ERROR Entry to HASP Catastrophic Error Routine			
88	136	$DISTERR Entry to HASP Disastrous Error Routine			
8C	140	$SYSTYPE System Type MFT or MVT	$OPTSTAT Initialization Options	$STATUS HASP Status	RESERVED
90	144	$HASPECF Master Event Control Field	MHASPECB RJE Event Control Field	$XEQACT O/S Execution Count	$ACTIVE Active Count
94	148	$ENBALL Enable All Mask	$DISALL Disable All Mask	$DISINT Disable Int Timer Mask	RESERVED
98	152	$PSRDRCT Pseudo Reader Count (2540)	$PSPRFCT Pseudo Printer Count (1443)	$PSPUFCT Pseudo Punch Count (1442)	RESERVED
9C	156	$EXCPCT Active I/O Count		$COMMCT Active Command Count	
A0	160				

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	4 bytes
A0	160	$CKPTRAK Checkpoint Track | R E S E R V E D
A4	164	$HASPTCB Address of HASP Task Control Block
A8	168	$PCEORG Address of First HASP Processor Control Element
AC	172	$BUFPOOL Address of First Available HASP Buffer
B0	176	$TPBPOOL Address of First Available HASP RJE Buffer
B4	180	$DCTPOOL Address of First HASP Device Control Table
B8	184	$JITABLE Address of HASP Job Information Table
BC	188	$CYLMAP Address of First HASP Cylinder Module Map
C0	192	$TEDADDR Address of First Track Extent Data Table
C4	196	$DCBLIST Address of HASP Direct Access DCB
C8	200	

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

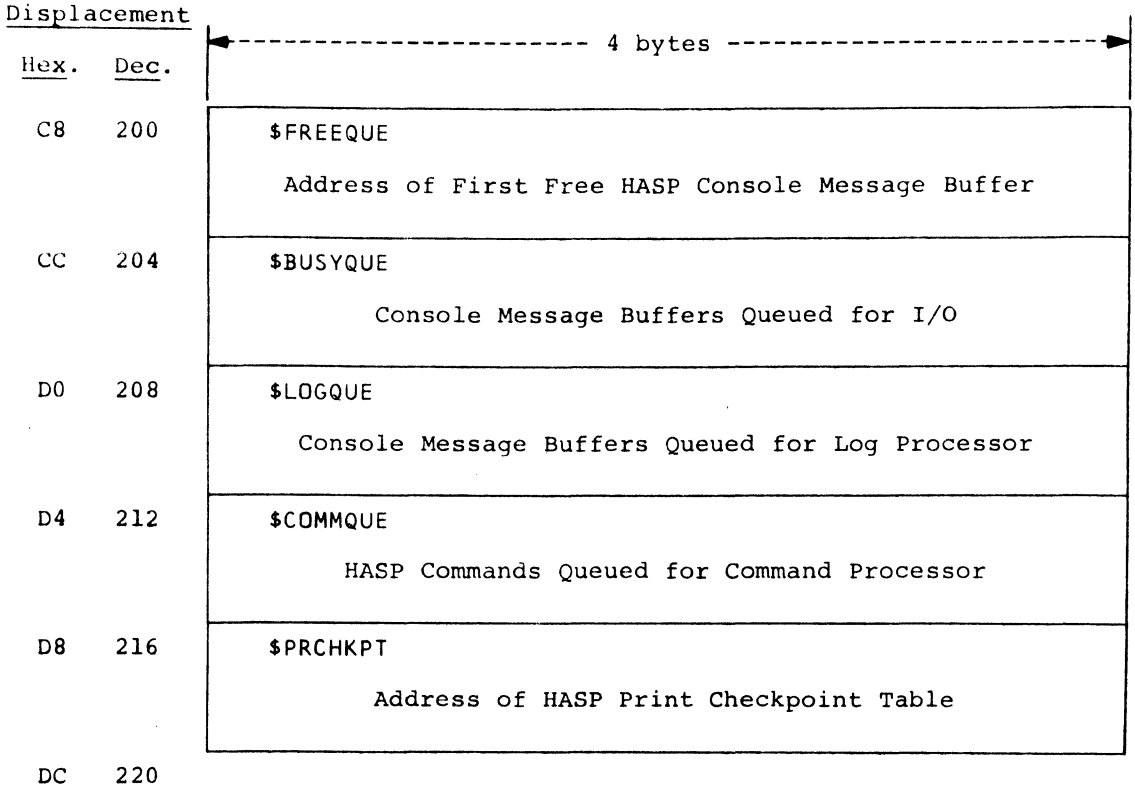


Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

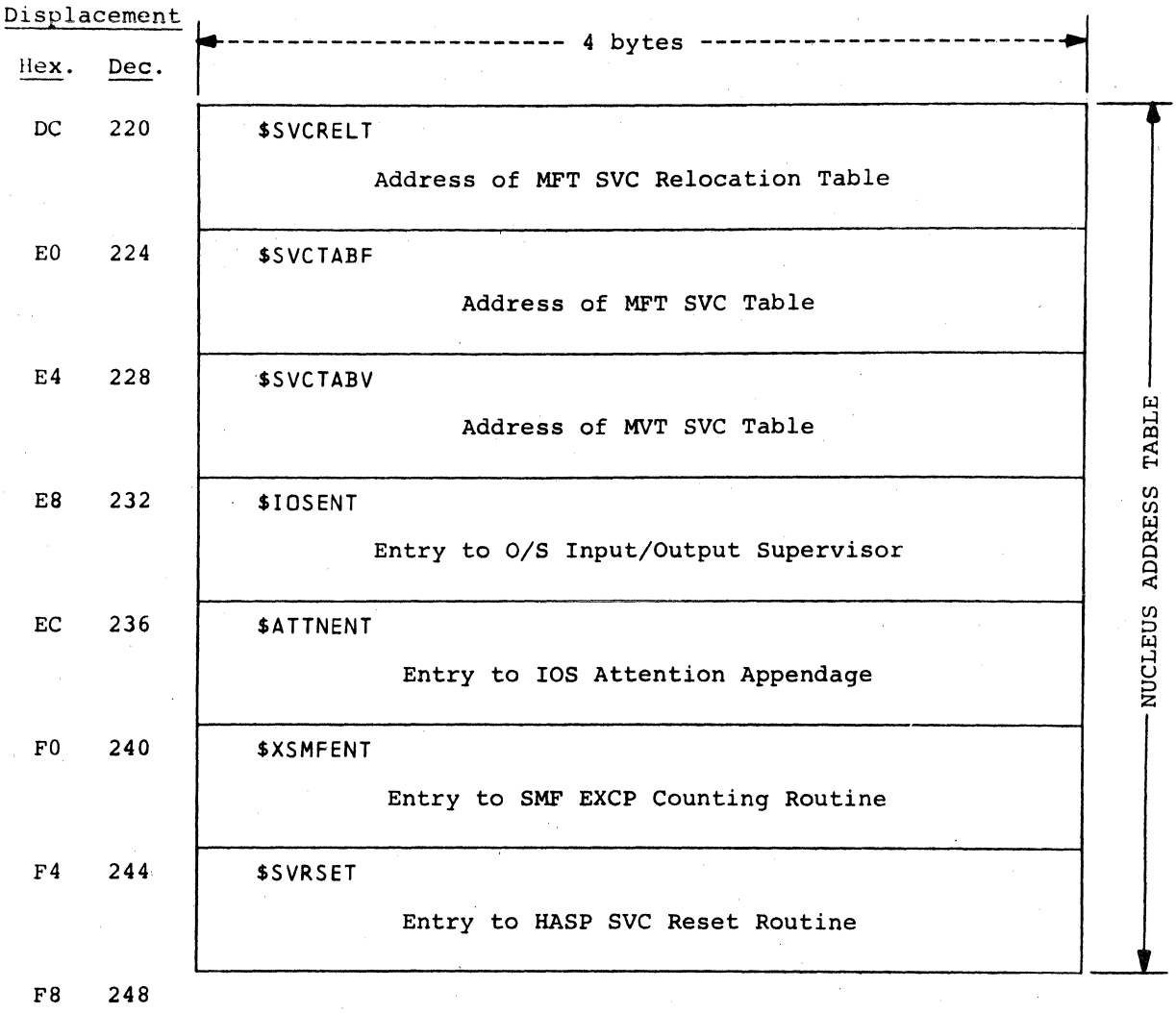


Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

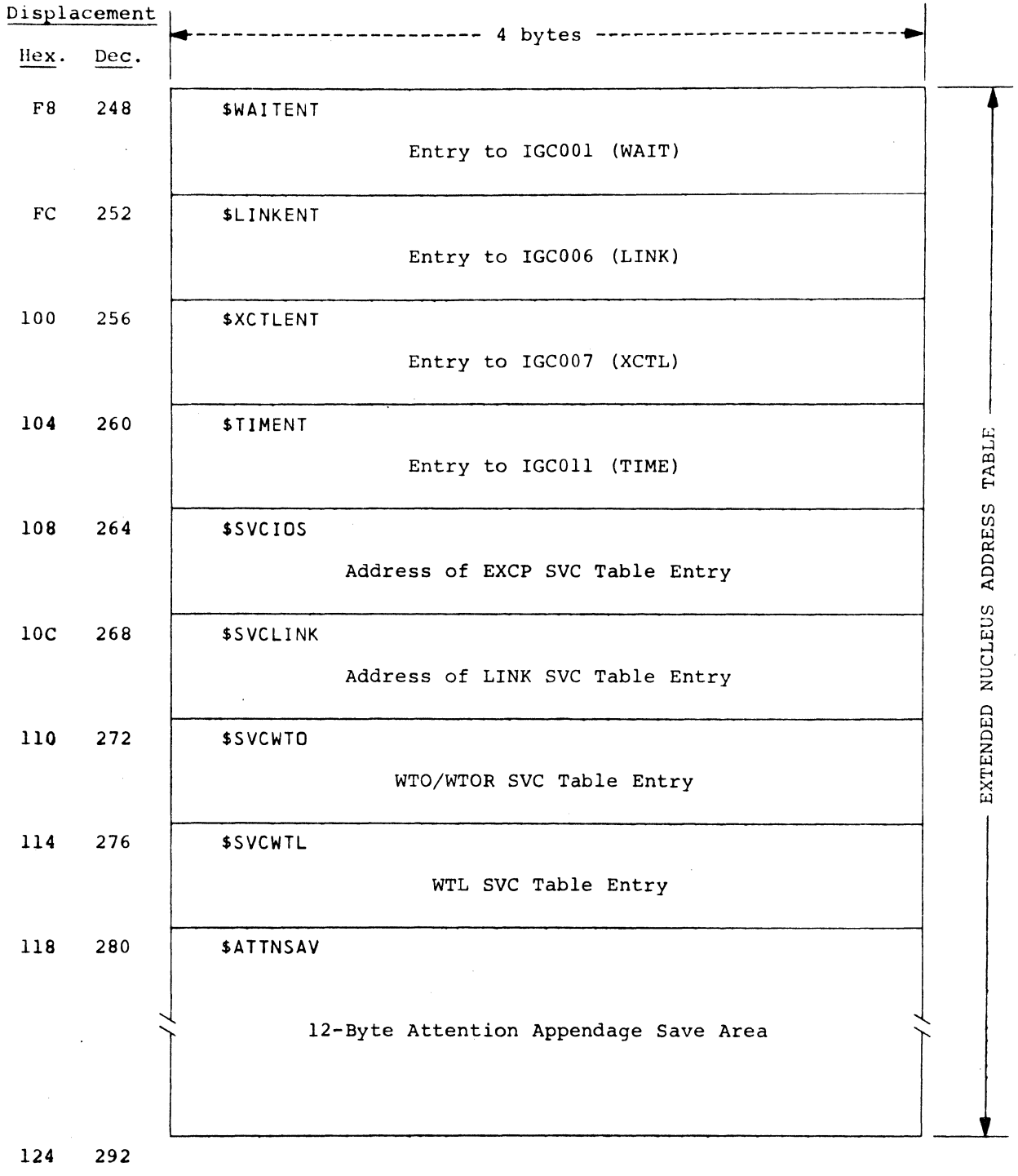


Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

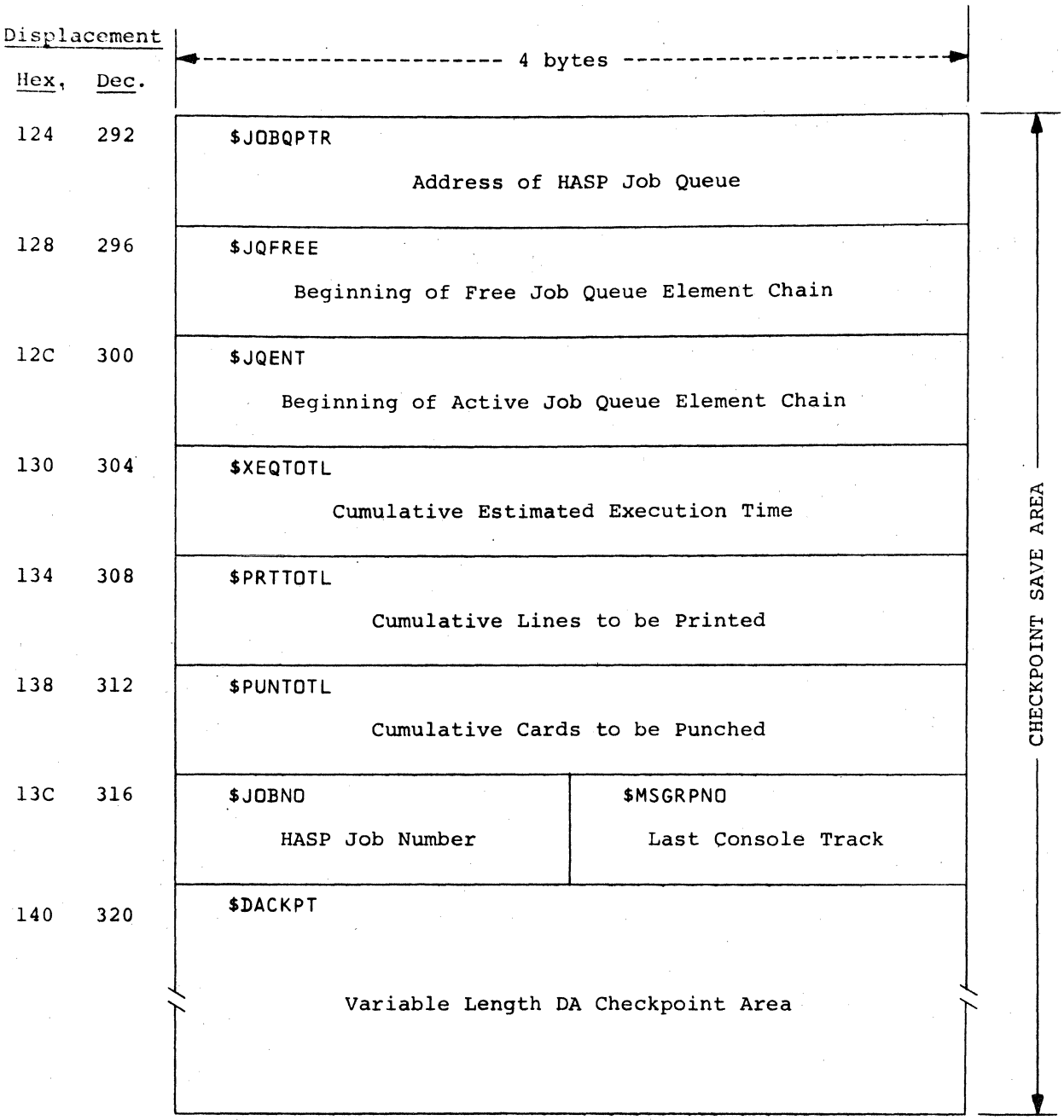


Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	$VERSION	8	HASP Version (" V v.m ").
8	8	$WAIT	4	Entry to HASP Dispatcher.
C	12	$GETBUF	4	Entry to HASP Buffer "GET" Routine.
10	16	$GETPBUF	4	Entry to HASP RJE Buffer "GET" Routine.
14	20	$FREEBUF	4	Entry to HASP Buffer "FREE" Routine.
18	24	$GETUNIT	4	Entry to HASP Unit "GET" Routine.
1C	28	$FREUNIT	4	Entry to HASP Unit "FREE" Routine.
20	32	$QADD	4	Entry to HASP Job Queue Element "ADD" Routine.
24	36	$QGET	4	Entry to HASP Job Queue Element "GET" Routine.
28	40	$QPUT	4	Entry to HASP Job Queue Element "PUT" Routine.
2C	44	$QREM	4	Entry to HASP Job Queue Element "REMOVE" Routine.
30	48	$QSIZ	4	Entry to HASP Job Queue "SIZE" Routine.
34	52	$QLOC	4	Entry to HASP Job Queue Element "LOCATE" Routine.
38	56	$QJITLOC	4	Entry to HASP Job Information Table Element "LOCATE" Routine.
3C	60	$TRACK	4	Entry to HASP Track Allocation Routine.
40	64	$PURGER	4	Entry to HASP Track Purge Routine.
44	68	$EXCP	4	Entry to HASP Input/Output Supervisor.
48	72	$EXTPOPE	4	Entry to HASP RTAM Open Routine.
4C	76	$EXTPGET	4	Entry to HASP RTAM Get Routine.
50	80	$EXTPPUT	4	Entry to HASP RTAM Put Routine.
54	84	$EXTPOPE	4	Entry to HASP RTAM Close Routine.
58	88	$RESTORE	4	Entry to HASP RTAM Restore Routine.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
5C	92	$ODEL	4	Entry to HASP Overlay $DELETE Routine.
60	96	$ORET	4	Entry to HASP Overlay $RETURN Routine.
64	100	$OLINK	4	Entry to HASP Overlay $LINK Routine.
68	104	$OXCTL	4	Entry to HASP Overlay $XCTL Routine.
6C	108	$OLOAD	4	Entry to HASP Overlay $LOAD Routine.
70	112	$WTO	4	Entry to HASP Write-to-Operator Routine.
74	116	$FREEMSG	4	Entry to HASP Console Message Buffer Free Routine.
78	120	$STIMER	4	Entry to HASP Set Interval Timer Routine.
7C	124	$TTIMER	4	Entry to HASP Test Interval Timer Routine.
80	128	$IOERROR	4	Entry to HASP Input/Output Error Logging Routine.
84	132	$ERROR	4	Entry to HASP Catastrophic Error Routine.
88	136	$DISTERR	4	Entry to HASP Disastrous Error Routine.
8C	140	$SYSTYPE	1	System Type --

Hex. Value	Meaning
10	MVT
14	MPS
20	MFT

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
8D	141	$OPTSTAT	1	Initialization Options --

Bit	Name	Meaning
0	$OPTFMT	FORMAT.
1	$OPTCOLD	COLD.
2	$OPTREQ	REQ.
3	$OPTREP	REP.
4	$OPTLIST	LIST.
5	$OPTRACE	TRACE.
6-7		Reserved for Future Use.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	Field Name	Bytes	Field Description
8E	142	$STATUS	1	HASP Status --

Bit	Name	Meaning
0	$RDRPEND	O/S Reader is Pending.
1	$ALMSGSW	ALL AVAILABLE FUNCTIONS COMPLETE Message has been Issued.
2	$DRAINED	System has been $DRAINed.
3	$CKPTACT	Checkpoint is in Progress.
4	$JITCKPT	Job Information Table (JIT) is to be Checkpointed.
5	$SYSEXIT	HASP System is in termination process.
6-7		Reserved for Future Use.

8F	143		1	Reserved for Future Use.
90	144	$HASPECF	1	Master Event Control Field --

Bit	Name	Meaning
0	$EWFPOST	A PCE has been $POSTed.
1	$EWFBUF	A Buffer has been Released.
2	$EWFTRAK	A Direct-Access Track has been Released.
3	$EWFJOB	A Job Queue Element has Changed Status.
4	$EWFUNIT	A HASP Unit has been Released.
5	$EWFCKPT	A HASP Checkpoint has Completed.
6	$EWFCMB	A Console Message Buffer has been Released.
7	$EWF8	Reserved for Future Use.

91	145	MHASPECF	1	Remote Job Entry Line Manager Event Control Field --

Bit	Name	Meaning
0-2		Reserved - Must be Zero.
3	$EWFJOB	A Job Queue Element has Changed Status.
4-7		Reserved - Must be Zero.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
92	146	$XEQACT	1	Count of Jobs in O/S Execution Phase.
93	147	$ACTIVE	1	Count of Active Processors.
94	148	$ENBALL	1	$ENABLE ALL Mask (X'FF').
95	149	$DISALL	1	$DISABLE ALL Mask (X'00').
96	150	$DISINT	1	$DISABLE INT Mask (X'FE').
97	151		1	Reserved for Future Use.
98	152	$PSRDRCT	1	Count of Pseudo 2540 Readers.
99	153	$PSPRFCT	1	Count of Pseudo 1443 Printers.
9A	154	$PSPUFCT	1	Count of Pseudo 1442 Punches.
9B	155		1	Reserved for Future Use.
9C	156	$EXCPCT	2	Count of Active I/O Operations.
9E	158	$COMMCT	2	Number of Console Message Buffers which are not Queued for the HASP Command Processor.
A0	160	$CKPTRAK	2	Checkpoint Track.
A2	162		2	Reserved for Future Use.
A4	164	$HASPTCB	4	Address of HASP Task Control Block.
A8	168	$PCEORG	4	Address of First HASP Processor Control Element.
AC	172	$BUFPOOL	4	Address of First Available HASP Buffer.
B0	176	$TPBPOOL	4	Address of First Available HASP RJE Buffer.
B4	180	$DCTPOOL	4	Address of First HASP Device Control Table.
B8	184	$JITABLE	4	Address of HASP Job Information Table.
BC	188	$CYLMAP	4	Address of First HASP Track Allocation Map.
C0	192	$TEDADDR	4	Address of First Track Extent Data Table.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
C4	196	$DCBLIST	4	Address of HASP Direct Access DCB.
C8	200	$FREEQUE	4	Address of First Free HASP Console Message Buffer.
CC	204	$BUSYQUE	4	Address of First Console Message Buffer which is Queued for I/O.
D0	208	$LOGQUE	4	Address of First Console Message Buffer which is Queued for the Log Processor.
D4	212	$COMMQUE	4	Address of First Console Message Buffer which is Queued for the Command Processor.
D8	216	$PRCHKPT	4	Address of HASP Print Checkpoint Table.
DC	220	$SVCRELT	4	Address of MFT SVC Relocation Table.
E0	224	$SVCTABF	4	Address of MFT SVC Table.
E4	228	$SVCTABV	4	Address of MVT SVC Table.
E8	232	$IOSENT	4	Address of Entry to O/S Input/Output Supervisor.
EC	236	$ATTNENT	4	Address of Entry to IOS Attention Appendage.
F0	240	$XSMFEMT	4	Address of Entry to SMF EXCP Counting Routine.
F4	244	$SVRSET	4	Address of Entry to HASP SVC Reset Routine.
F8	248	$WAITENT	4	Address of Entry to IGC001 (WAIT).
FC	252	$LINKENT	4	Address of Entry to IGC006 (LINK).
100	256	$XCTLENT	4	Address of Entry to IGC007 (XCTL).
104	260	$TIMENT	4	Address of Entry to IGC011 (TIME).
108	264	$SVCIOS	4	Address of EXCP SVC Table Entry.
10C	268	$SVCLINK	4	Address of LINK SVC Table Entry.

Figure 8.1.1 -- HASP COMMUNICATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
110	272	$SVCWTO	4	WTO/WTOR SVC Table Entry.
114	276	$SVCWTL	4	WTL SVC Table Entry.
118	280	$ATTNSAV	12	Attention Appendage Save Area.
124	292	$JOBQPTR	4	Address of HASP Job Queue.
128	296	$JQFREE	4	Beginning of Free Job Queue Element Chain.
12C	300	$JQENT	4	Beginning of Active Job Queue Element Chain.
130	304	$XEQTOTL	4	Cumulative Estimated Execution Time.
134	308	$PRTTOTL	4	Cumulative Lines to be Printed.
138	312	$PUNTOTL	4	Cumulative Cards to be Punched.
13C	316	$JOBNO	2	HASP Job Number.
13E	318	$MSGRPNO	2	Last Remote Console Message Queueing Track.
140	320	$DACKPT		Variable Length Direct Access Checkpoint Area.

Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT

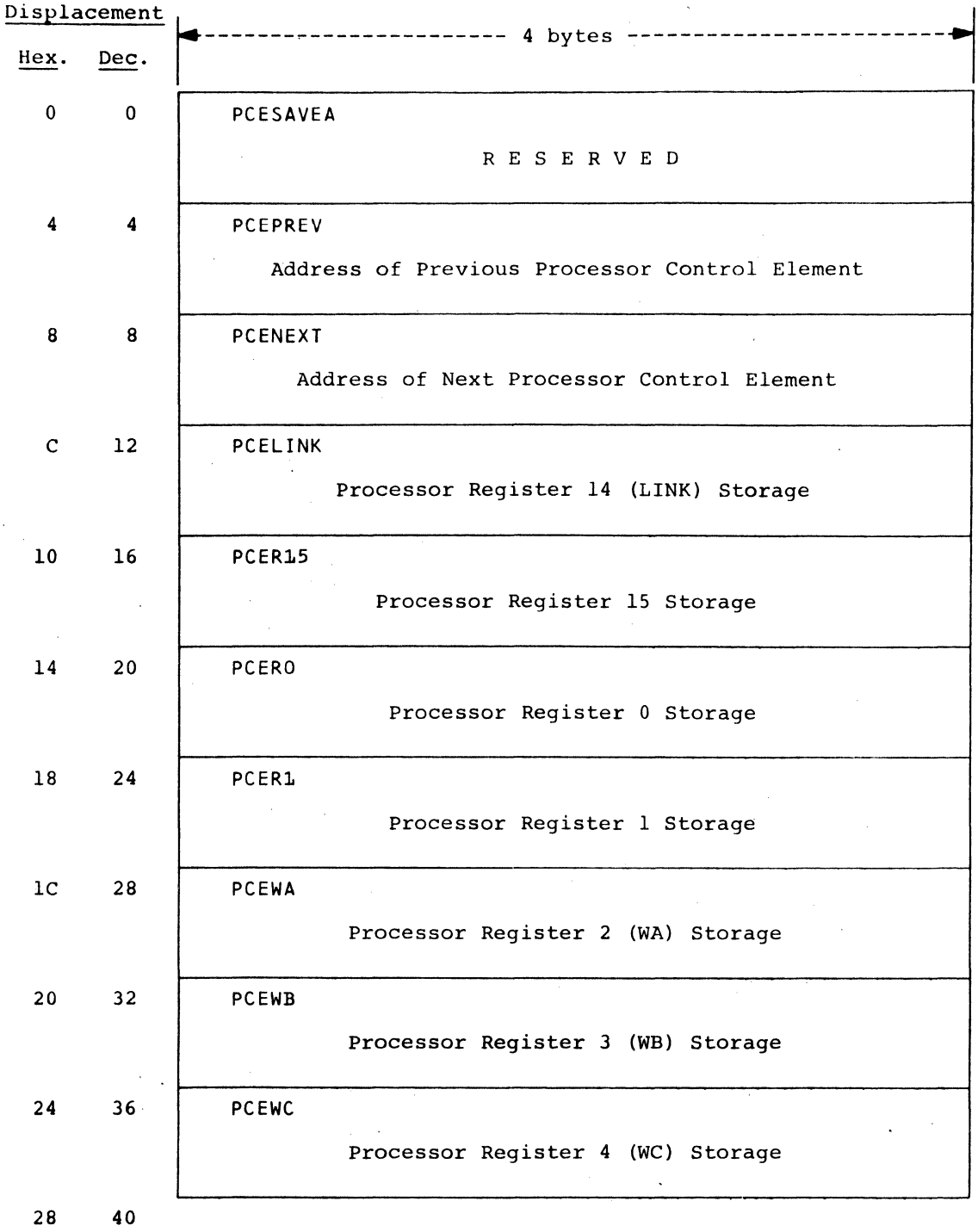
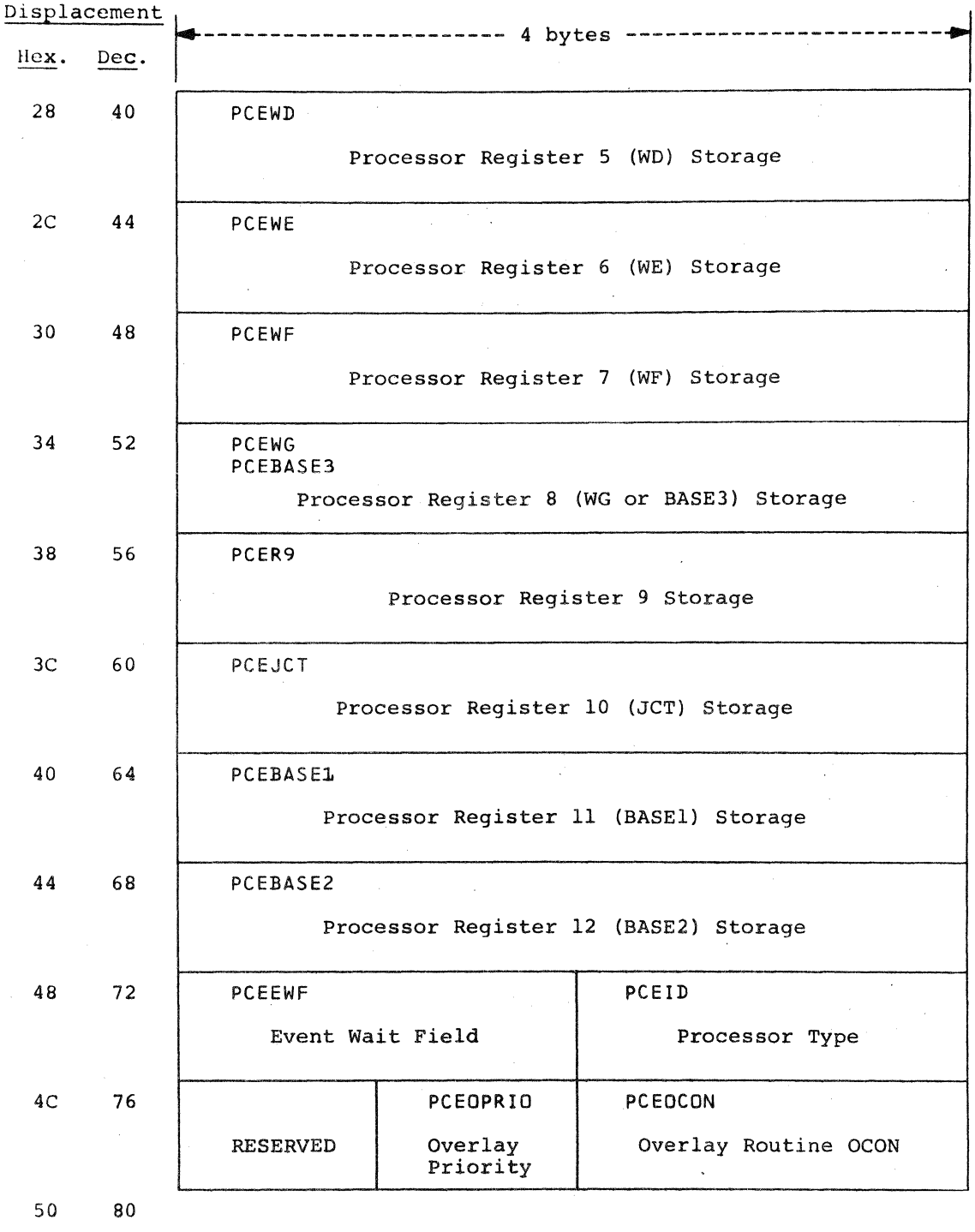


Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT (CONTINUED)



Processor Control Element Format -- Page 8.2-2

Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT (CONTINUED)

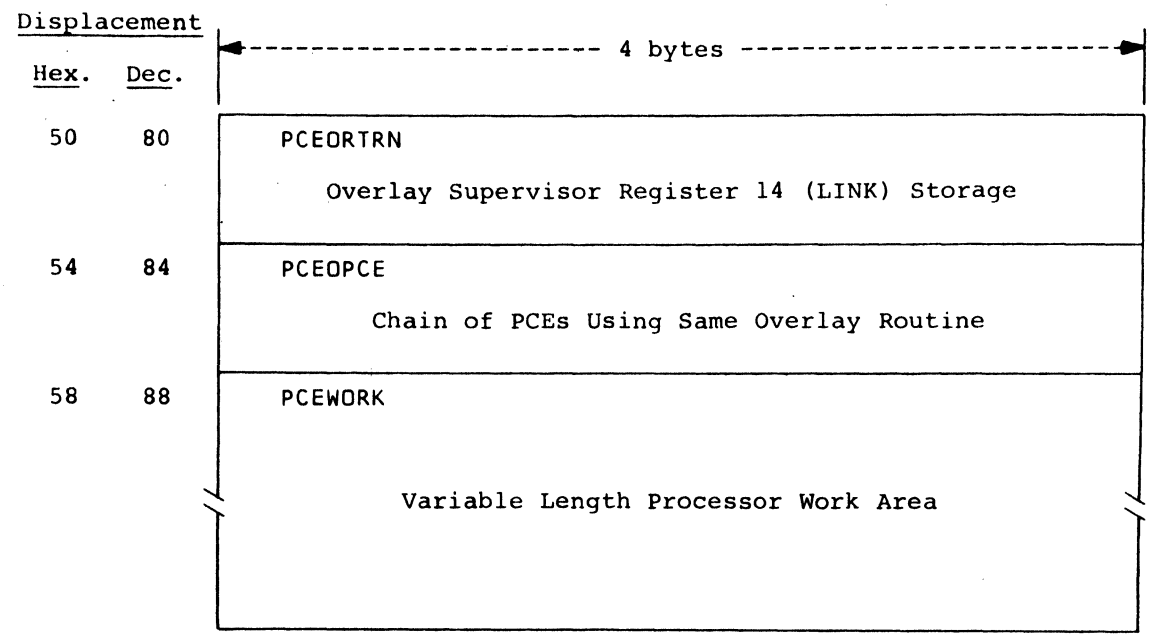


Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	PCESAVEA	4	Reserved.
4	4	PCEPREV	4	Address of Previous Processor Control Element.
8	8	PCENEXT	4	Address of Next Processor Control Element.
C	12	PCELINK	4	Processor Register 14 (LINK) Storage.
10	16	PCER15	4	Processor Register 15 Storage.
14	20	PCER0	4	Processor Register 0 Storage.
18	24	PCER1	4	Processor Register 1 Storage.
1C	28	PCEWA	4	Processor Register 2 (WA) Storage.
20	32	PCEWB	4	Processor Register 3 (WB) Storage.
24	36	PCEWC	4	Processor Register 4 (WC) Storage.
28	40	PCEWD	4	Processor Register 5 (WD) Storage.
2C	44	PCEWE	4	Processor Register 6 (WE) Storage.
30	48	PCEWF	4	Processor Register 7 (WF) Storage.
34	52	PCEWG PCEBASE3	4	Processor Register 8 (WG or BASE3) Storage.
38	56	PCER9	4	Processor Register 9 Storage.
3C	60	PCEJCT	4	Processor Register 10 (JCT) Storage.
40	64	PCEBASE1	4	Processor Register 11 (BASE1) Storage.
44	68	PCEBASE2	4	Processor Register 12 (BASE2) Storage.

Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
48	72	PCEEWF	2	Event Wait Field --

	Hex. Value	Name	Meaning
Byte 1	80	$EWFPOST	Reserved.
	40	$EWFBUF	Waiting for a Buffer.
	20	$EWFTRAK	Waiting for HASP Direct-Access Space.
	10	$EWFJOB	Waiting for a Job.
	08	$EWFUNIT	Waiting for a Unit.
	04	$EWFCKPT	Waiting for the completion of a HASP Checkpoint.
	02	$EWFCMB	Waiting for a Console Message Buffer.
	01	$EWF8	Reserved for Future Use.
Byte 2	80	$EWFOPER	Waiting for Operator Response.
	40	$EWFIO	Waiting for the Completion of I/O.
	20	$EWFWORK	Waiting to be Re-directed.
	10	$EWFHOLD	Waiting for a $S Command.
	08	$EWFDDB	Waiting for a DDT or UCB.
	04	$EWFOLAY	Waiting for an Overlay Area.
	02	$EWF15	Reserved for Future Use.
	01	$EWFOROL	Relinquished Overlay Area.

4A	74	PCEID	2	Processor Type --

	Bit	Name	Meaning
Byte 1	0	PCEPRSID	Print Processor.
	1	PCEPUSID	Punch Processor.
	2-4		Reserved for Future Use.
	5	PCEINRID	Internal Reader Processor.
	6	PCERJEID	Remote Terminal Processor.
	7	PCELCLID	Local Processor.

Figure 8.2.1 -- PROCESSOR CONTROL ELEMENT FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
				Processor Type (continued) --
				Hex. Value / Name / Meaning
			Byte 2	00 PCEASYID ASYNCH Processor.
				01 PCERDRID Input Service Processor.
				03 PCEXEQID Execution Service Processor.
				04 PCETHWID Execution Thaw Processor.
				05 PCEXZMID Execution Task Monitor.
				07 PCEPRTID Print Processor.
				08 PCEPUNID Punch Processor.
				09 PCEPRGID Purge Processor.
				0A PCECONID Console Processor.
				0B PCEMLMID Line Manager Processor.
				0C PCETIMID Timer Processor.
				0D PCECKPID Checkpoint Processor.
				0E PCEGPRID Priority Aging Processor.
				0F PCEOROID Overlay Roll Processor.
4C	76		1	Reserved for Future Use.
4D	77	PCEOPRIO	1	Priority of Current Overlay Routine.
4E	78	PCEOCON	2	Overlay Constant (OCON) of Current Overlay Routine.
50	80	PCEORTRN	4	Overlay Supervisor Register 14 (LINK) Storage.
54	84	PCEOPCE	4	Chain of PCEs Using Same Overlay Routine.
58	88	PCEWORK		Variable Length Processor Work Area.

Figure 8.3.1 -- BUFFER FORMAT

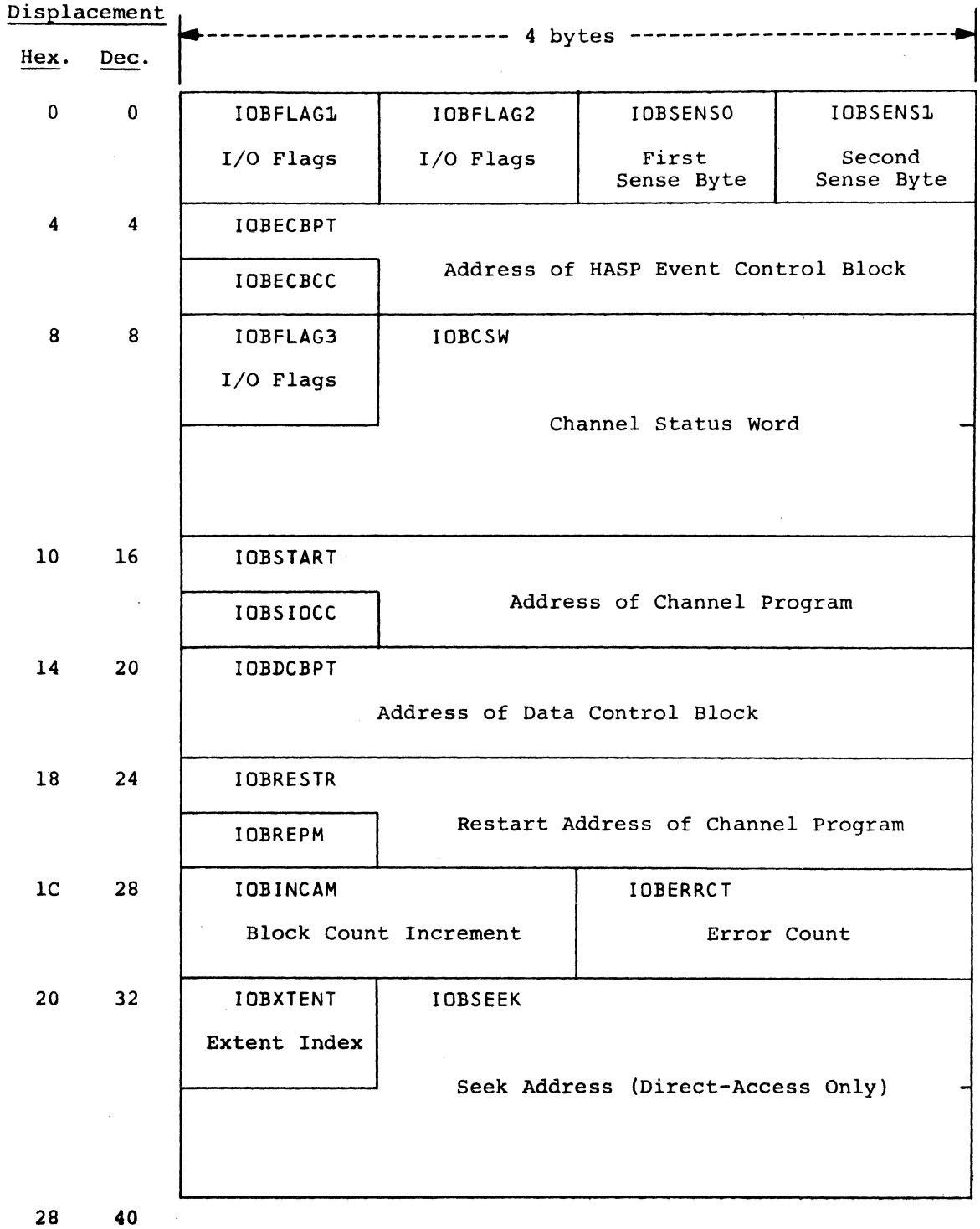


Figure 8.3.1 -- BUFFER FORMAT (CONTINUED)

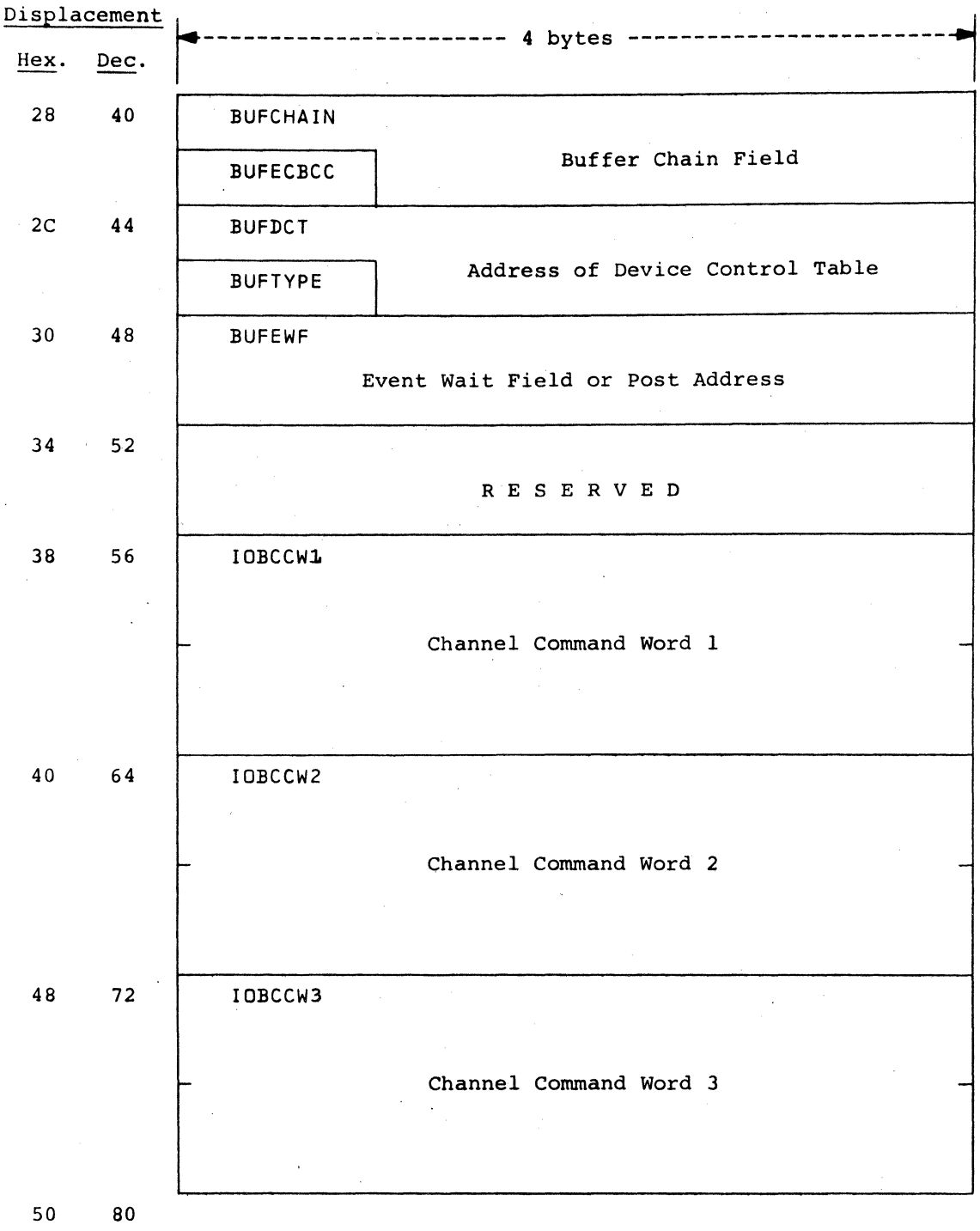


Figure 8.3.1 -- BUFFER FORMAT (CONTINUED)

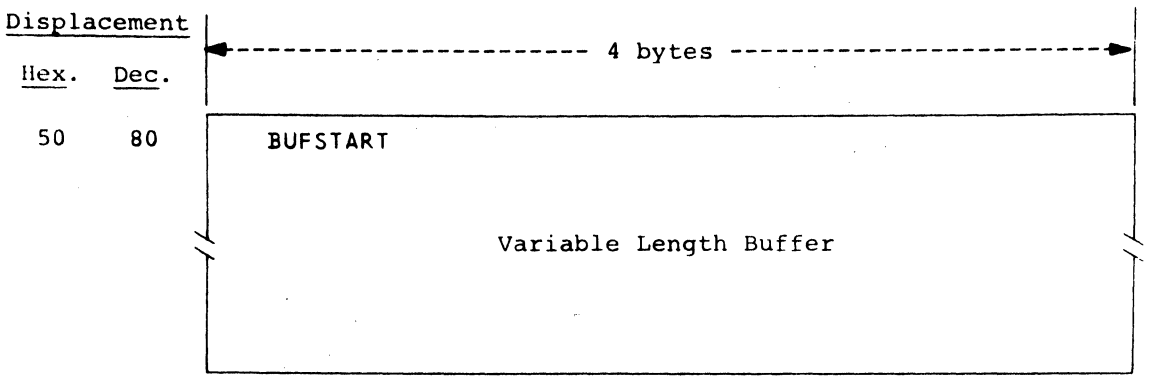


Figure 8.3.1 -- BUFFER FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	IOBFLAG1	1	Standard OS/360 IOB Flag Byte.
1	1	IOBFLAG2	1	Standard OS/360 IOB Flag Byte.
2	2	IOBSENS0	1	First Sense Byte (Device Dependent).
3	3	IOBSENS1	1	Second Sense Byte (Device Dependent).
4	4	IOBECBCC	1	Completion Code for I/O Event.
4	4	IOBECBPT	4	Address of HASP Event Control Block: $HASPECB.
8	8	IOBFLAG3	1	I/O Supervisor Error Routine Flag Byte (Device Dependent).
9	9	IOBCSW	7	Low-Order Seven Bytes of the Last CSW that Reflects the Status of the Last Request.
10	16	IOBSIOCC	1	Condition Code Returned after Execution of SIO Instruction for Last Request.
10	16	IOBSTART	4	Address of Channel Program to be Executed.
14	20	IOBDCBPT	4	Address of Data Control Block Associated with this IOB.
18	24	IOBREPM	1	Operation Code Used by I/O Supervisor Error Routines for Repositioning Procedures.
18	24	IOBRESTR	4	Restart Address of Channel Program Used by I/O Supervisor Error Routines During Error Correction.
1C	28	IOBINCAM	2	Value used to Increment Block Count Field in DCB for Magnetic Tape.
1E	30	IOBERRCT	2	Used by I/O Supervisor Error Routines to Count Temporary Errors during Retry.
20	32	IOBXTENT	1	The Number of the DEB Extent to be Used for this Request.

Figure 8.3.1 -- BUFFER FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
21	33	IOBSEEK	7	Seek Address Required for this I/O Request (Direct-Access Only).
28	40	BUFECBCC	1	Completion Code for I/O Event --

Hex. Value	Meaning
00	The I/O Event has not Completed.
7F	The I/O Event has Completed Successfully.
other	The I/O Event has Completed Unsuccessfully.

28	40	BUFCHAIN	4	Buffer Chain Field.
2C	44	BUFTYPE	1	Buffer Type --

Hex. Value	Name	Meaning
00	HASPBUF	HASP Buffer

2C	44	BUFDCT	4	Address of Device Control Table Associated with this I/O Request.
30	48	BUFEWF	4	Event Wait Field or Post Address.
34	52		4	Reserved for Future Use.
38	56	IOBCCW1	8	Channel Command Word 1.
40	64	IOBCCW2	8	Channel Command Word 2.
48	72	IOBCCW3	8	Channel Command Word 3.
50	80	BUFSTART		Variable Length Buffer.

Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT

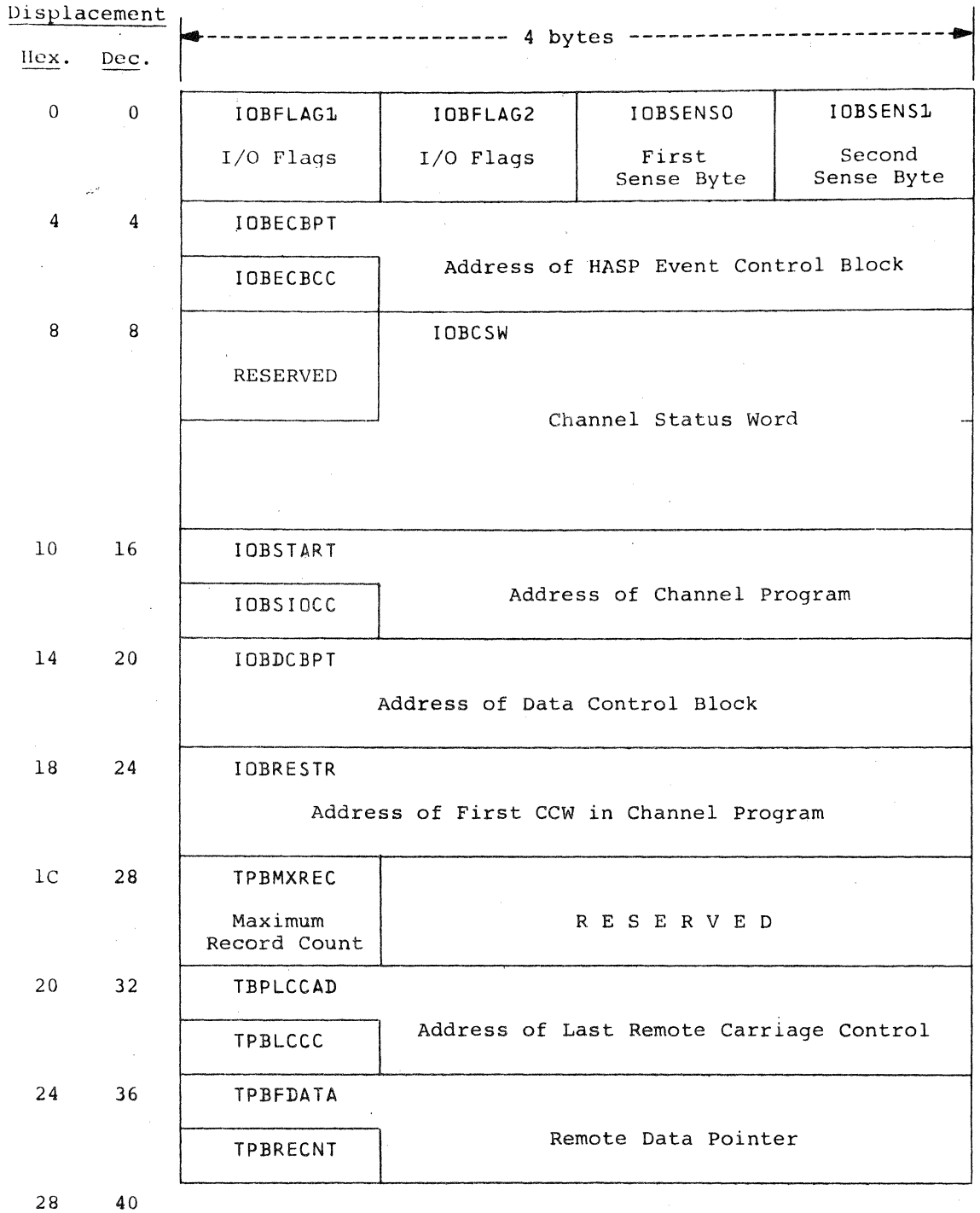


Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

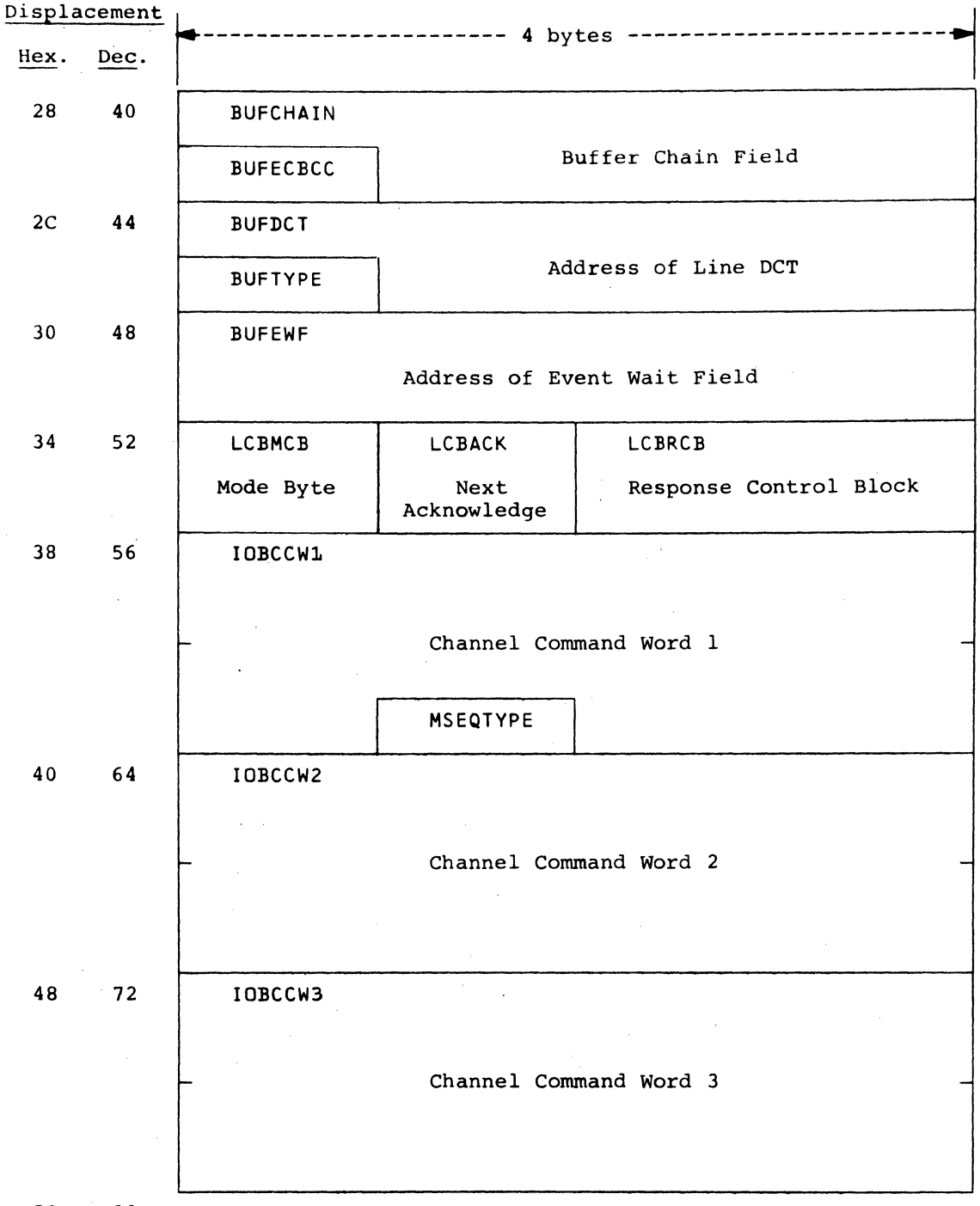


Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

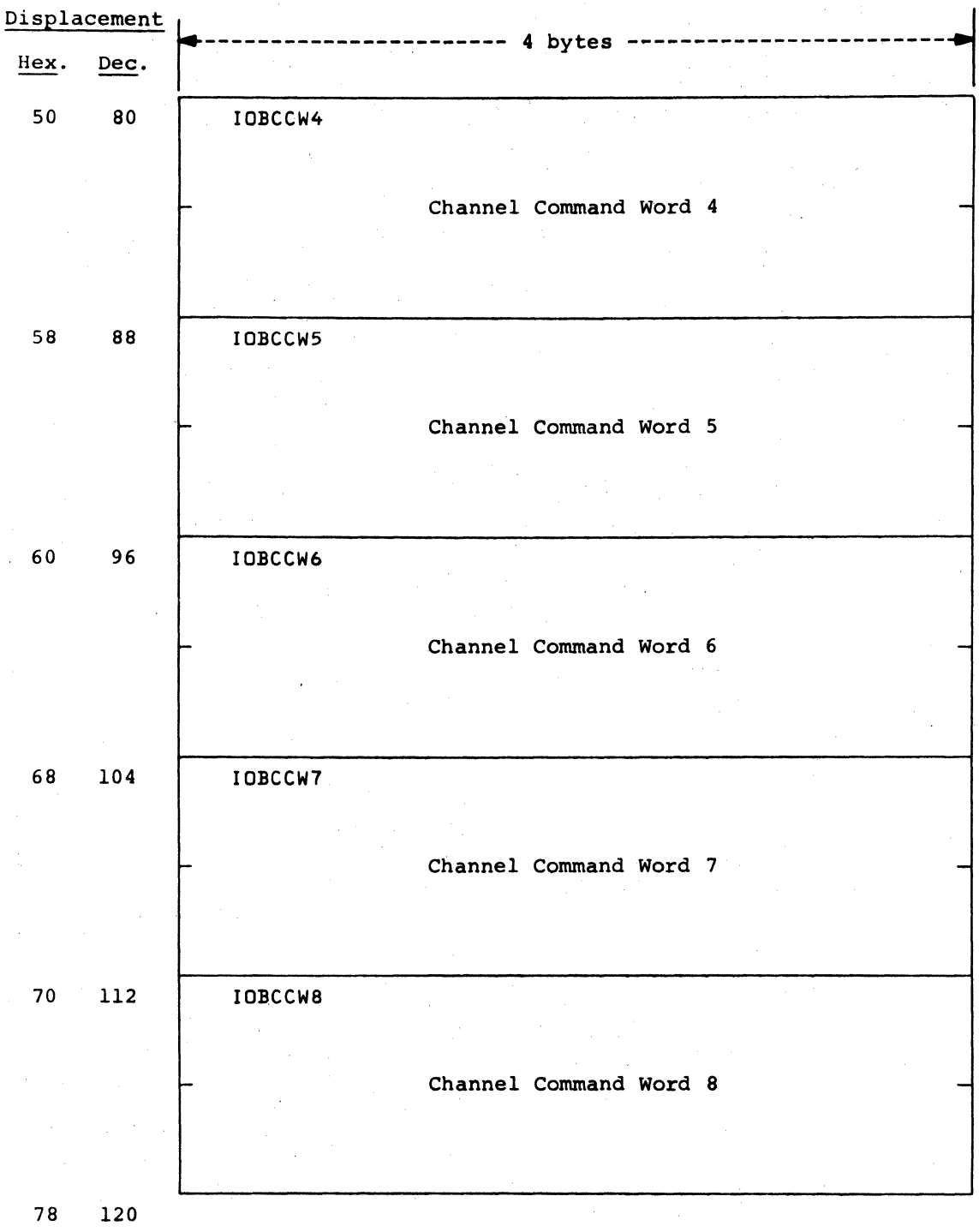


Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

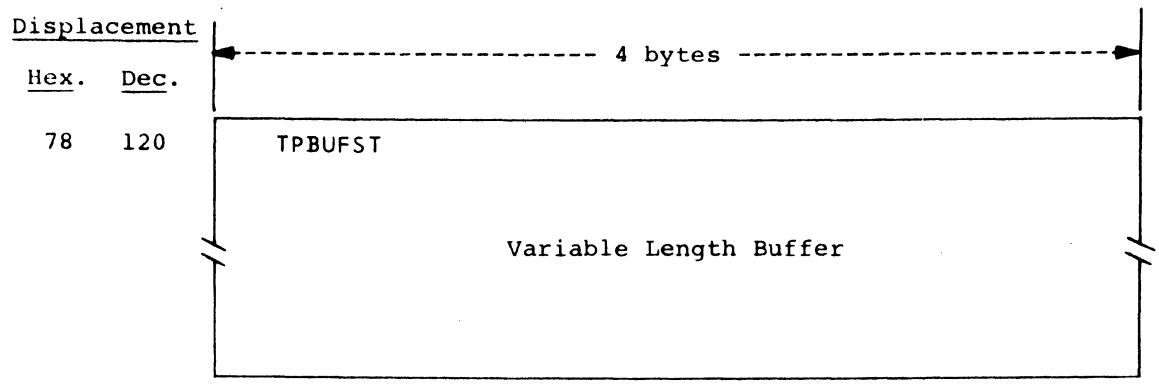


Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	IOBFLAG1	1	Standard OS/360 IOB Flag Byte.
1	1	IOBFLAG2	1	Standard OS/360 IOB Flag Byte.
2	2	IOBSENS0	1	First Sense Byte (Device Dependent).
3	3	IOBSENS1	1	Second Sense Byte (Device Dependent).
4	4	IOBECBCC	1	Completion Code for I/O Event.
4	4	IOBECBPT	4	Address of HASP Event Control Block: $HASPECB.
8	8		1	Reserved.
9	9	IOBCSW	7	Low Order Seven Bytes of the Last CSW that Reflects the Status of the Last Request.
10	16	IOBSIOCC	1	Condition Code Returned after Execution of SIO Instruction for Last Request
10	16	IOBSTART	4	Address of Channel Program to be Executed.
14	20	IOBDCBPT	4	Address of Data Control Block Associated with this IOB.
18	24	IOBRESTR	4	Address of Normal Channel Program to be Executed.
1C	28	TPBMXREC	1	Maximum Output Record Count.
1D	29		3	Reserved for Future Use.
20	32	TPBLCCC	1	Last Output Channel Command Operation.
20	32	TPBLCCAD	4	Address of Last Remote Carriage Control.
24	36	TPBRECNT	1	Current Output Record Count.
24	36	TPBFDATA	4	Address of Next Data in Buffer.
28	40	BUFECBCC	1	Completion Code for I/O Event.
28	40	BUFCHAIN	4	Buffer Chain Field.

Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
2C	44	BUFTYPE	1	Buffer Type --

Hex. Value	Name	Meaning
80	TPBUF	Remote Job Entry Buffer

2C	44	BUFDCT	4	Address of Line Device Control Table Associated with this I/O Request.
30	48	BUFEWF	4	Address of Event Wait Field.
34	52	LCBMCB	1	Mode Byte Used to Set SDA Mode.
35	53	LCBACK	1	BSC: Next Acknowledgement Character (Expected or to be Sent). STR: Second Mode Byte.
36	54	LCBRCB	2	BSC: Response Control Block. STR: Unused.
38	56	IOBCCW1	8	Channel Command Word 1.
3D	61	MSEQTYPE	1	Sequence and Command Type --

Bits 0-3 Sequence Type

Bit	Name	Value	Meaning
0	MBSCSEQ	0	STR Sequence.
		1	BSC Sequence.
1	MPREPSEQ	0	Text Sequence.
		1	Prepare Sequence.
2	MWRITSEQ	0	Read Sequence.
		1	Write Sequence.
3	MCPUSEQ	0	Hardware Sequence.
		1	CPU Sequence.

Figure 8.3.2 -- REMOTE JOB ENTRY BUFFER FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
				Sequence and Command Type (continued) --
			Bits 4-7	Command Type

Hex. Value	Name	Meaning
0	MDISCMD	Disable Command.
1	MSETMCMD	Set Mode Command.
2	MENBCMD	Enable Command.
3	MTSYNCMD	Test Synch Command.
4	MREADCMD	Read Text Command.
5	MRRSPCMD	Read Response (Normal).
6	MRREQCMD	Read Response (To ENQ).
7	MPREPCMD	Prepare Command.
8	MWRITCMD	Write Text Command.
9	MWRSPCMD	Write Response Command.
A	MWENQCMD	Send Inquiry Command.
B	MSEOTCMD	Send EOT Command.

Hex.	Dec.			
40	64	IOBCCW2	8	Channel Command Word 2.
48	72	IOBCCW3	8	Channel Command Word 3.
50	80	IOBCCW4	8	Channel Command Word 4.
58	88	IOBCCW5	8	Channel Command Word 5.
60	96	IOBCCW6	8	Channel Command Word 6.
68	104	IOBCCW7	8	Channel Command Word 7.
70	112	IOBCCW8	8	Channel Command Word 8.
78	120	TPBUFST		Variable Length Buffer.

Figure 8.3.3 -- OVERLAY AREA FORMAT

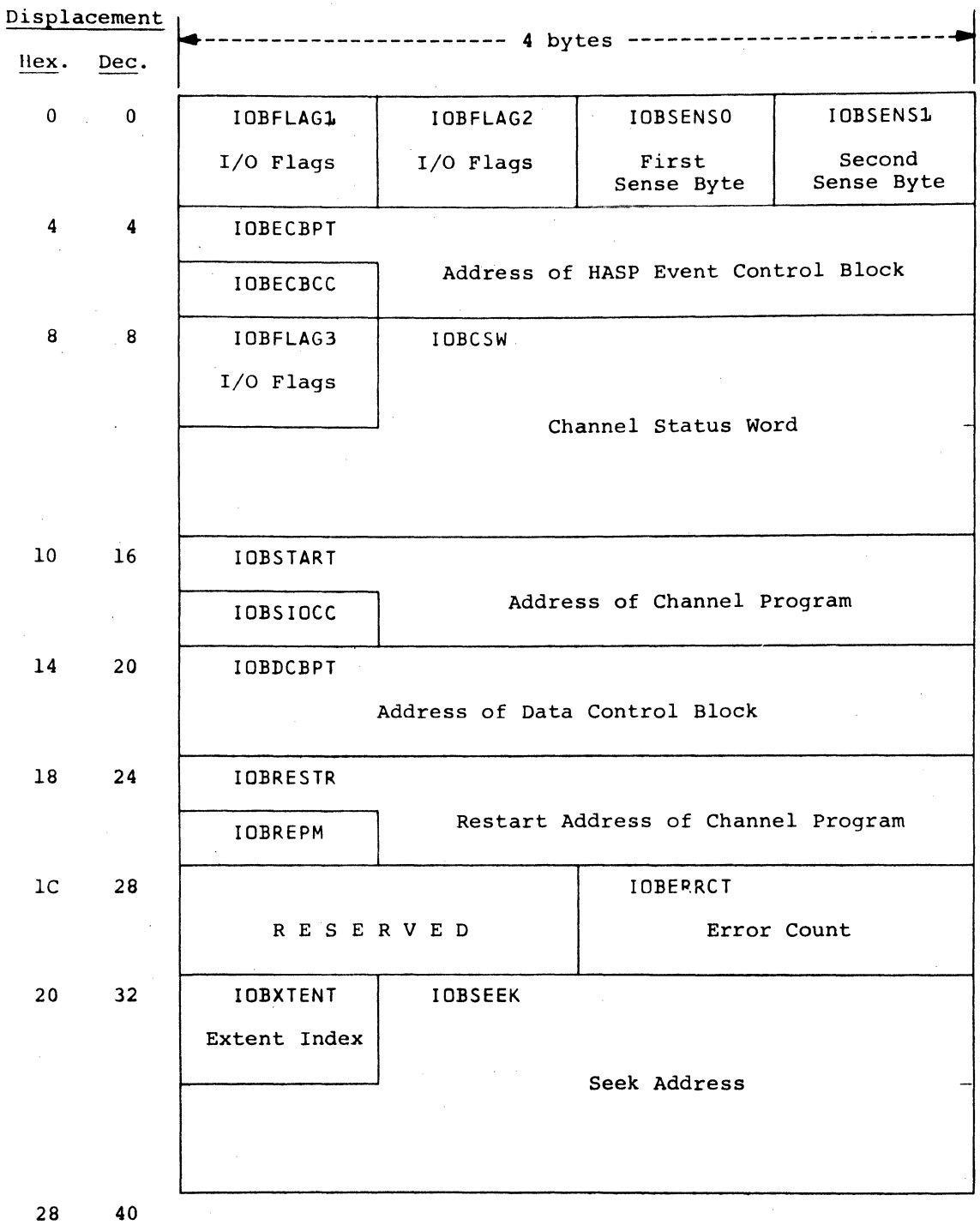


Figure 8.3.3 -- OVERLAY AREA FORMAT (CONTINUED)

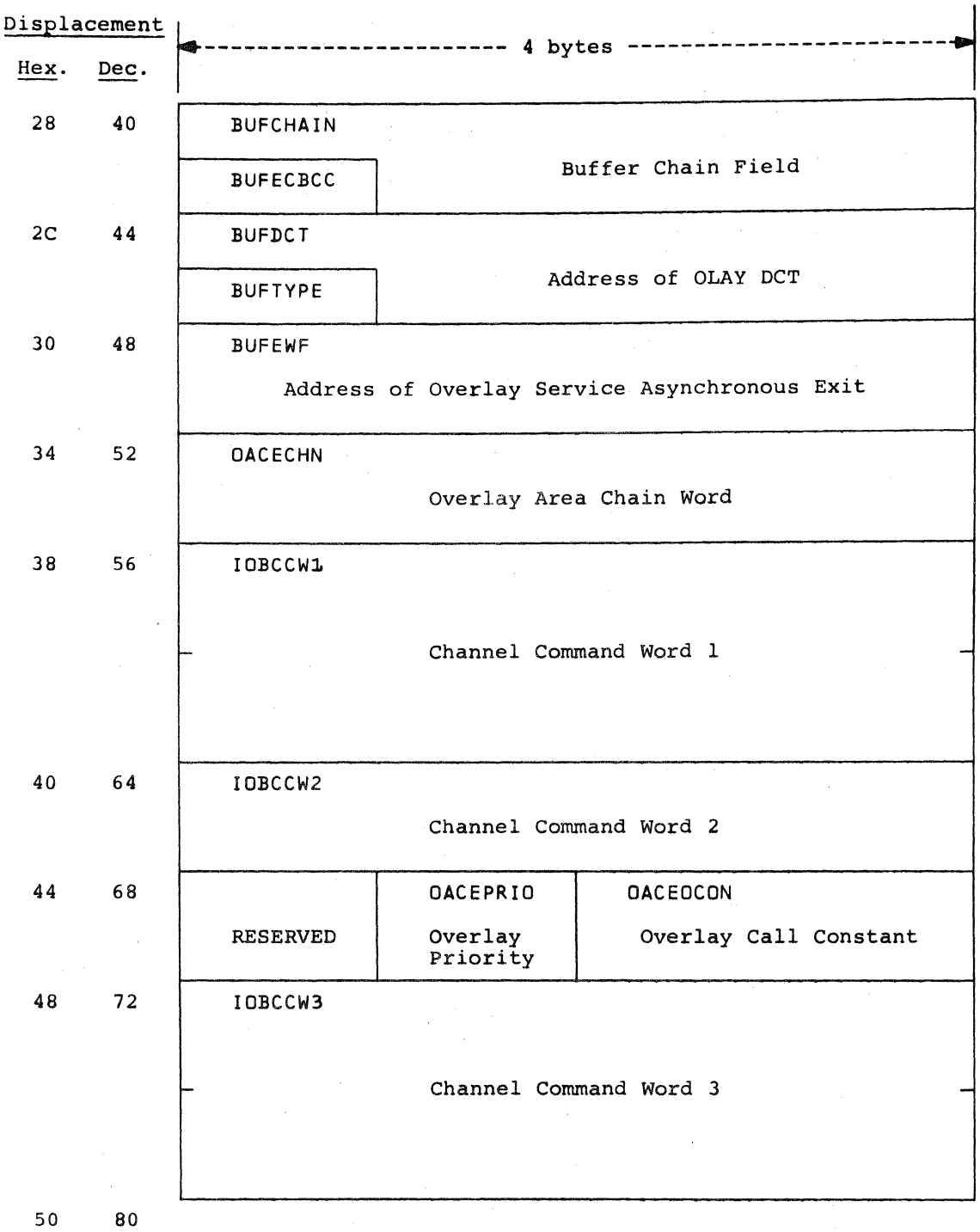


Figure 8.3.3 -- OVERLAY AREA FORMAT (CONTINUED)

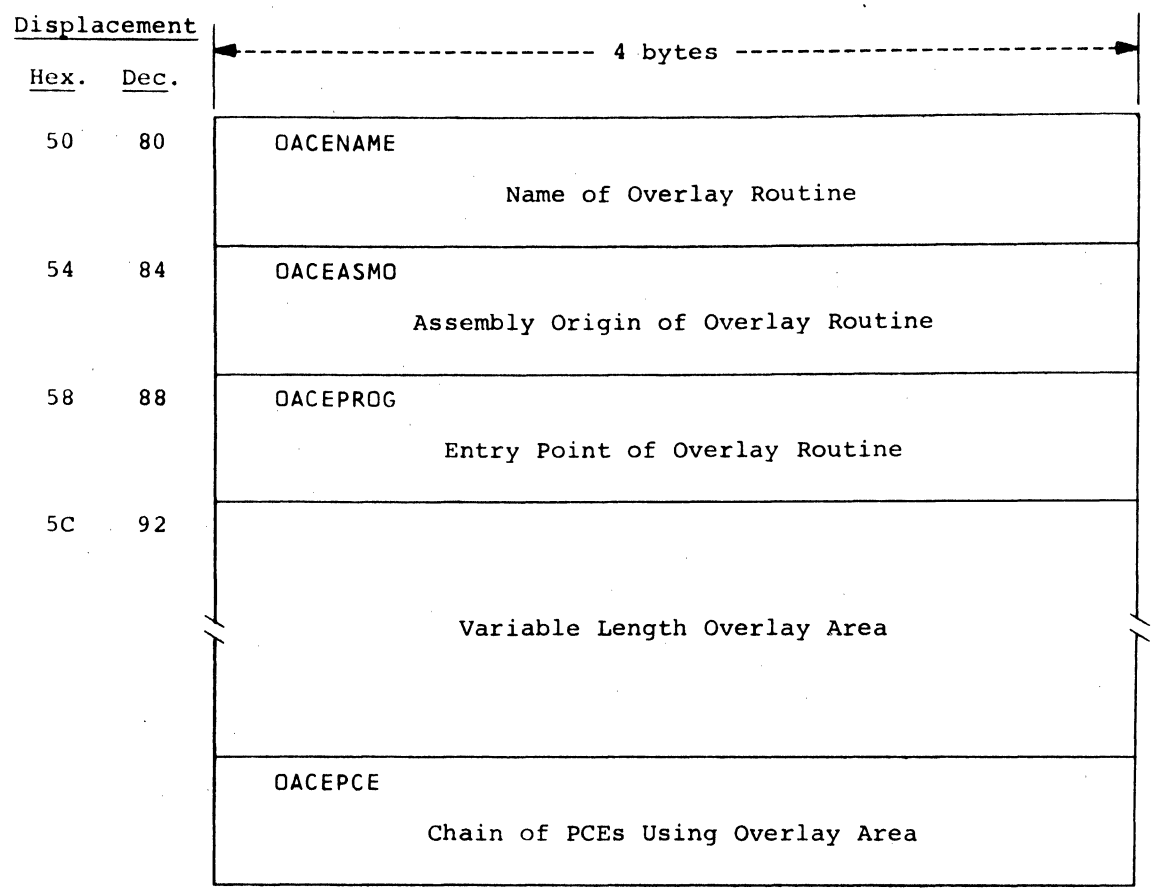


Figure 8.3.3 -- OVERLAY AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	IOBFLAG1	1	Standard OS/360 IOB Flag Byte.
1	1	IOBFLAG2	1	Standard OS/360 IOB Flag Byte.
2	2	IOBSENS0	1	First Sense Byte (Device Dependent).
3	3	IOBSENS1	1	Second Sense Byte (Device Dependent).
4	4	IOBECBCC	1	Completion Code for Overlay Read.
4	4	IOBECBPT	4	Address of HASP Event Control Block: $HASPECB.
8	8	IOBFLAG3	1	I/O Supervisor Error Routine Flag Byte (Device Dependent).
9	9	IOBCSW	7	Low Order Seven Bytes of the Last CSW that Reflects the Status of the Last Read.
10	16	IOBSIOCC	1	Condition Code Returned After the Execution of the SIO Instruction for the Last Read.
10	16	IOBSTART	4	Address of the Channel Program to be Executed.
14	20	IOBDCBPT	4	Address of the Data Control Block Associated with this IOB.
18	24	IOBREPM	1	Operation Code Used by I/O Supervisor Error Routines for Repositioning Procedures.
18	24	IOBRESTR	4	Restart Address of Channel Program Used by I/O Supervisor Error Routines During Error Correction.
1C	28		2	Reserved.
1E	30	IOBERRCT	2	Used by I/O Supervisor Error Routines to Count Temporary Errors During Retry.
20	32	IOBXTENT	1	The Number of the DEB Extent to be Used for this Read.
21	33	IOBSEEK	7	The Seek Address for the Requested Overlay Routine.

Figure 8.3.3 -- OVERLAY AREA FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
28	40	BUFECBCC	1	Completion Code for Overlay Read --

Hex. Value	Meaning
00	The Read has not Completed.
7F	The Read has Completed Successfully.
other	The Read has Completed Unsuccessfully.

28	40	BUFCHAIN	4	Buffer Chain Field.
2C	44	BUFTYPE	1	Buffer Type --

Hex. Value	Name	Meaning
40	OLAYBUF	Overlay Area.

2C	44	BUFDCT	4	Address of Overlay Device Control Table.
30	48	BUFEWF	4	Address of Overlay Service Asynchronous Exit.
34	52	OACECHN	4	Overlay Area Chain Word.
38	56	IOBCCW1	8	Channel Command Word 1.
40	64	IOBCCW2	8	Channel Command Word 2.
44	68		1	Reserved for Future Use.
45	69	OACEPRIO	1	Priority of Current Overlay Routine.
46	70	OACEOCON	2	Overlay Constant (OCON) of Current Overlay Routine.
48	72	IOBCCW3	8	Channel Command Word 3.
50	80	OACENAME	4	Name of Overlay Routine.
54	84	OACEASMO	4	Assembly Origin of Overlay Routine.
58	88	OACEPROG		Entry Point of Overlay Routine.
				Variable Length Overlay Area
		OACEPCE	4	Chain of PCEs Using Overlay Area.

Figure 8.4.1 -- CONSOLE MESSAGE BUFFER FORMAT

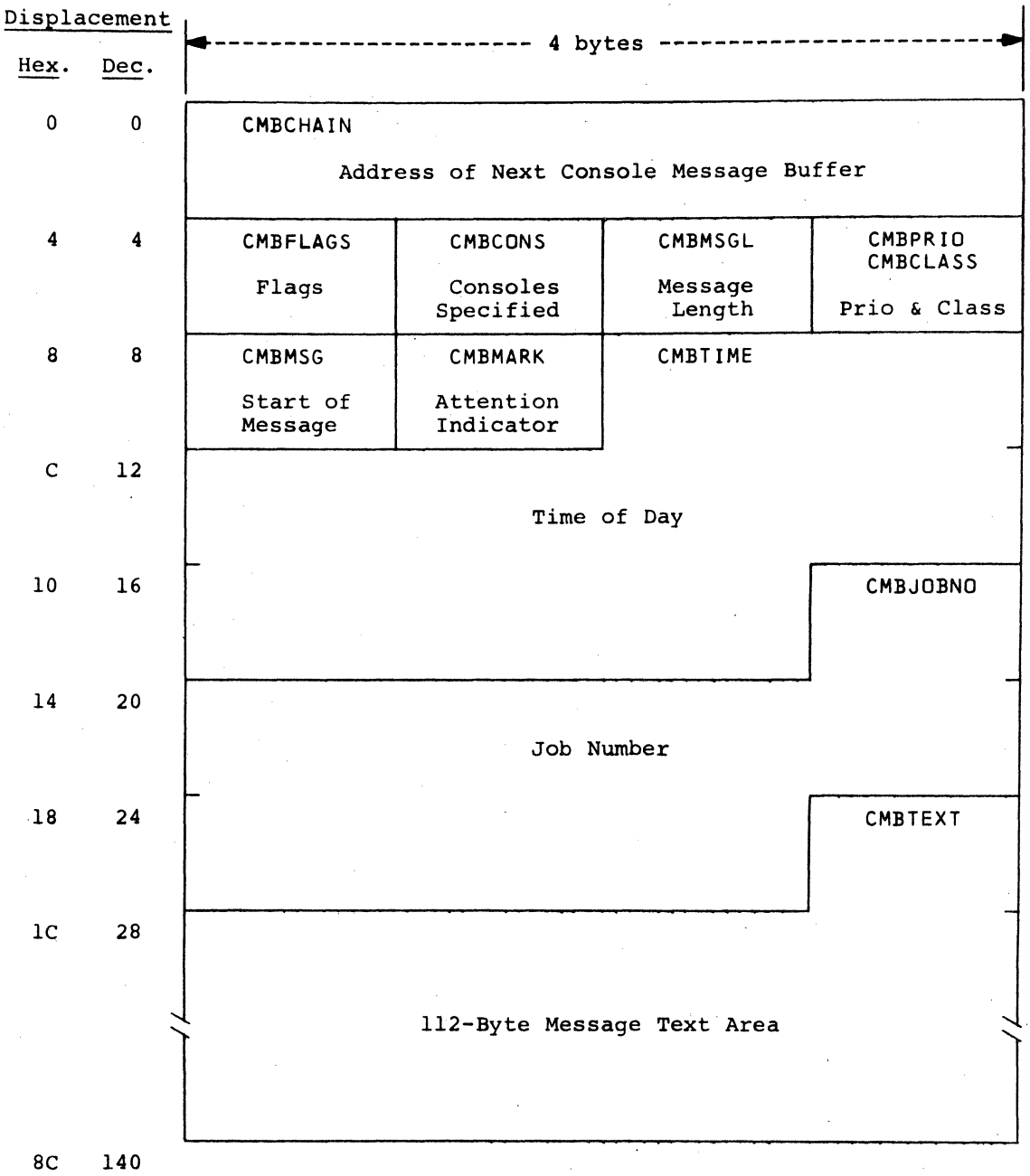


Figure 8.4.1 -- CONSOLE MESSAGE BUFFER FORMAT (CONTINUED)

Displacement Hex.	Dec.	Field Name	Bytes	Field Description
0	0	CMBCHAIN	4	Address of Next Console Message Buffer.
4	4	CMBFLAGS	1	Console Buffer Flags --

Bit	Name	Meaning
0	WCMBFD	CMBCONS Contains Physical Consoles.
1	WCMBFH	Operation Type.
2	WCMBFE	Message for HASP Log Only.
3	WCMBFF	CMBCONS Contains UCMID.
4	WCMBFG	CMBCONS Contains Remote No.
5	WCMBFA	Reserved for
6	WCMBFB	Command
7	WCMBFC	Processor.

5	5	CMBCONS	1	Console Specifications or Remote Number.
6	6	CMBMSGL	1	Message Length.
7	7	CMBCLASS	1	Message Class --

Bits 0-3

Value	Name	Meaning
1	$TRIVIA	Non-Essential Messages.
3	$NORMAL	Normal Messages.
5	$ACTION	Messages Requiring Operator Action.
7	$ALWAYS	Essential Messages.

CMBPRIO — Message Priority --

Bits 4-7

Value	Name	Meaning
1	$LO	Low Priority.
4	$ST	Standard Priority.
7	$HI	High Priority.

8	8	CMBMSG	132	Message Area.
9	9	CMBMARK	1	Asterisk (*) if Message Class is 5 or More.
A	10	CMBTIME	9	Time of Day (HH.MM.SS).
13	19	CMBJOBNO	8	Job Number (If Applicable).
1B	27	CMBTEXT	112	Message Text Area.

Figure 8.5.1 -- DIRECT-ACCESS DEVICE CONTROL TABLE FORMAT

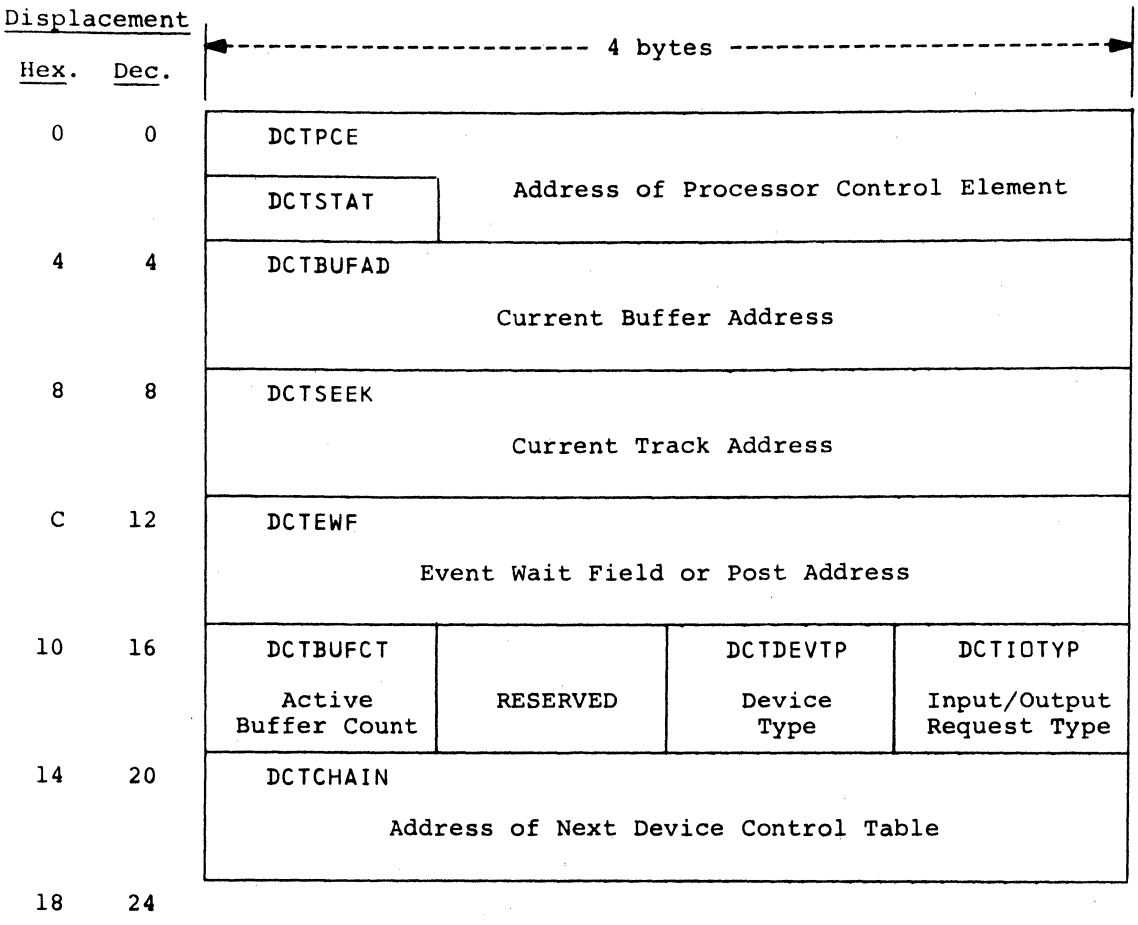


Figure 8.5.1 -- DIRECT-ACCESS DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	DCTSTAT	1	DCT Status --

Bit	Name	Meaning
0	DCTINUSE	DCT is In Use.
1-7		Reserved.

0	0	DCTPCE	4	Address of Processor Control Element.
4	4	DCTBUFAD	4	Address of Current Buffer.
8	8	DCTSEEK	4	Current Track Address.
C	12	DCTEWF	4	Event Wait Field or Post Address.
10	16	DCTBUFCT	1	Number of I/O Requests Outstanding.
11	17		1	Reserved for Future Use.
12	18	DCTDEVTP	1	Device Type --

Hex. Value	Name	Device Type
00	DCTDA	Direct-Access Device.

13	19	DCTIOTYP	1	Input/Output Request Type --

Bit	Name	Meaning
0	DCTREAD	Read Request.
1	DCTWRITE	Write Request.
2-7		Reserved for Future Use.

14	20	DCTCHAIN	4	Address of Next Device Control Table.

Figure 8.5.2 -- OVERLAY DEVICE CONTROL TABLE FORMAT

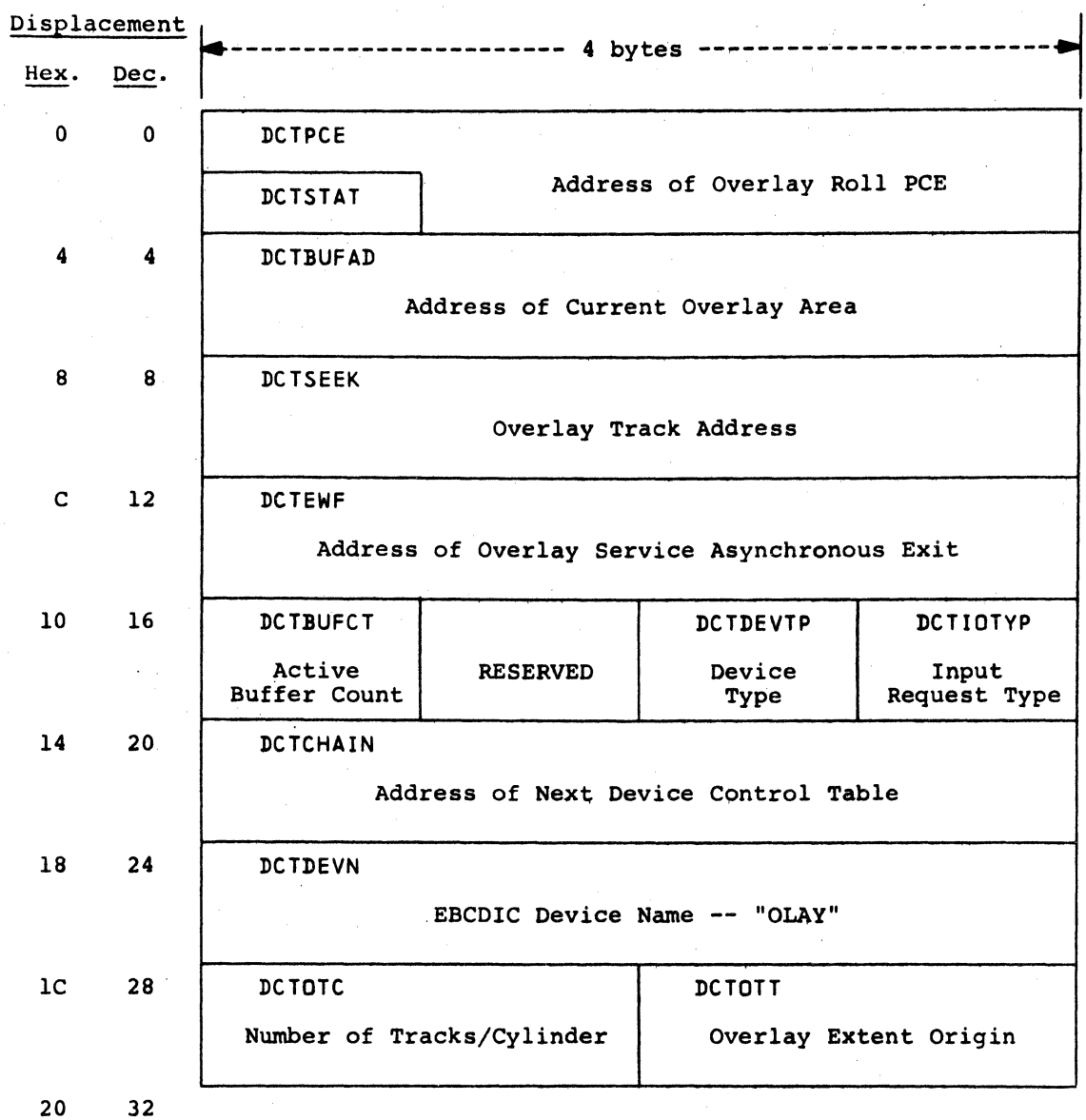


Figure 8.5.2 -- OVERLAY DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	DCTSTAT	1	DCT Status -- Bit 0 Name DCTINUSE Meaning DCT is In Use. Bit 1-7 Meaning Reserved.
0	0	DCTPCE	4	Address of Overlay Roll PCE.
4	4	DCTBUFAD	4	Address of Current Overlay Area.
8	8	DCTSEEK	4	Overlay Track Address.
C	12	DCTEWF	4	Address of Overlay Service Asynchronous Exit.
10	16	DCTBUFCT	1	Number of I/O Requests Outstanding.
11	17		1	Reserved for Future Use.
12	18	DCTDEVTP	1	Device Type -- Hex. Value 00 Name DCTDA Meaning Overlay Data Set Resides on SPOOL Disk. Hex. Value 01 Name DCTOLAY Meaning Overlay Data Set does not Reside on SPOOL Disk.
13	19	DCTIOTYP	1	Input Request Type -- Bit 0 Name DCTREAD Meaning Read Request. Bit 1-7 Meaning Reserved for Future Use.
14	20	DCTCHAIN	4	Address of Next Device Control Table.
18	24	DCTDEVN	4	EBCDIC Device Name -- "OLAY".
1C	28	DCTOTC	2	Number of Tracks per Cylinder on Overlay Direct-Access Device.
1E	30	DCTOTT	2	Overlay Extent Origin.

Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT

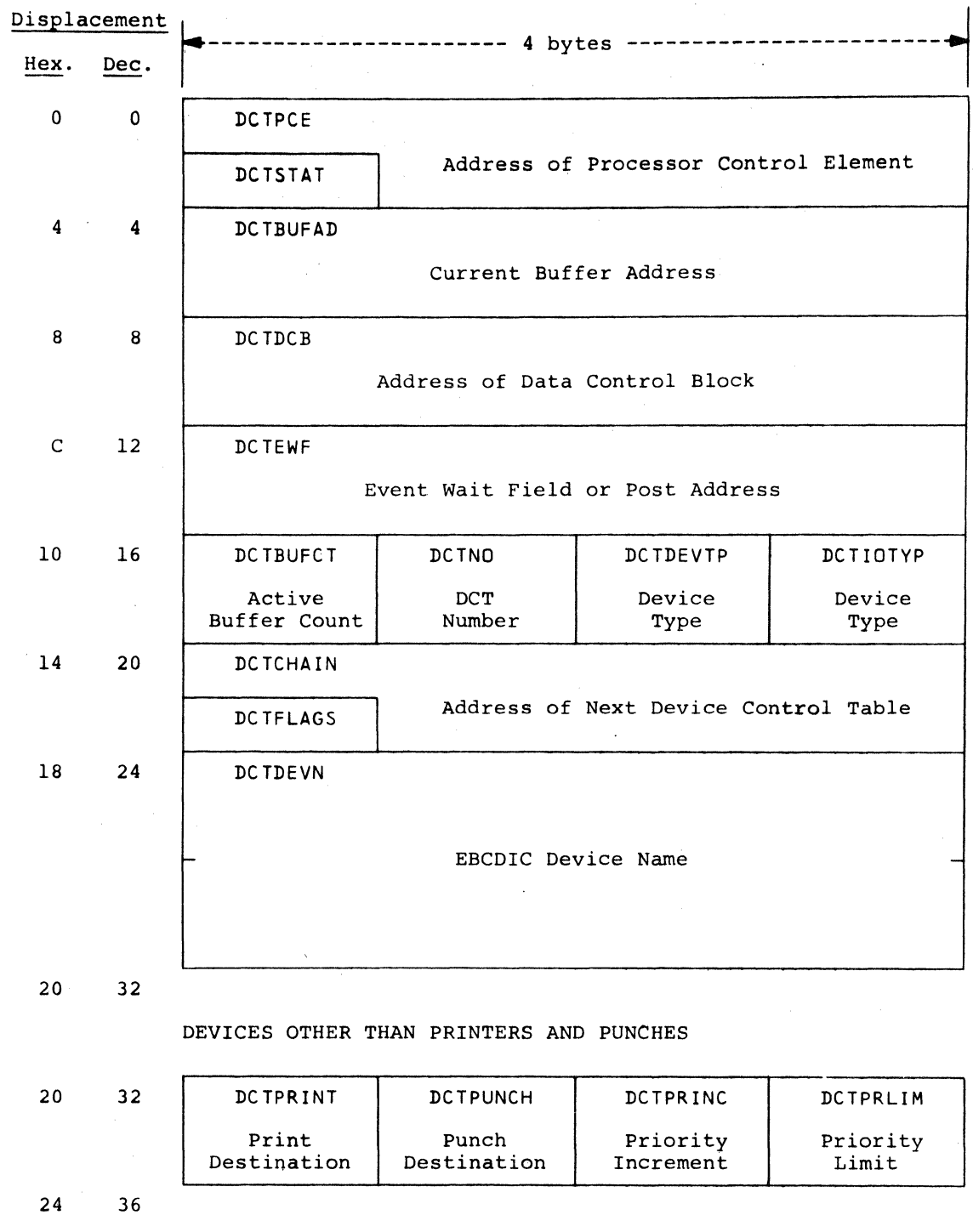


Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT (CONTINUED)

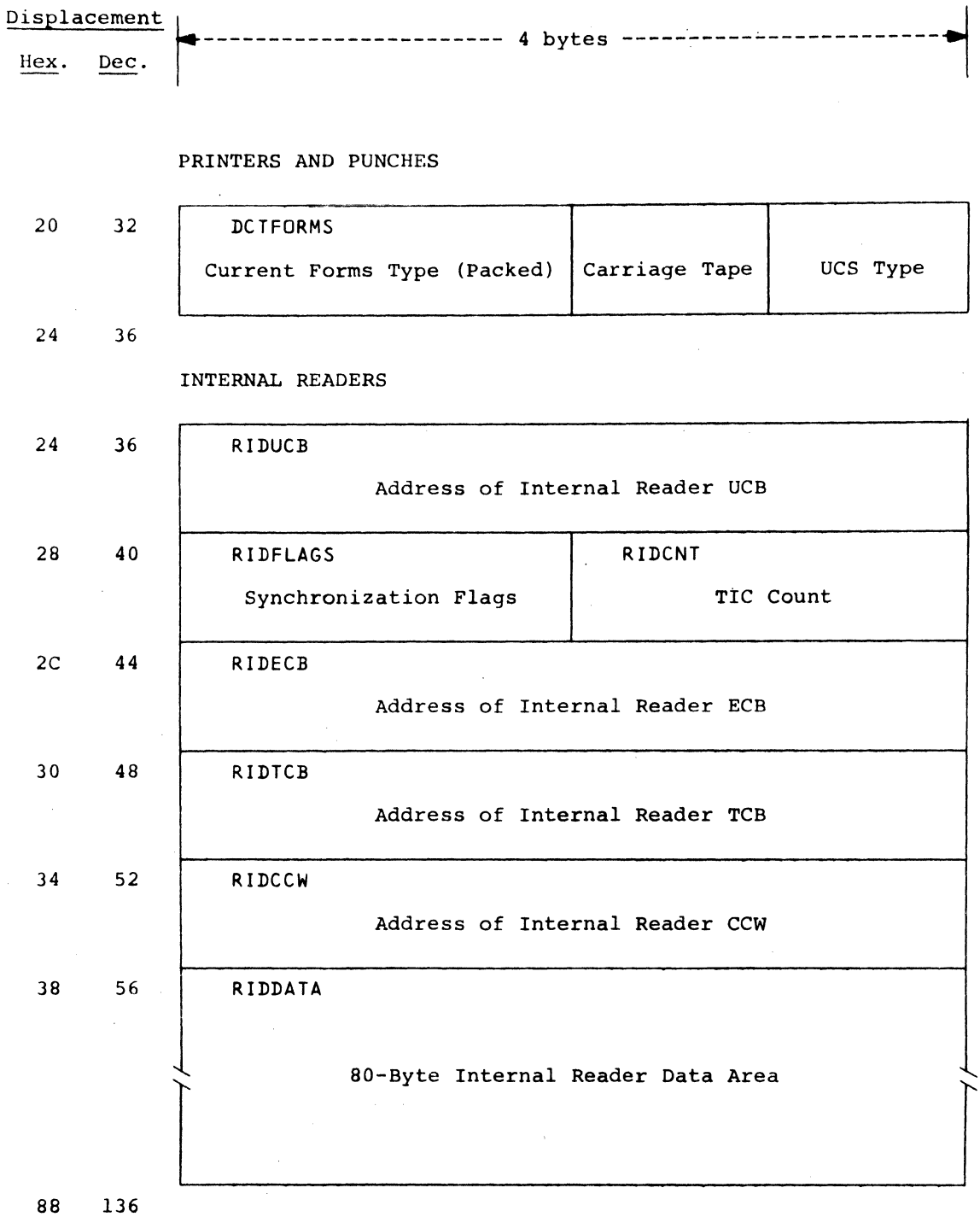


Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT (CONTINUED)

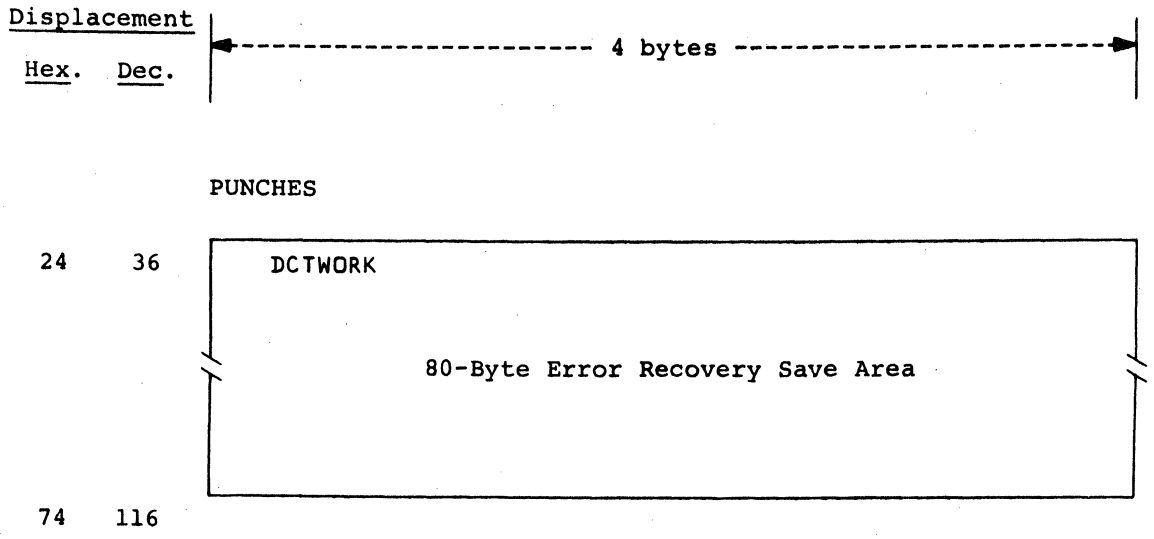


Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	DCTSTAT	1	DCT Status --

Bit	Name	Meaning
0	DCTINUSE	DCT is In Use.
1	DCTDRAIN	DCT is Drained.
2	DCTHOLD	DCT is Held.
3-7		Reserved.

0	0	DCTPCE	4	Address of Processor Control Element.
4	4	DCTBUFAD	4	Current Buffer Address.
8	8	DCTDCB	4	Address of Data Control Block for this Unit.
C	12	DCTEWF	4	Event Wait Field or Post Address.
10	16	DCTBUFCT	1	Number of I/O Requests Outstanding.
11	17	DCTNO	1	Device Number.
12	18	DCTDEVTP	1	Device Type --

Hex. Value	Name	Device Type
10	DCTRDR	Card Reader.
11	DCTTPE	Input Tape.
14	DCTINR	Internal Reader.
20	DCTPRT	Printer.
30	DCTPUN	Punch.
40	DCTCON	Console.

13	19	DCTIOTYP	1	Device Type and Console Restrictions --

Bit	Name	Meaning
0-1		Reserved.
2	DCT1053	1053 Console.
3	DCT2260	2260 Console.
4	DCTREJRM	Reserved.
5	DCTREJJB	Job Command Restriction.
6	DCTREJDV	Device Command Restriction.
7	DCTREJSY	System Command Restriction.

Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
14	20	DCTFLAGS	1	Operator Command Flags --

Bit	Name	Command
0	DCTSTOP	$Z ($STOP)
1	DCTDELET	$C ($DELETE)
2	DCTRSTRT	$E ($RESTART)
3	DCTRPT	$N ($REPEAT)
4	DCTBKSP	$B ($BACKSPACE)
		$F
5	DCTHOLDJ	$T...,H
	DCTSPACE	$T...,C=1
2+4		$I
6-7		Reserved for Future Use.

14	20	DCTCHAIN	4	Address of Next Device Control Table.
18	24	DCTDEVN	8	EBCDIC Device Name.
20	32	DCTPRINT	1	Print Destination.
21	33	DCTPUNCH	1	Punch Destination.
22	34	DCTPRINC	1	Priority Increment.
23	35	DCTPRLIM	1	Priority Limit.
20	32	DCTFORMS	2	Current Forms Type (Packed).
22	34		1	Carriage Tape --

Bit	Name	Meaning
0	DCTFSPEC	Special Forms Routing.
1	DCTFOPER	Operator Controlled Forms.
2-7		Carriage Tape Type.

23	35		1	Universal Character Set --

Bit	Name	Meaning
0	DCTIDSEP	Generate Separator Page/Card.
1		Reserved for Future Use.
2-7		UCS Type.

Figure 8.5.3 -- UNIT RECORD DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
24	36	RIDUCB	4	Address of Internal Reader UCB.
28	40	RIDFLAGS	2	Synchronization Flags --
			Byte 1	Bit 0 RIDBUSY I/O Simulation in Progress.
				Bit 1-7 Reserved for Future Use.
			Byte 2	Reserved for Future Use.
2A	42	RIDCNT	2	Count of Transfer-In-Channel Commands.
2C	44	RIDECB	4	Address of Internal Reader ECB.
30	48	RIDTCB	4	Address of Internal Reader TCB.
34	52	RIDCCW	4	Address of Internal Reader CCW.
38	56	RIDDATA	80	Internal Reader Data Area.
24	36	DCTWORK	80	Punch Error Recovery Save Area.

Figure 8.5.4 -- LINE DEVICE CONTROL TABLE FORMAT

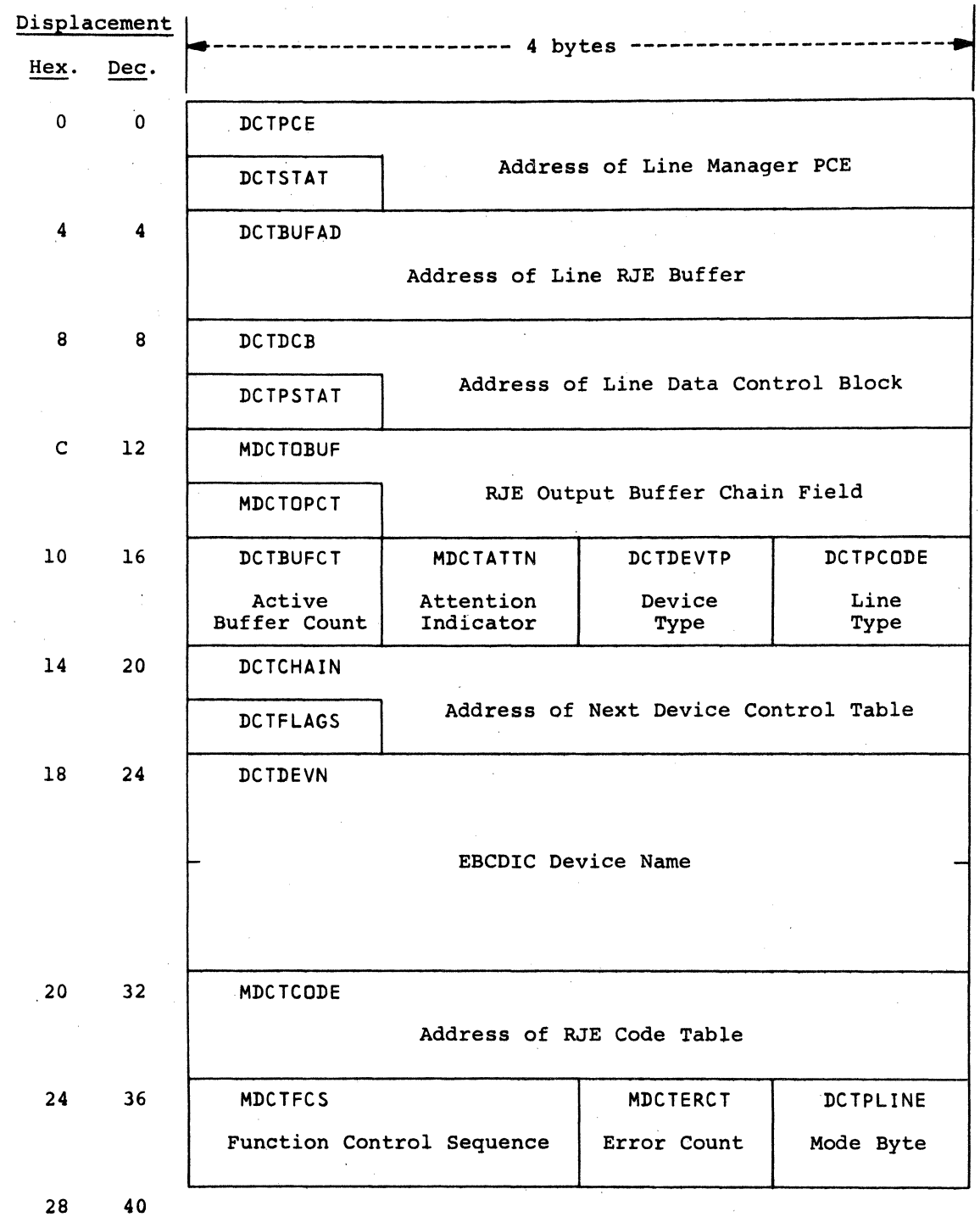


Figure 8.5.4 -- LINE DEVICE CONTROL TABLE FORMAT (CONTINUED)

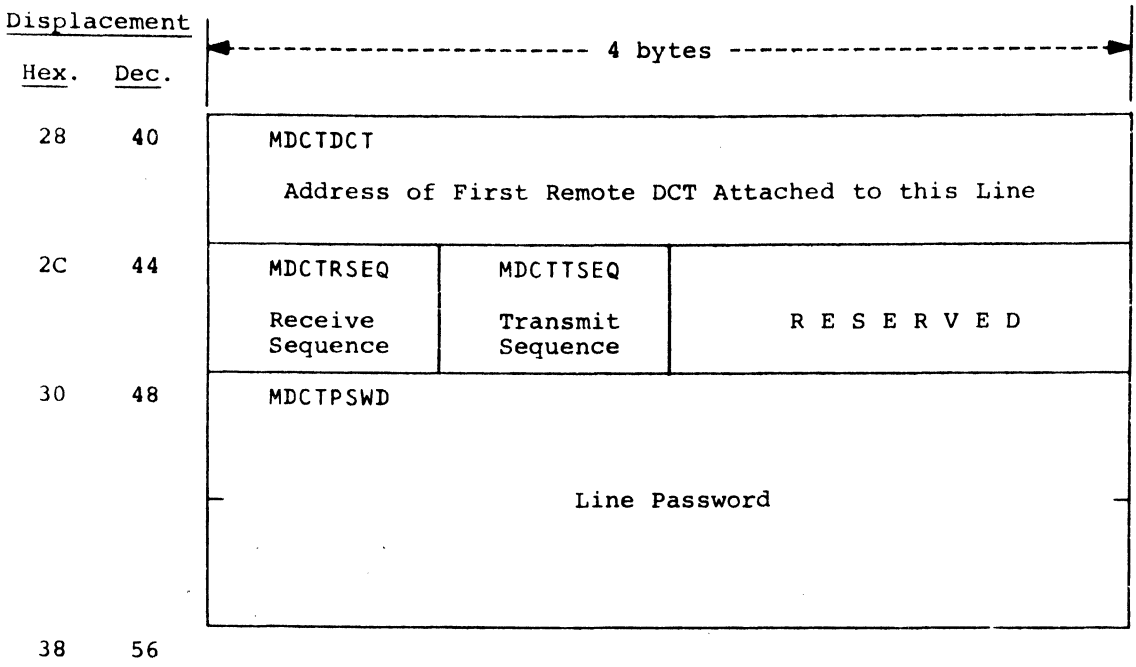


Figure 8.5.4 -- LINE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	DCTSTAT	1	DCT Status --

Bit	Name	Meaning
0	DCTINUSE	DCT is In Use.
1	DCTDRAIN	DCT is Drained.
2-7		Reserved.

0	0	DCTPCE	4	Address of Line Manager PCE.
4	4	DCTBUFAD	4	Address of Line RJE Buffer.
8	8	DCTPSTAT	1	Line Flags --

Bit	Name	Meaning
0	DCTLOGAL	Log Every Channel End.
1	DCTLEASE	Leased Line.
2	DCTETX	An ETX has been Received.
3	DCTSOFF	A /*SIGNOFF Card has been Processed.
4-7		Reserved for Future Use.

8	8	DCTDCB	4	Address of Line Data Control Block.
C	12	MDCTOPCT	1	MULTI-LEAVING Terminal Open Count.
C	12	MDCTOBUF	4	RJE Output Buffer Chain Field.
10	16	DCTBUFCT	1	Number of I/O Requests Outstanding.
11	17	MDCTATTN	1	Line Attention Requests --

Bit	Name	Meaning
0	MDCTIMER	Timed Action Requested.
1	MDCTPAWS	Line Pause Requested.
2	MDCTJOB1	Job Post Indicator 1.
3	MDCTJOB2	Job Post Indicator 2.
2+3	MDCTJOB	Job Post Indication.
4-7		Reserved for Future Use.

Figure 8.5.4 -- LINE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
12	18	DCTDEVTP	1	Device Type --

Hex. Value	Name	Device Type
02	DCTLNE	Line.

13	19	DCTPCODE	1	Line Type --

Bit	Name	Value	Meaning
0	DCTPBSC	0	STR Line.
		1	BSC Line.
1	DCTPTRSP	0	No Transparency.
		1	Transparency.
2	DCTPASCI	0	EBCDIC Code.
		1	USASCII Code.
3	DCTPHASP		Reserved.
4-5			Reserved.
6	DCTPWIDE	0	Low-Speed Line.
		1	Wide-Band Line.
7	DCTPHALF	0	Half-Duplex Line.
	DCTPFULL	1	Full-Duplex Line.

14	20	DCTFLAGS	1	Operator Command Flags --

Bit	Name	Command
0-1		Reserved.
2	DCTRSTRT	$E ($RESTART) -- Abort.
3-7		Reserved.

14	20	DCTCHAIN	4	Address of Next Device Control Table.
18	24	DCTDEVN	8	EBCDIC Device Name.
20	32	MDCTCODE	4	Address of RJE Code Table.
24	36	MDCTFCS	2	Last Function Control Sequence Received.
26	38	MDCTERCT	1	Line Error Count/Indicator.
27	39	DCTPLINE	1	SDA Mode Byte.
28	40	MDCTDCT	4	Address of First Remote DCT Attached to this Line.

Figure 8.5.4 -- LINE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
2C	44	MDCTRSEQ	1	Receive Block Sequence Count.
2D	45	MDCTTSEQ	1	Transmit Block Sequence Count.
2E	46		2	Reserved for Future Use.
30	48	MDCTPSWD	8	Line Password.

Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT

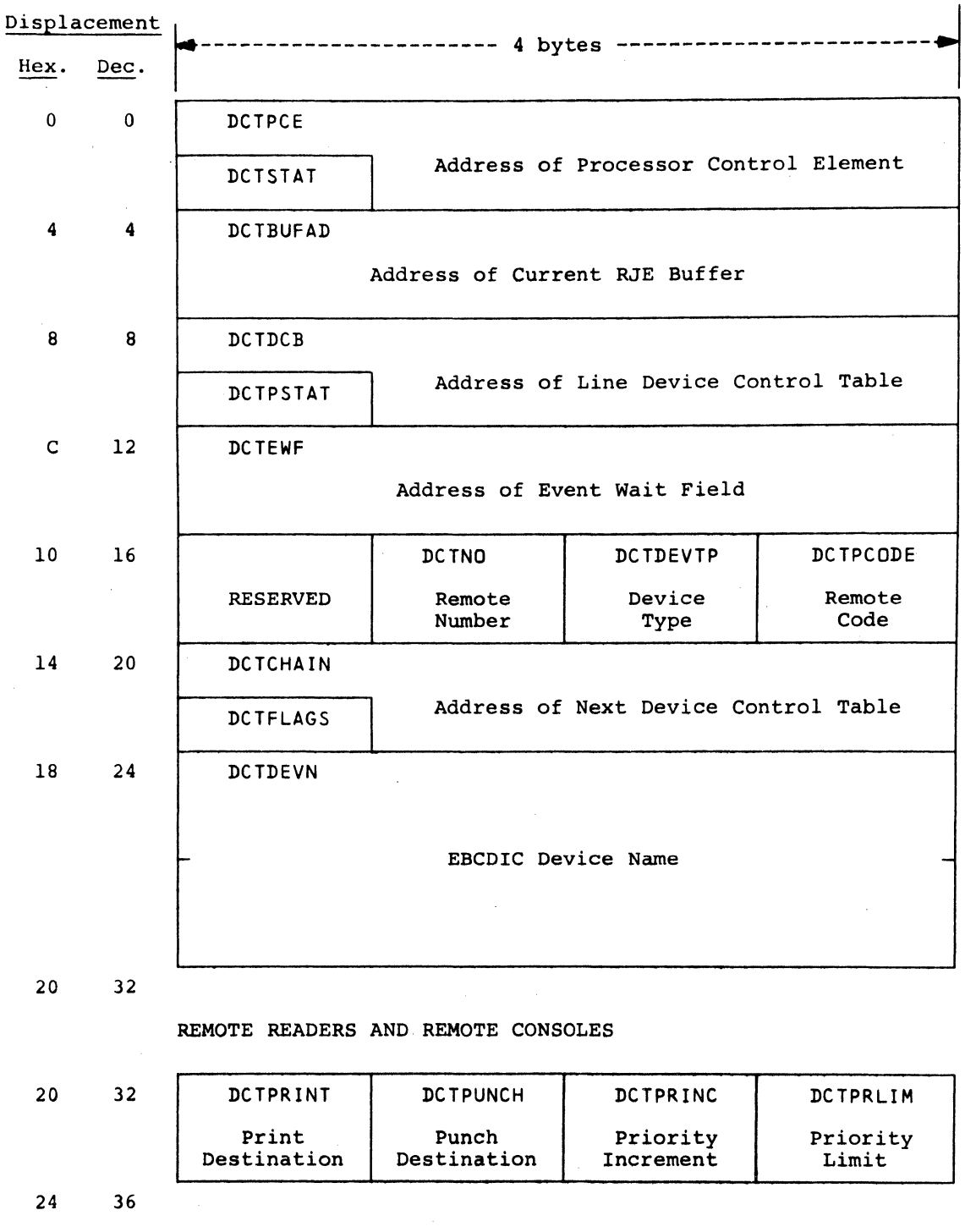


Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT (CONTINUED)

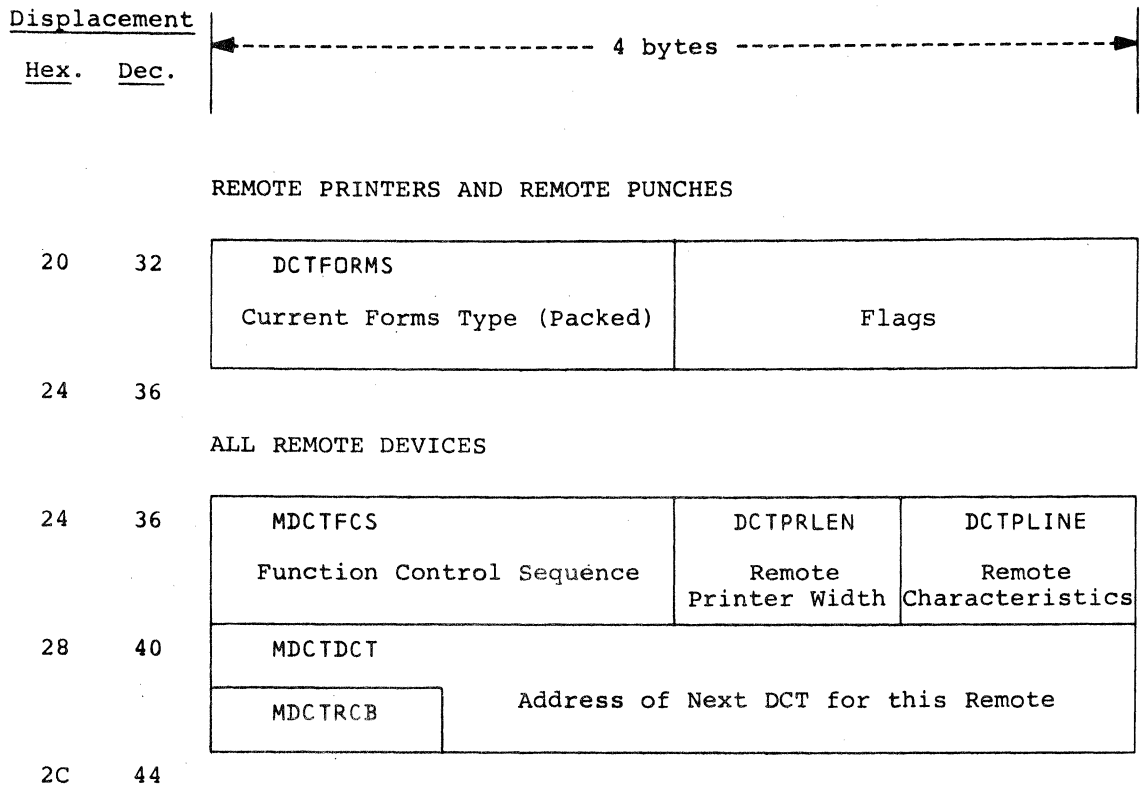


Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	DCTSTAT	1	DCT Status --

Bit	Name	Meaning
0	DCTINUSE	DCT is In Use.
1	DCTDRAIN	DCT is Drained.
2	DCTHOLD	DCT is Held.
3-7		Reserved.

0	0	DCTPCE	4	Address of Processor Control Element.
4	4	DCTBUFAD	4	Address of Current RJE Buffer.
8	8	DCTPSTAT	1	Remote Flags --

Bit	Name	Meaning
0-3		Reserved for Future Use.
4	DCTSINON	DCT is Attached to Line DCT.
5	DCTPOST	I/O Complete Flag.
6	DCTABORT	Transmission was Aborted.
7	DCTPBUF	Remote has Output Buffer.

8	8	DCTDCB	4	Address of Line Device Control Table.
C	12	DCTEWF	4	Address of Event Wait Field.
10	16		1	Reserved -- Must be zero.
11	17	DCTNO	1	Remote Number.
12	18	DCTDEVTP	1	Device Type --

Hex. Value	Name	Meaning
12	DCTRJR	Remote Reader.
22	DCTRPR	Remote Printer.
32	DCTRPU	Remote Punch.
42	DCTRCON	Remote Console.

Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
13	19	DCTPCODE	1	Remote Code --

Bit	Name	Meaning
0		Reserved.
1	DCTPTRSP	Terminal Transparency.
2		Reserved.
3	DCTPCON	Terminal Console.
	DCTPMRF	Multiple-Record Feature.
		Buffer Expansion Feature.
4	DCTPTAB	Horizontal Format Control.
5	DCTPROG	Programmable Interface.
6	DCTPVAR	Variable Length Records.
7	DCTPBLK	Blocked Records.

14	20	DCTFLAGS	1	Operator Command Flags --

Bit	Name	Command
0	DCTSTOP	$Z ($STOP)
1	DCTDELET	$C ($DELETE)
2	DCTRSTRT	$E ($RESTART)
3	DCTRPT	$N ($REPEAT)
4	DCTBKSP	$B ($BACKSPACE)
		$F
5	DCTHOLDJ	$T...,H
	DCTSPACE	$T...,C=1
2+4		$I
6-7		Reserved for Future Use.

14	20	DCTCHAIN	4	Address of Next Device Control Table.
18	24	DCTDEVN	8	EBCDIC Device Name.
20	32	DCTPRINT	1	Print Destination.
21	33	DCTPUNCH	1	Punch Destination.
22	34	DCTPRINC	1	Priority Increment.
23	35	DCTPRLIM	1	Priority Limit.

Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Dec.	Field Name	Bytes	Field Description
20	32	DCTFORMS	2	Current Forms Type (Packed).
22	34		2	Flags --

Byte 1

Bit	Name	Meaning
0	DCTFSPEC	Special Forms Routing.
1	DCTFOPER	Operator Controlled Forms.
2-7		Reserved for Future Use.

Byte 2

Bit	Name	Meaning
0	DCTIDSEP	Generate Separator Page/Card.
1-7		Reserved for Future Use.

24	36	MDCTFCS	2	Function Control Sequence Mask --

	Bit	Meaning
Byte 1	0-3	Reserved.
	4	Reader 1 or Printer 1.
	5	Reader 2, Printer 2, or Punch 7.
	6	Reader 3, Printer 3, or Punch 6.
	7	Reader 4, Printer 4, or Punch 5.
Byte 2	0	Reserved.
	1	Remote Console.
	2-3	Reserved.
	4	Reader 5, Printer 5, or Punch 4.
	5	Reader 6, Printer 6, or Punch 3.
	6	Reader 7, Printer 7, or Punch 2.
	7	Punch 1.

26	38	DCTPRLEN	1	Remote Printer Width and Remote Input Size.
27	39	DCTPLINE	1	Remote Characteristics --

Bits 0-3 Adapter/Terminal Characteristics

Bit	Name	Value	Meaning
0	DCTPBSC	0	STR Adapter.
		1	BSC Adapter.
1	DCTPTRSP	0	No Transparency.
		1	Transparency.
2	DCTPASCI	0	EBCDIC Code.
		1	USASCII Code.
3			Reserved.

Figure 8.5.5 -- REMOTE DEVICE CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
				Remote Characteristics (continued) --
			Bits 4-7	Terminal Type

Hex. Value	Name	Terminal Type
0	DCTP2770	2770, 1009.
1	DCTPHARD	2780, 1978.
2	DCTP20	360/20 Sub-Model 5.
4	DCTP360	360/25, 30, 40, etc.
6	DCTP20S2	360/20 Sub Model 2.
8	DCTP1130	1130.
A	DCTPSYS3	System/3.

28	40	MDCTRCB	1	Record Control Byte --

Bits	Meaning
0	Always One.
1-3	Device Number.
4-7	Device Type --

Value	Device Type
1	Output Console.
2	Input Console.
3	Reader.
4	Printer.
5	Punch.

28	40	MDCTDCT	4	Address of Next DCT for this Remote.

Figure 8.6.1 -- JOB QUEUE ELEMENT FORMAT

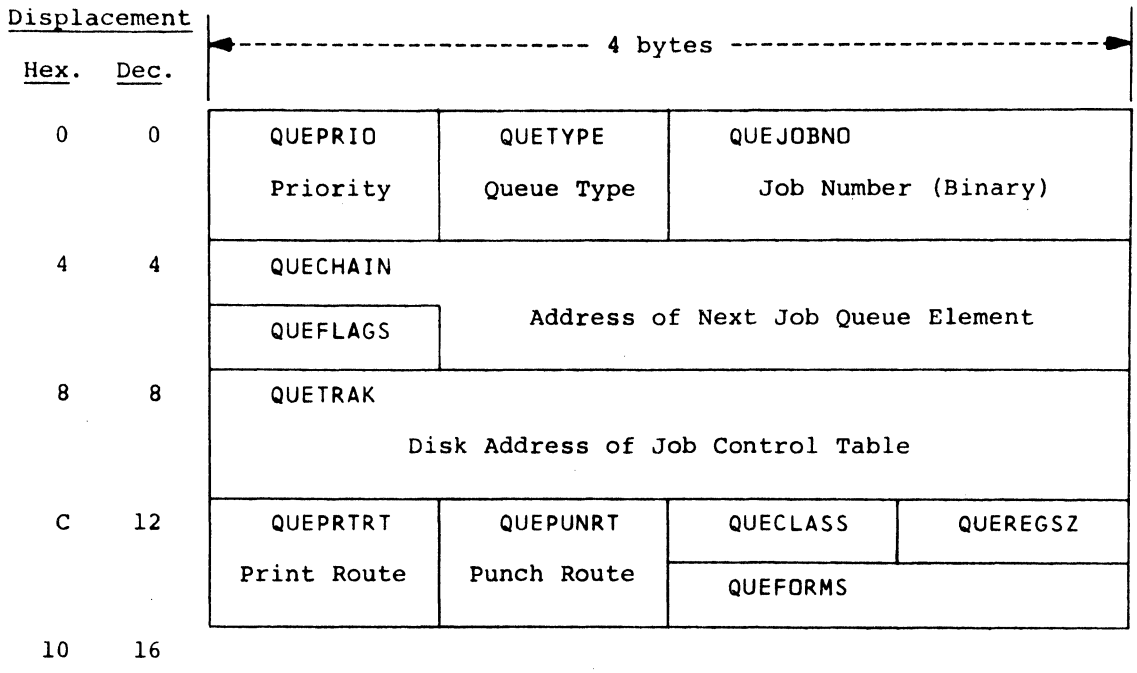


Figure 8.6.1 -- JOB QUEUE ELEMENT FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	QUEPRIO	1	Queueing Priority --
			Bits 0-3	Priority (0-15).
			Bits 4-7	Reserved.
1	1	QUETYPE	1	Queue Type --

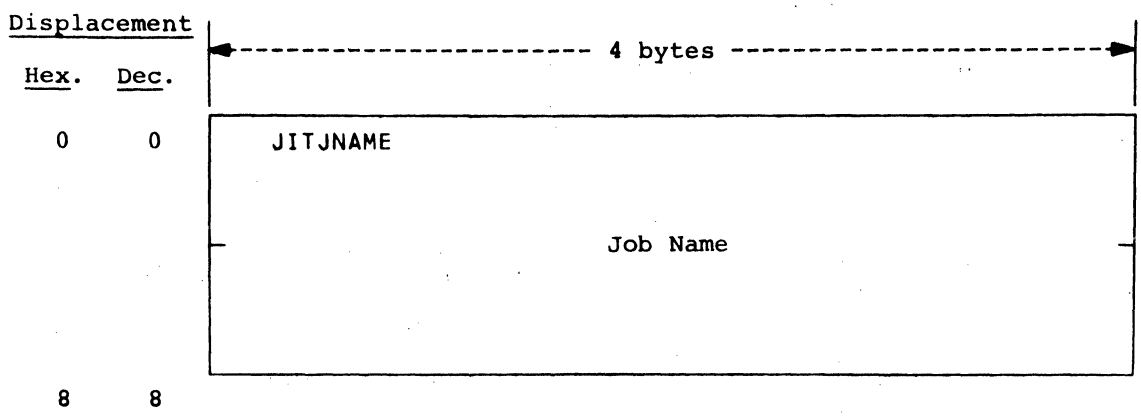
Binary Value	Name	Meaning
lxxxxxxx	QENTBY	Queue Entry is In Use.
xlcccccc	$XEQ	Execution -- cccccc = Job Class - X'C0'.
x0100000	$INPUT	Input Queue.
x0000100	$PRINT	Print Queue.
x0000010	$PUNCH	Punch Queue.
x0000000	$PURGE	Purge Queue.

2	2	QUEJOBNO	2	Job Number (Binary)
4	4	QUEFLAGS	1	Queue Flags --

Bit	Name	Meaning
0	QUEHOLDA	Job Held ($H A)
1	QUEHOLD1	Job Held (Single Job)
2	QUEHOLD2	Job Held (Duplicate Job Name).
3	QUEPURGE	Job Deleted.
4-7	QUEUSECT	Entry Use Count.

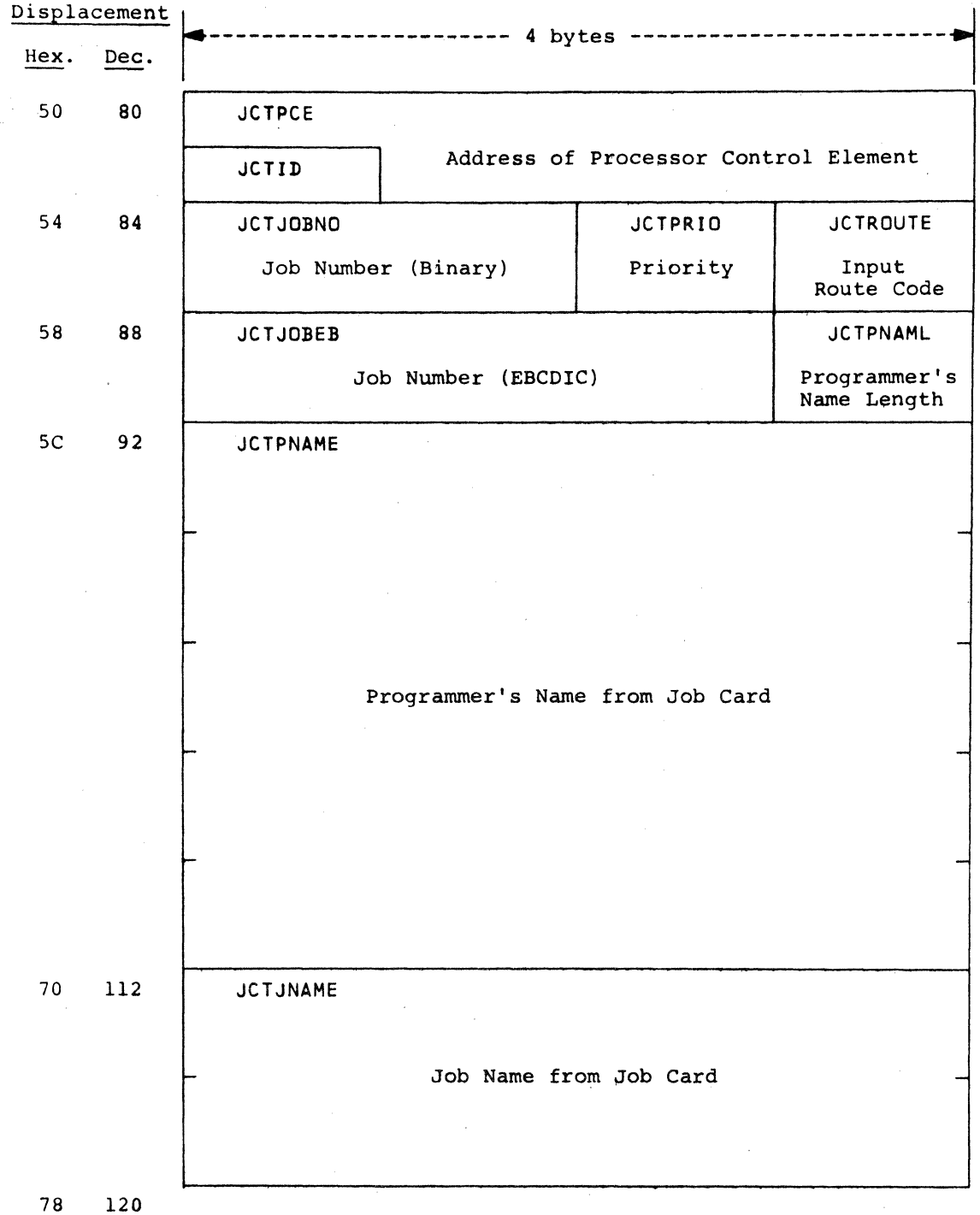
4	4	QUECHAIN	4	Address of Next Job Queue Element.
8	8	QUETRAK	4	Track Address of Job Control Table.
C	12	QUEPRTRT	1	Print Routing: 0 = Local. n = Remote n.
D	13	QUEPUNRT	1	Punch Routing: 0 = Local. n = Remote n.
E	14	QUECLASS	1	Sub-Class -- Unused.
E	14	QUEFORMS	2	Forms Code (Packed).
F	15	QUEREGSZ	1	Region Size -- Unused.

Figure 8.7.1 -- JOB INFORMATION TABLE ELEMENT FORMAT



Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	JITJNAME	8	Job Name.

Figure 8.8.1 -- JOB CONTROL TABLE FORMAT



Job Control Table Format -- Page 8.8-1

Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

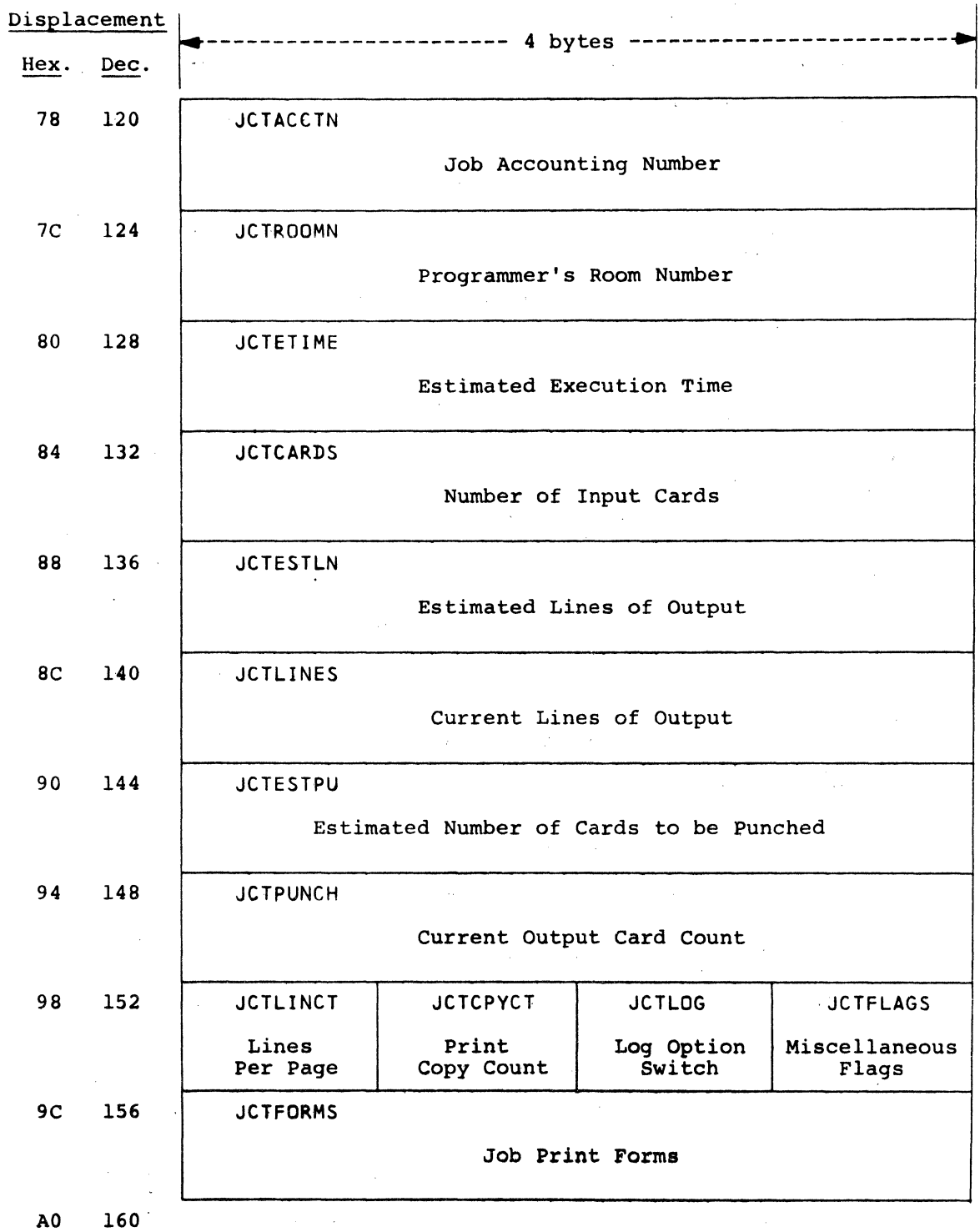


Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

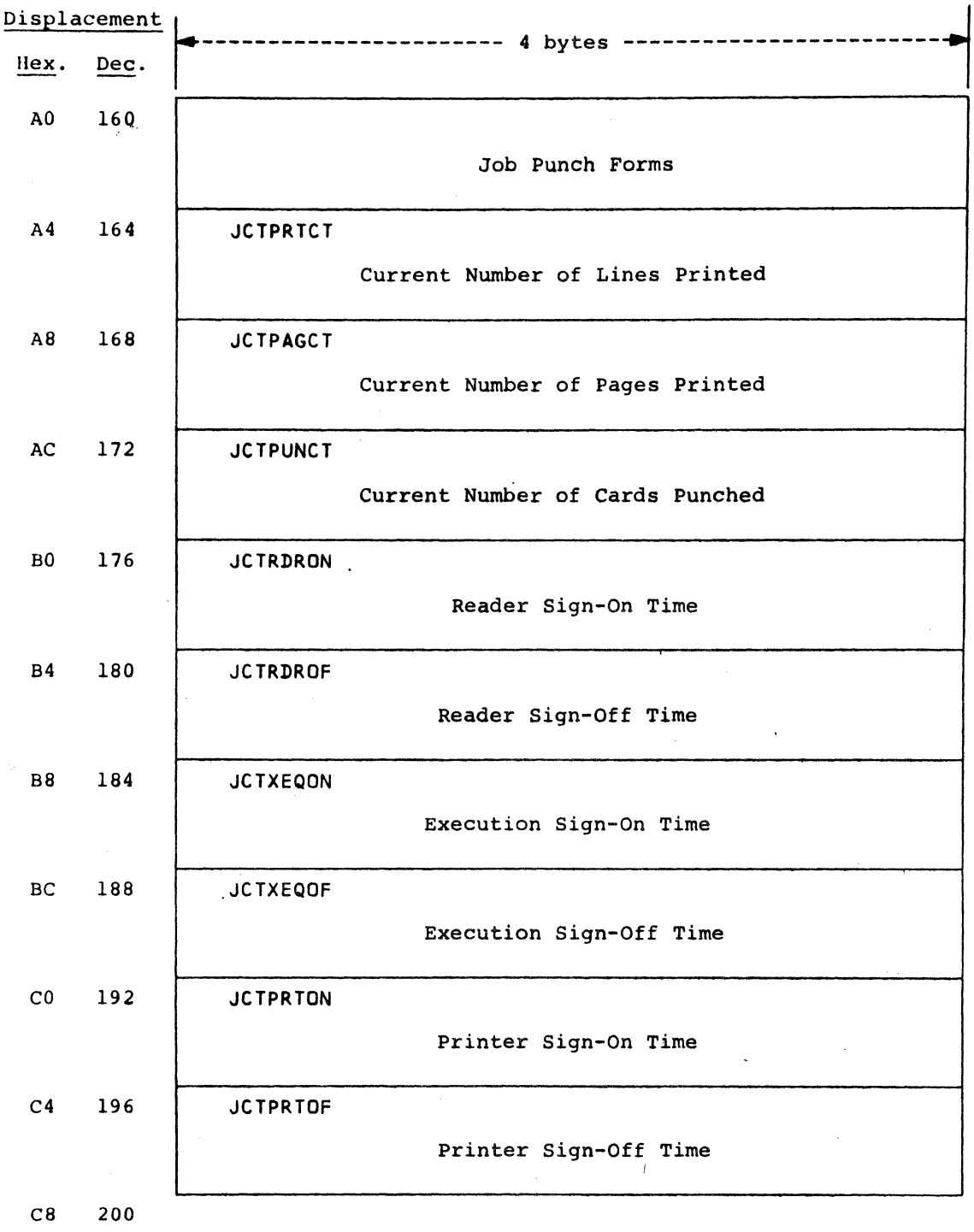


Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

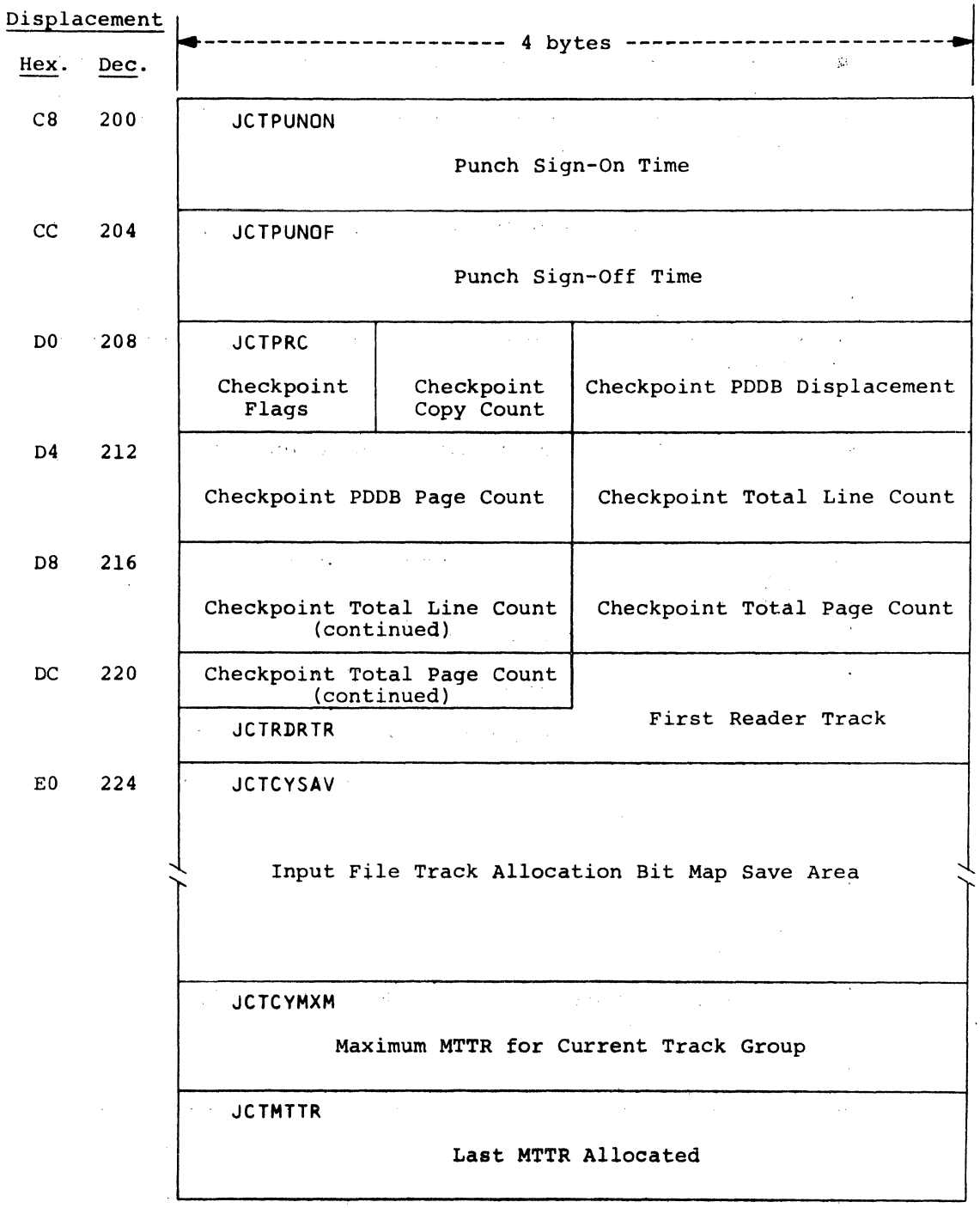


Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

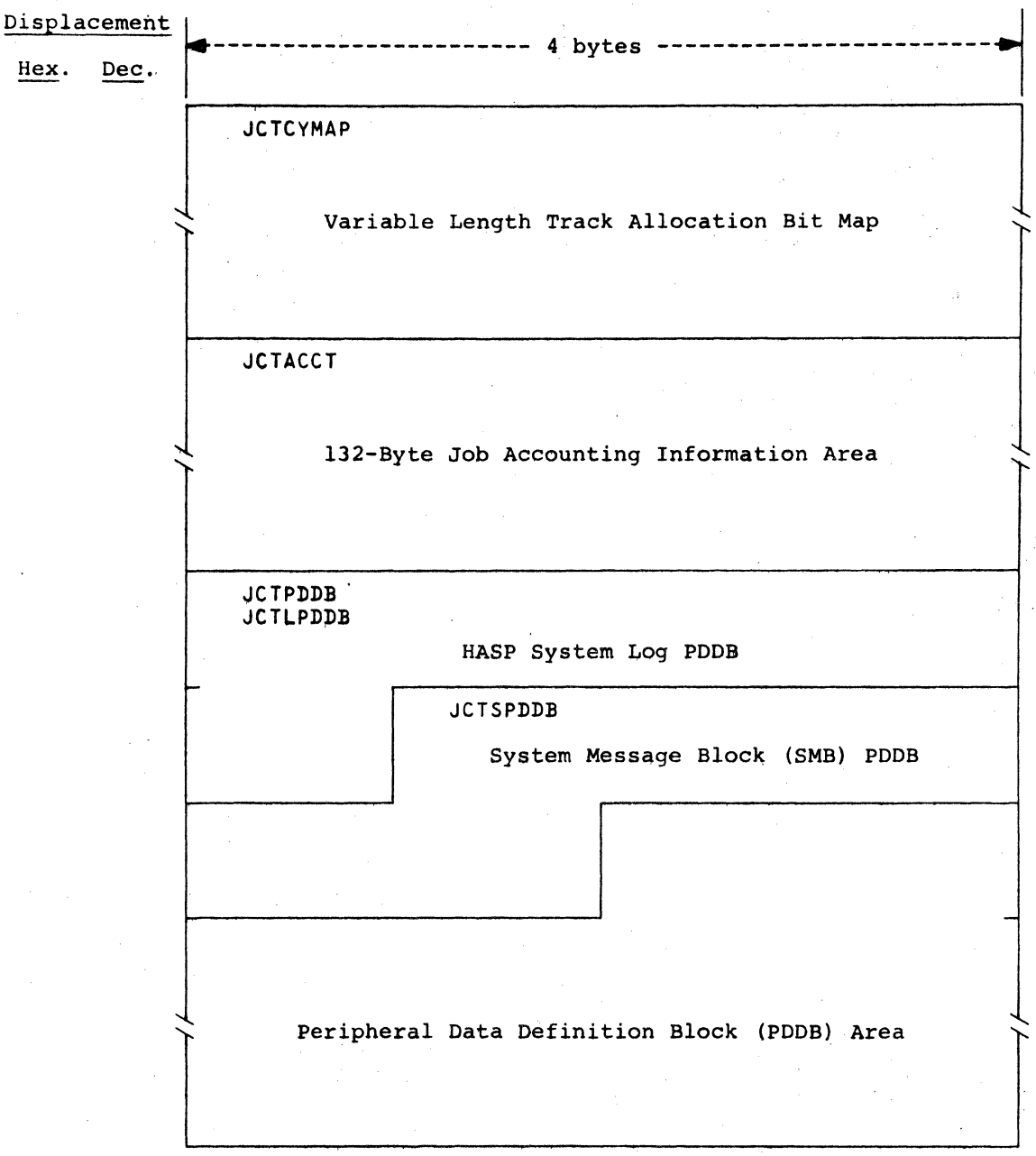


Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
50	80	JCTID	1	JCT Identification -- X'FF'.
50	80	JCTPCE	4	Address of Processor Control Element.
54	84	JCTJOBNO	2	Job Number (Binary).
56	86	JCTPRIO	1	Priority from /*PRIORITY Card.
57	87	JCTROUTE	1	Route Code of Input Device: 0 = Local. n = Remote n.
58	88	JCTJOBEB	3	Job Number (EBCDIC).
5B	91	JCTPNAML	1	Length of Programmer's Name.
5C	92	JCTPNAME	20	Programmer's Name from Job Card.
70	112	JCTJNAME	8	Job Name from Job Card.
78	120	JCTACCTN	4	Job Accounting Number.
7C	124	JCTROOMN	4	Programmer's Room Number.
80	128	JCTETIME	4	Estimated Execution Time.
84	132	JCTCARDS	4	Number of Input Cards.
88	136	JCTESTLN	4	Estimated Lines of Output.
8C	140	JCTLINES	4	Generated Lines of Output.
90	144	JCTESTPU	4	Estimated Number of Cards to be Punched.
94	148	JCTPUNCH	4	Number of Output Cards Generated.
98	152	JCTLINCT	1	Lines per Page.
99	153	JCTCPYCT	1	Number of Copies of Print.
9A	154	JCTLOG	1	Log Option Switch --

EBCDIC Value	Meaning
L	Produce HASP SYSTEM LOG.
N	Do not Produce HASP SYSTEM LOG.

Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
9B	155	JCTFLAGS	1	Miscellaneous Flags --

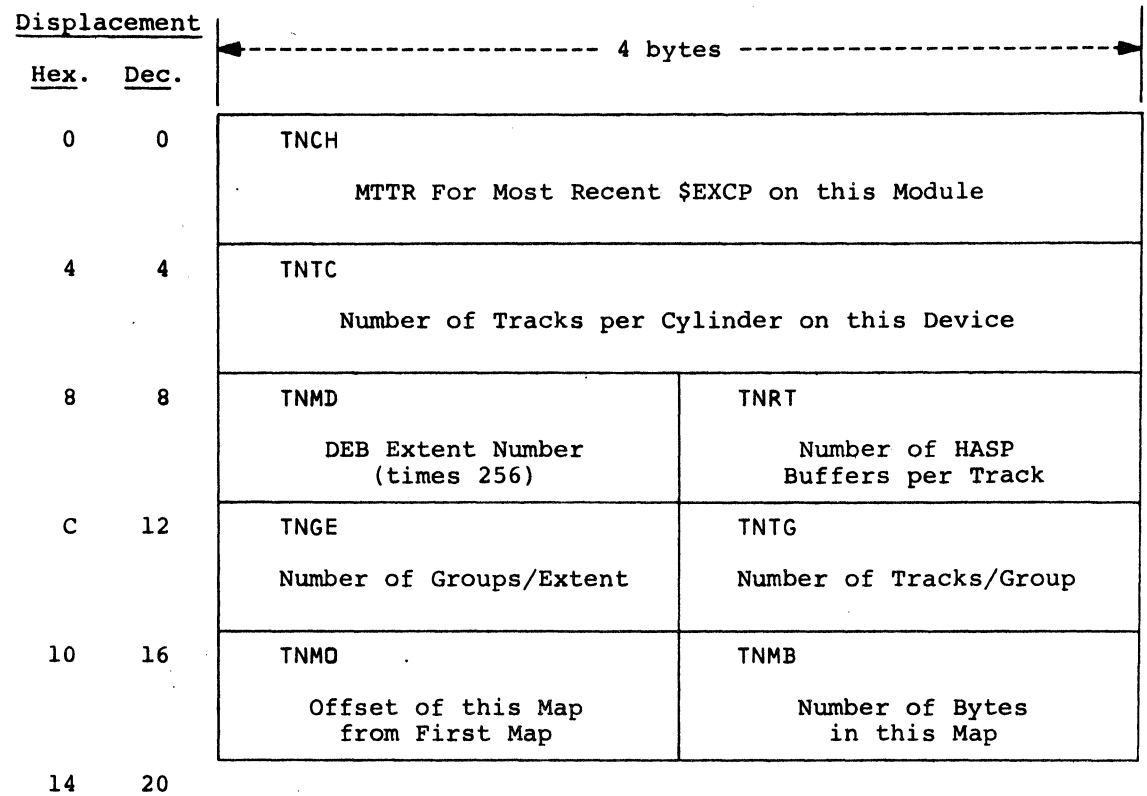
Bit	Name	Meaning
0	JCTDSRT	Processing Special Forms.
1-7		Count of Input Data Sets SPOOLed by HASP.

9C	156	JCTFORMS	4	Job Print Forms.
A0	160		4	Job Punch Forms.
A4	164	JCTPRTCT	4	Number of Lines Printed.
A8	168	JCTPAGCT	4	Number of Pages Printed.
AC	172	JCTPUNCT	4	Number of Cards Punched.
B0	176	JCTRDRON	4	Reader Sign-On Time.
B4	180	JCTRDROF	4	Reader Sign-Off Time.
B8	184	JCTXEQON	4	Execution Sign-On Time.
BC	188	JCTXEQOF	4	Execution Sign-Off Time.
C0	192	JCTPRTON	4	Print Sign-On Time.
C4	196	JCTPRTOF	4	Print Sign-Off Time.
C8	200	JCTPUNON	4	Punch Sign-On Time.
CC	204	JCTPUNOF	4	Punch Sign-Off Time.
D0	208	JCTPRC	14	Print Checkpoint Element.
DC	220	JCTRDRTR	4	First Reader Track.
E0	224	JCTCYSAV		Variable Length Input File Track Allocation Bit Map Save Area.
		JCTCYMXM	4	Maximum MTTR for Current Track Group.
		JCTMTTR	4	Last MTTR Allocated.
		JCTCYMAP		Variable Length Track Allocation Bit Map.

Figure 8.8.1 -- JOB CONTROL TABLE FORMAT (CONTINUED)

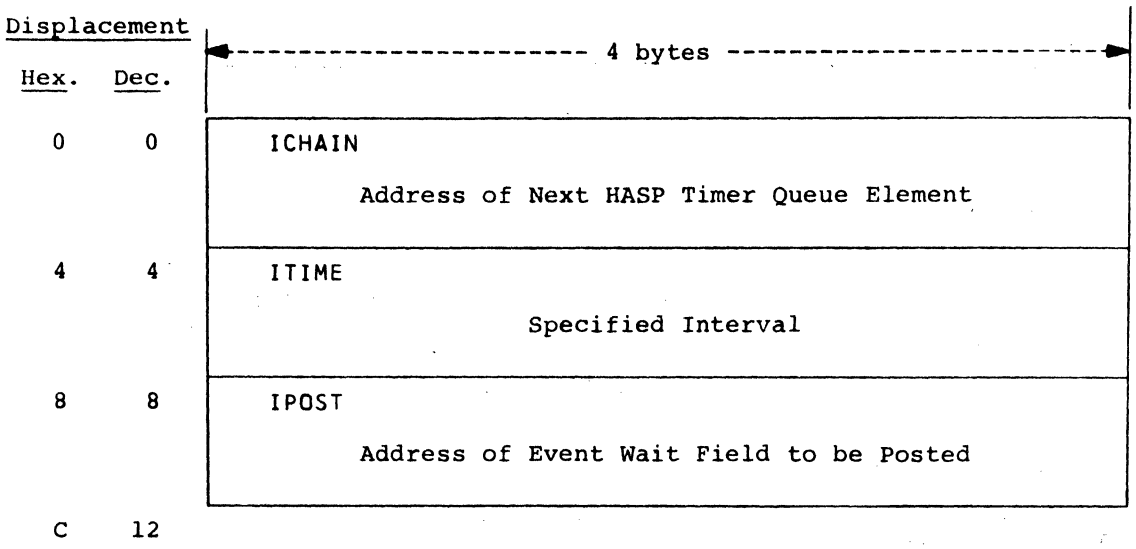
Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
		JCTACCT	132	Job Accounting Information Area.
		JCTPDDB		Peripheral Data Definition Block (PDDB) Area.
		JCTLPDDB	5	HASP System Log PDDB.
		JCTSPDDB	5	System Message Block (SMB) PDDB.

Figure 8.9.1 -- TRACK EXTENT DATA TABLE FORMAT



Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	TNCH	4	MTTR for Most Recent $EXCP on this Module.
4	4	TNTC	4	Number of Tracks per Cylinder on this Device.
8	8	TNMD	2	DEB Extent Number (times 256).
A	10	TNRT	2	Number of HASP Buffers per Track.
C	12	TNGE	2	Number of Track Groups per Extent.
E	14	TNTG	2	Number of Tracks per Track Group.
10	16	TNMO	2	Offset of This Map from First Map.
12	18	TNMB	2	Number of Bytes in This Map.

Figure 8.10.1 -- TIMER QUEUE ELEMENT



Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	ICHAIN	4	Address of Next HASP Timer Queue Element.
4	4	ITIME	4	Timer Interval.
8	8	IPOST	4	

Byte 1 Flag Byte --

Bit	Value	Meaning
0	0	Timer Interval has not Expired.
	1	Timer Interval has Expired.
1-7		Reserved.

Bytes 2-4 Address of Event Wait Field to be Posted.

Timer Queue Element Format -- Page 8.10-1

Figure 8.11.1 -- OVERLAY TABLE FORMAT

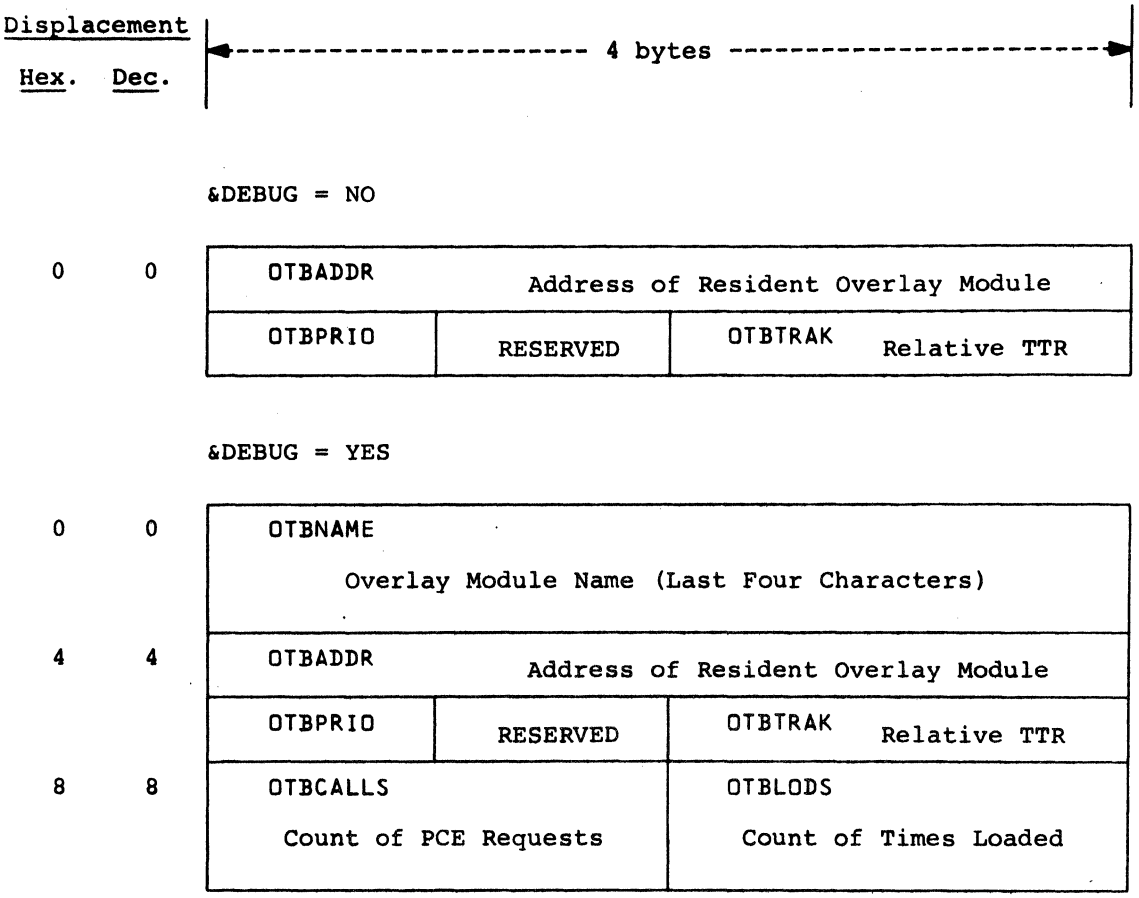


Figure 8.11.1 -- OVERLAY TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	OTBADDR	4	Address of Resident Overlay Module.
			Byte 1	X'FF'.
			Bytes 2-4	Addressability Address of Resident Overlay Module -- Assembly Origin - X'50'.
0	0	OTBPRIO	1	Priority of Overlay Module.
1	1		1	Reserved for Future Use.
2	2	OTBTRAK	2	Relative Track and Record Address of Overlay Module.
0	0	OTBNAME	4	Last Four Characters of Overlay Module Name.
8	8	OTBCALLS	2	Number of Times This Overlay Module was Requested.
A	10	OTBLODS	2	Number of Times This Overlay Module was Loaded.

Figure 8.12.1 -- DATA DEFINITION TABLE FORMAT

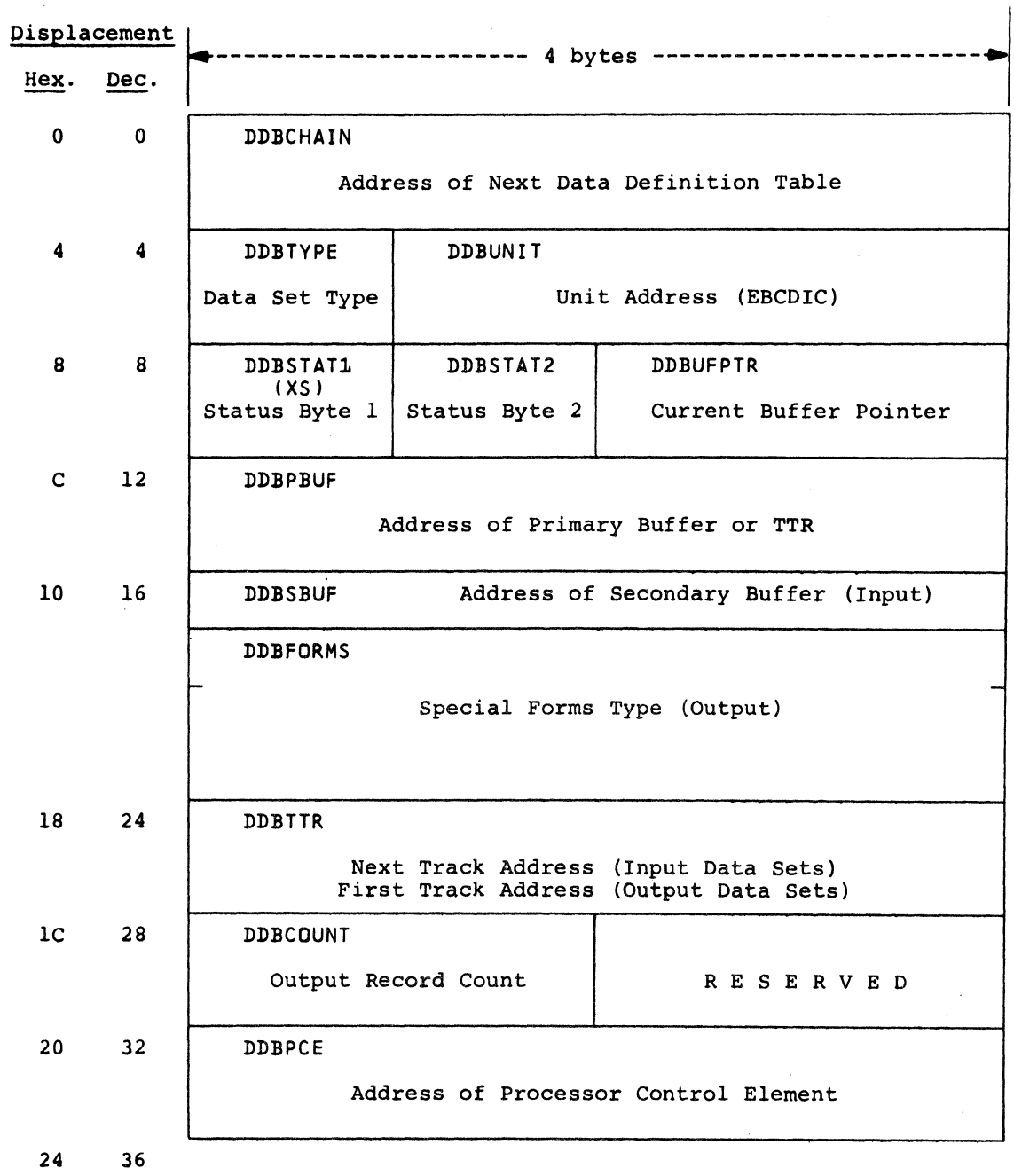


Figure 8.12.1 -- DATA DEFINITION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Definition
0	0	DDBCHAIN	4	Address of Next Data Definition Table.
4	4	DDBTYPE	1	Data Set Type --

Hex. Value	Name	Data Set Type
01	XSPROUTE	Special Route SYSOUT.
02	XPRTDDB	Print.
04	XPUNDDB	Punch.
08	XPLOTDDB	Plot.
10	XLOGDDB	Log.
40	XNULLDDB	Dummy (Null).
80	XINDDB	Input.

5	5	DDBUNIT	3	Unit Address (EBCDIC).
8	8	DDBSTAT1 (XS)	1	Status Byte 1 --

Bit	Name	Meaning
0	XSEOD	End of Data on Secondary.
1	XSIOA	I/O Active on Secondary.
2	XSIO	I/O Required on Secondary.
3	XNSB	No Secondary Buffer.
4	XPEOD	End of Data on Primary.
5	XPIOA	I/O Active on Primary.
6	XPIO	I/O Required on Primary.
7	XNPB	No Primary Buffer.

9	9	DDBSTAT2	1	Status Byte 2 --

Bit	Name	Meaning
0	XACT	Action Required on This DDT.
1		Reserved for Future Use.
2	XLOGHEAD	Log Title Switch.
3	XOPEN	DDT has been Used.
4	XUCB	Allocatable UCB Exists.
5	XIOC	I/O Error on Read.
6	XROLL	Roll Output Buffer.
7	XTERM	Terminate DDT.

Figure 8.12.1 -- DATA DEFINITION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
A	10	DDBUFPTR	2	Current Displacement of Data in Primary Buffer.
C	12	DDBPBUF	4	Address of Primary Buffer -- Or TTR if No Primary Buffer.
10	16	DDBSBUF	4	Address of Secondary Buffer (Input Only).
10	16	DDBFORMS	8	Special Forms Type (Output Only).
18	24	DDBTTR	4	Input: Next Track Address. Output: First Track Address.
1C	28	DDBCOUNT	2	Output Record Count.
1E	30		2	Reserved for Future Use.
20	32	DDBPCE	4	Address of Processor Control Element.

Figure 8.13.1 -- PARTITION INFORMATION TABLE FORMAT

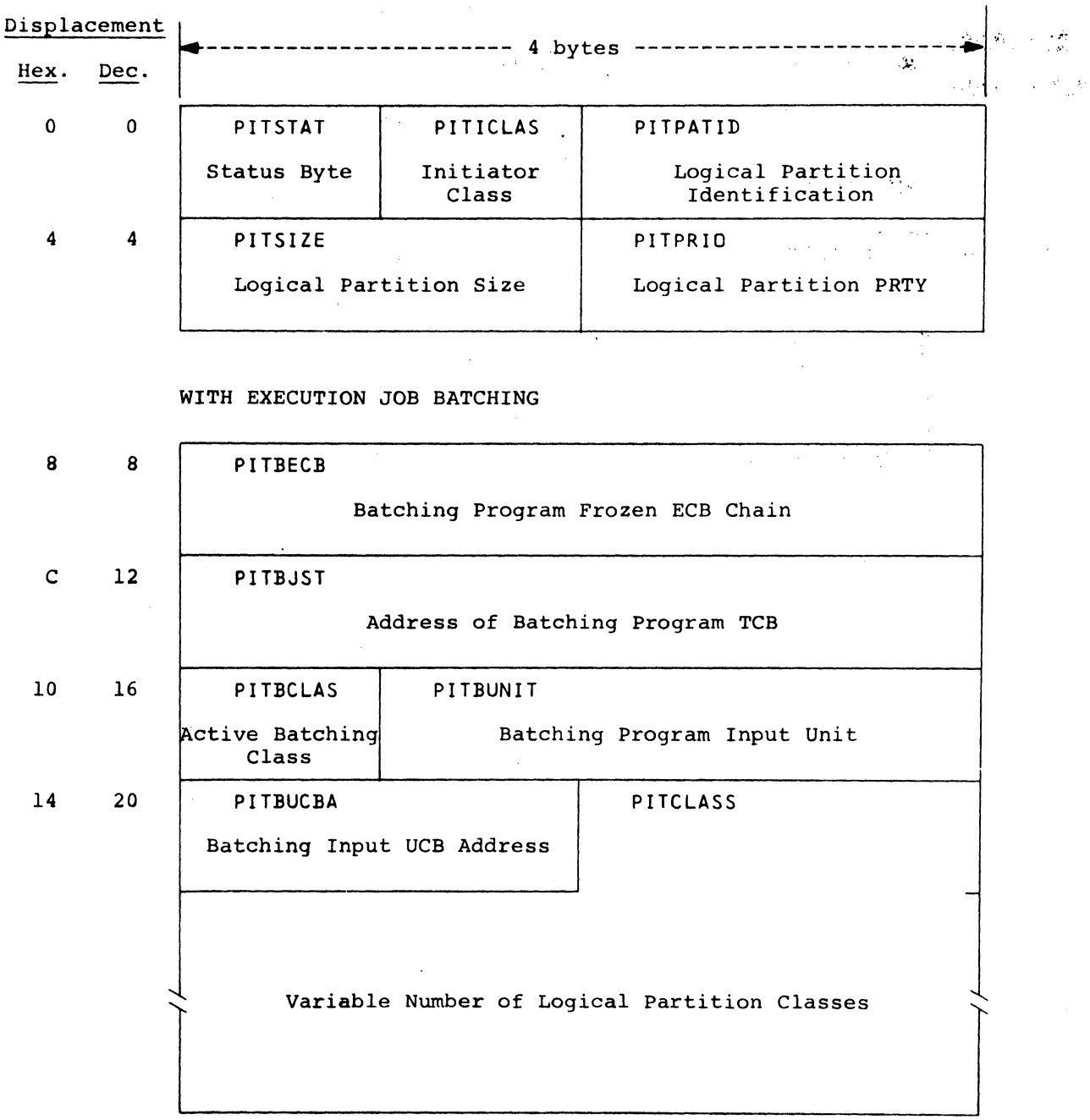


Figure 8.13.1 -- PARTITION INFORMATION TABLE FORMAT (CONTINUED)

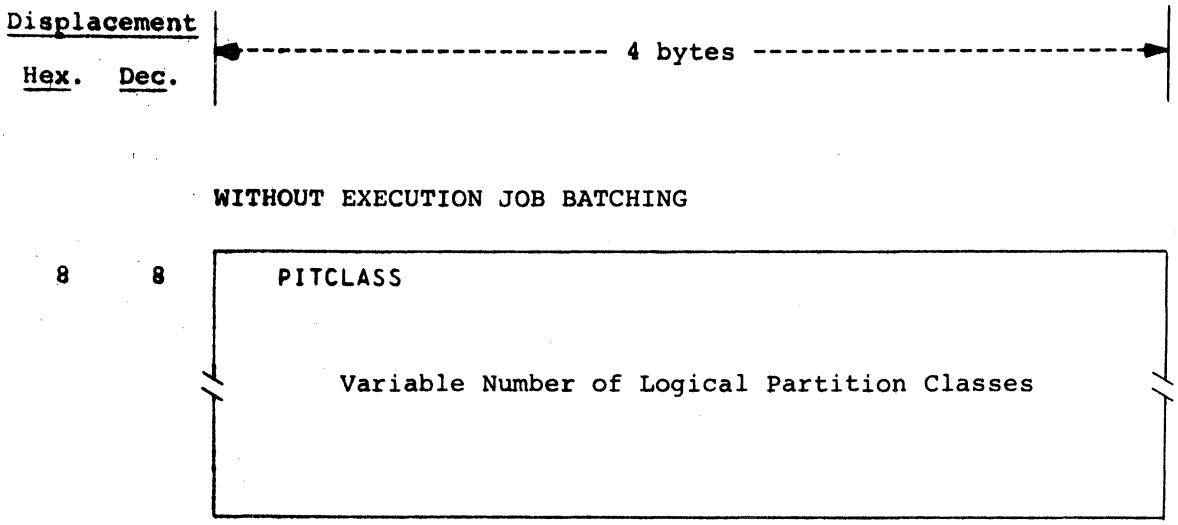


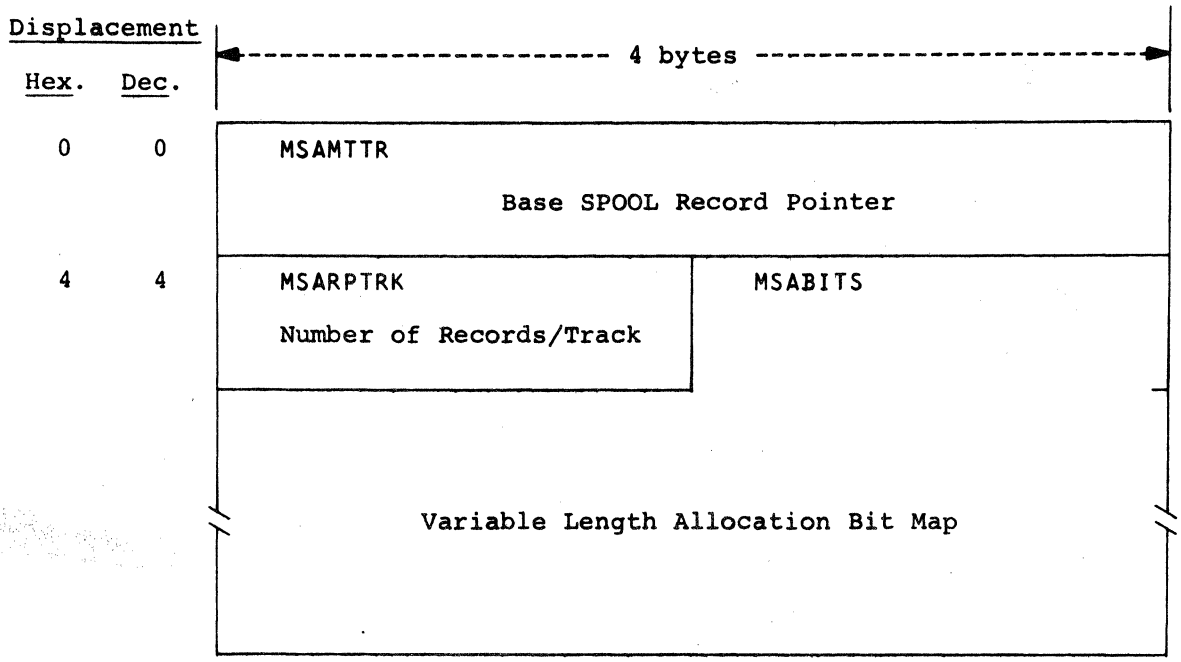
Figure 8.13.1 -- PARTITION INFORMATION TABLE FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	PITSTAT	1	Status Byte --

Bit	Name	Meaning
0	PITHOLDA	PIT is Drained ($P I).
1	PITHOLD1	PIT is Drained ($P In).
2	PITBUSY	Partition Busy Indicator.
3	PITIDLE	PIT Idle Message Switch.
4-6		Reserved for Future Use.
7	PITLAST	Last PIT Indicator.

1	1	PITICLAS	1	O/S Initiator Class.
2	2	PITPATID	2	Logical Partition Identification.
4	4	PITSIZE	2	Logical Partition Size (Unused).
6	6	PITPRIO	2	Logical Partition PRTY.
8	8	PITBECB	4	Batching Program Frozen ECB Chain.
C	12	PITBJST	4	Address of Batching Program TCB.
10	16	PITBCLAS	1	Active Batching Class.
11	17	PITBUNIT	3	Batching Program Input Unit.
14	20	PITBUCBA	2	Batching Input Unit Control Block (UCB) Address.
		PITCLASS		Variable Number of Logical Partition Classes.

Figure 8.14.1 -- MESSAGE ALLOCATION CONTROL BLOCK



Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
0	0	MSAMTTR	4	Base SPOOL Record Pointer.
4	4	MSAPTRK	2	Number of Records/Track.
6	6	MSABITS		Variable Length Allocation Bit Map.

Figure 8.15.1 -- DATA BLOCK FORMAT

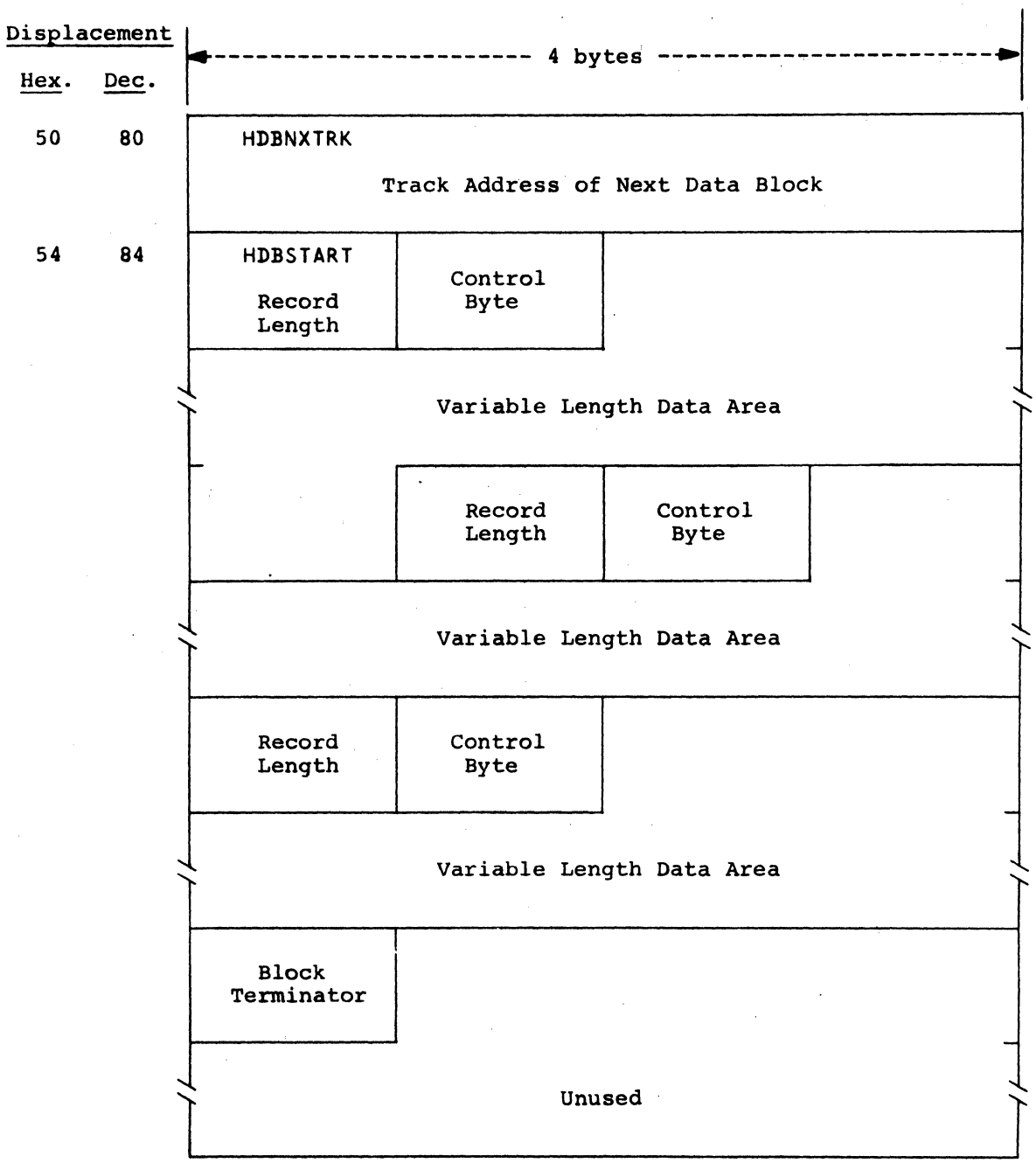


Figure 8.15.1 -- DATA BLOCK FORMAT (CONTINUED)

Displacement Hex.	Displacement Dec.	Field Name	Bytes	Field Description
50	80	HDBNXTRK	4	Track Address of Next Data Block.
54	84	HDBSTART		Start of Data Block.
		Record Length	1	Length of Data Area (0-254).
		Control Byte	1	Control Byte --

Input Data Sets

Hex. Value	Meaning
00	Normal Record.
03	Internally Generated Card.
04	HASP Control Card.
13	Illegal HASP Control Card.
19	Last JCL Card.
73	Dummy Track Address Record.

Print Data Sets -- Carriage Control.

Punch Data Sets -- Stacker Select.

Data Area	Variable Length Data Area.
Block Terminator	Record Length of 255 (X'FF').

SR27-9720-0

OS/HASP Volume 1 Supplementary Course Material Printed in U.S.A. SR27-9720-0

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]