SPERRY UNIVAC
COMPUTER SYSTEMS

PUBLICATIONS UPDATE

Operating System/3 (OS/3)

Supervisor

User Guide

UP-8075 Rev. 3–A

This Library Memo announces the release and availability of Updating Package A to "SPERRY UNIVAC Operating System/3 (OS/3) Supervisor User Guide", UP-8075 Rev. 3.

This update documents the following changes for release 8.0:

- Enhancement of the OC STXIT routine
- Restrictions to the monitor routine
- Expansion of the Soft-Patch Symbiont debugging aid
- Enhancement of the job accounting facility

This update also includes minor technical corrections to material applicable to the supervisor prior to release 8.0.

Copies of Updating Package A are now available for requisitioning. Either the updating package only, or the complete manual with the updating package may be requisitioned by your local Sperry Univac representative. To receive only the updating package, order UP-8075 Rev. 3–A. To receive the complete manual, order UP-8075 Rev. 3.

LIBRARY MEMO ONLY	LIBRARY MEMO AND ATTACHMENTS	THIS SHEET IS
Mailing Lists BZ, CZ and MZ	Mailing Lists A00, A01, 18, 18U, 19, 19U, 20, 20U, 21, 21U, 75, 75U, 76, and 76U (Package A to UP-8075 Rev. 3, 75 pages plus Memo)	Library Memo for UP-8075 Rev. 3–A
		RELEASE DATE: September, 1982

PAGE STATUS SUMMARY

ISSUE: Update A – UP-8075 Rev. 3
RELEASE LEVEL: 8.0 Forward

Part/Section	Page Number	Update Level
Cover/Disclaimer		Orig.
PSS	1	A
Preface	1, 2	Orig.
Contents	1 2, 3 4 thru 6 7, 8 9, 10	Orig. A Orig. A Orig.
PART 1	Title Page	Orig.
1	1 thru 3	Orig.
2	1 thru 8 9 10, 11	Orig. A Orig.
3	1 thru 10 11 12	Orig. A Orig.
PART 2	Title Page	Orig.
4	1 thru 3 4 5 6 thru 14 15 16 thru 18 19 thru 22 23 thru 28 29 30 thru 34 35 thru 37	A Orig. A Orig. A Orig. A Orig. A Orig. A
5	1 thru 18	Orig.
6	1 thru 7 8 9 thru 24 25 26 thru 55 56, 57 58 thru 63	Orig. A Orig. A Orig. A Orig.
PART 3	Title Page	Orig.
7	1 thru 22	Orig.
PART 4	Title Page	Orig.
8	1 thru 6 7 8 thru 48 49 50 thru 63	Orig. A Orig. A Orig.
9	1 thru 22 23 24 thru 26 27 28 29 30 thru 34 35 36 thru 42 43 44, 45 46, 47 48 thru 57 58, 59 60 thru 63	Orig. A Orig. A Orig. A Orig. A Orig. A Orig. A Orig. A A*
10	1 thru 23	Orig.
11	1 2 3, 4 5 6 7, 8 9 10 11, 12	Orig. A Orig. A Orig. A Orig. A Orig.
Index	1 thru 10 11 thru 13 14, 15	Orig. A Orig.
User Comment Sheet		

Part/Section	Page Number	Update Level

*New pages

All the technical changes are denoted by an arrow (➔) in the margin. A downward pointing arrow (↓) next to a line indicates that technical changes begin at this line and continue until an upward pointing arrow (↑) is found. A horizontal arrow (➔) pointing to a line indicates a technical change in only that line. A horizontal arrow located between two consecutive lines indicates technical changes in both lines or deletions.

Contents

2.2.10.4. Standard System Error Message Interface

An error message service routine provides complete and specific error messages without requiring each system module to contain alphanumeric error information. This routine locates the message in a disk file and transfers control to the system console handler for message display or system logging.

2.2.11. Automatic Volume Recognition

Automatic volume recognition allows the console operator to premount magnetic tapes and disk packs before the devices are required for a job step. This reduces time lost due to job step setup and console responses. The automatic volume recognition function is performed during supervisor initialization and as a result of an attention interrupt being received from an online I/O device. This attention interrupt is caused by physically activating the device online, or, in the case of a device that does not have an attention interrupt capability, by the operator issuing an AVR console command.

Using the physical unit block (PUB) for the devices, automatic volume recognition checks to see if the required tape and disk volumes are already mounted. In addition, it performs special processing to handle unique characteristics of various devices. For example, when required at supervisor initialization, it distinguishes between an 8418 disk pack with high density and an 8418 disk pack with low density or an 8416; it performs special interrupt processing for the 8415 disk; it identifies an 0776 printer configured as an 0770 printer. It then marks the device type in the PUB for that device. It also distinguishes between block numbered and unnumbered tapes. If a tape is not at loadpoint, it rewinds the tape so that it can read the label and the volume serial number.

The automatic volume recognition function displays console messages to the operator to indicate such conditions as a disk or tape not prepped, an I/O error, or a duplicate volume serial number.

A system generation option incorporates a retry on the attention interrupts feature in the AVR function. This permits automatic retry of a recoverable error when an attention interrupt is received on a printer, card reader, or card punch that has an unanswered PIOCS error message. ← The operator can initiate the recovery retry at the device by placing it online, instead of having to return to the console to respond to the error message.

2.2.12. Main Storage Consolidation

Main storage consolidation is a system generation option that repositions jobs and reallocates space in main storage so that enough contiguous space can be made available when needed to hold the next job to be initiated. This reduces fragmentation of main storage and permits a job to be run that requires more contiguous space than is currently available without consolidation.

When a job or a symbiont terminates, the next job to be run is evaluated to determine whether there is enough space available or whether main storage consolidation is necessary and which jobs must be moved. If this job is scheduled and consolidation is required, the jobs are moved down one by one, starting with those farthest from the supervisor. Each job to be moved is brought to an idle state, then moved down. Addresses are adjusted and the job is reactivated. When all these jobs have been moved, the next scheduled job is read in and initiated.

Main storage consolidation does not move symbionts because they do not have an associated relocation register. Nor does main storage consolidation move jobs with open interfaces to the integrated communications access method (ICAM), because these jobs may be reading or writing directly into or out of user main storage. This restriction is minimized if ICAM is loaded first, then ICAM user jobs next, in order to retain the maximum continuous main storage region for further allocation.

2.2.13. Rollout/Rollin

The rollout/rollin function is a system generation option that temporarily transfers jobs from main storage to disk to make room for a job with a preemptive scheduling priority. Jobs currently in main storage are suspended and written to the job's run library. The preemptive job is then read into main storage and initiated. As enough space becomes available, the rolled-out jobs are read back into main storage and allowed to continue processing.

When a job or a symbiont terminates and there is a preemptive job in the job queue, the preemptive job is evaluated to determine whether there is enough existing main storage available, or whether main storage consolidation or rollout is necessary to make space available. If the job is scheduled and rollout is required, the rollout function brings each job marked for rollout to an idle state, delinks the TCBs from the switch list, and writes the job's image from the job region to disk. These rolled-out jobs have asterisks appended to their names on the top line of the display on the system console. If the needed I/O devices are available, the preemptive job is read into the freed main storage and initiated.

As space becomes available and if there are no other preemptive jobs, the job scheduler tries to bring in the rolled-out jobs, one by one. The job slots and I/O devices remain in effect from the time the jobs were rolled out. The job scheduler ignores any jobs on the high- or normal-priority job queues until all of the rolled-out jobs have been rolled back in and reactivated.

2.2.14. Cochanneling

Cochanneling is the capability of accessing a single peripheral device through either of two physical paths. Under OS/3, it provides for the support of both the dual access and dual channel capabilities of the 90/30 hardware.

Dual access cochanneling permits simultaneous I/O operations (read/read, read/write, write/write) on any two devices using two control units and two selector channels. Each input/output device is connected to both control units, one control unit on each selector channel. Depending on the control units used, dual access cochanneling is applicable to

Table 3—1. Supervisor Macro Instructions (Part 2 of 3)

MULTITASKING

Task Management

ECB	Generate an event control block.
ATTACH	Create and activate an additional task.
DETACH	Terminate a task normally.
TYIELD	Deactivate a task.
AWAKE	Reactivate an existing nonactive task.
CHAP	Change the priority of a task.

Task Synchronization

WAIT	Wait for a task request to complete.
WAITM	Wait for one of several task requests to complete.
POST	Activate the waiting task.
TPAUSE	Deactivate one or more tasks other than the issuing task.
TGO	Reactivate one or more tasks other than the issuing task.

PROGRAM MANAGEMENT

Program Loader

LOAD	Load a program phase and return control.
LOADR	Load a program phase, relocate address-constants, and return control.
LOADI	Locate a program phase and store its phase header in a work area.
FETCH	Load a program phase and branch.

Job and Task Termination

EOJ	Terminate a job step normally.
CANCEL	Terminate a job abnormally.

Timer Services

GETIME	Obtain current time and date.
SETIME	Set an elapsed time counter for the requesting task.

Program Linkage

CALL/VCALL	Call a program.
ARGLST	Generate an argument list.
SAVE	Save register contents.
RETURN	Restore registers and return.

Island Code Linkage

STXIT	Link to island code subroutine.
EXIT	Exit from island code subroutine.

Table 3—1. Supervisor Macro Instructions (Part 3 of 3)

PROGRAM MANAGEMENT (cont)

System Information Control

GETCOM	Retrieve data from job communication area.
PUTCOM	Place data into job communication area.
GETINF	Retrieve data from system control tables.

Control Stream Reader

GETCS	Retrieve embedded data file submitted in job control stream.
SETCS	Reset pointer to embedded data file.

DIAGNOSTIC AND DEBUGGING

Storage Displays

SNAP/SNAPF	Print out portions of main storage and return control.
DUMP	Print out the job main storage and terminate the job step.

Checkpoint Facility

CHKPT	Record a checkpoint.
DDCPF	Define a SAT checkpoint file.
DCPOPN	Open a SAT checkpoint file.
DCPCLS	Close a SAT checkpoint file.
DCFLT	Generate a file list table.

Monitor and Trace

// **OPTION TRACE**	Monitor from start of job. (This is a job control statement, not a macro instruction.)

MESSAGE DISPLAY, LOGGING, AND OPERATOR COMMUNICATION

WTL	Write a message into system log file.
WTLD	Write a message into system log file after displaying on system console or workstation.
GETMSG	Retrieve message from canned message file.
OPR	Display a message on system console or workstation.

OTHER SERVICES

Spooling

BRKPT	Create a breakpoint in a spool output file.

4. Physical Input/Output Control System (PIOCS)

4.1. GENERAL

The resident supervisor of OS/3 contains a set of routines called the physical input/output control system (PIOCS) that controls the activity between the processor and all peripheral devices connected to the mutliplexer, selector, and integrated channels. These input/output (I/O) channels operate independently of the processor and allow I/O operations on a channel to overlap with processing and with operations on other I/O channels.

PIOCS:

- schedules I/O requests to maintain optimum I/O throughput without burdening the problem program;
- initiates I/O operations;
- tests for error or other exceptional conditions pertinent to the actual physical transfer of data; and
- activates error recovery procedures in the event of peripheral device errors.

Problem program interface to the IOCS is provided at two levels: data management (logical I/O control system) and PIOCS macro instructions.

Data management routines substantially reduce programming effort, especially for jobs requiring a great amount of I/O processing. The routines, by handling the foregoing I/O functions for the programmer automatically, enable you to concentrate on the logical record, because the applicable PIOCS macro instructions are contained in the data management macro routines and you need only limited knowledge of the peripheral device. The data management macro instructions are described in the data management user guide, UP-8068 (current version).

The use of the PIOCS macro instructions may be advantageous for certain programs, which, because of unique I/O devices, need to control the actual handling of the data to be read or written. To use PIOCS macro instructions, you must have an in-depth knowledge of the particular peripheral device and its control requirements. At the PIOCS level, the problem program is responsible for performing functions such as:

- constructing the actual I/O commands processed by the device as well as constructing the control blocks used by PIOCS for issuing the I/O order;
- ensuring the desired sequence of I/O commands by the proper use of I/O synchronization macro instructions;
- blocking/deblocking logical records;
- alternating I/O buffer areas;
- detecting wrong-length records;
- handling end-of-file (EOF) or end-of-volume (EOV) conditions;
- processing labels;
- translating ASCII data to EBCDIC on input, or EBCDIC data to ASCII on output; and
- handling unique error conditions.

4.2. PHYSICAL I/O CONTROL

4.2.1. General

Detailed tabular information pertaining to each request must be supplied if the problem program is to communicate effectively with the IOCS facilities of the resident supervisor through the PIOCS macro instructions.

The following PIOCS macro instructions are available for establishing the tabular information and for requesting services of the supervisor and the IOCS:

- Table generation macro instructions (declarative)

  BCW
  : Constructs a buffer control word (BCW), which is used by the integrated I/O channels and multiplexer channel.

  CCW
  : Constructs a channel command word (CCW), which is used by the selector I/O channel and the physical device.

  CCB
  : Constructs a command control block (CCB), which is used as a bidirectional communications medium between the problem program and the IOCS routines in the supervisor.

  PIOCB
  : Constructs a physical input/output control block (PIOCB), which is used as a buffer for file control blocks (FCB) containing file and device information that is compiled by job control at the time the job control stream is processed.

- Service request macro instructions (imperative)

RDFCB

Reads a file control block (FCB), which completes the PIOCB with information compiled at job execution time by job control. (The RDFCB macro instruction must be executed prior to any service for an associated PIOCB.)

EXCP

Requests execution of a channel program. The EXCP macro instruction initiates the PIOCS routine. Before this instruction can be executed, you must construct an I/O control packet that consists of a CCB, a CCW or a BCW, and a PIOCB.

The relationship of the basic PIOCS macro instructions is illustrated in Figure 4—1.

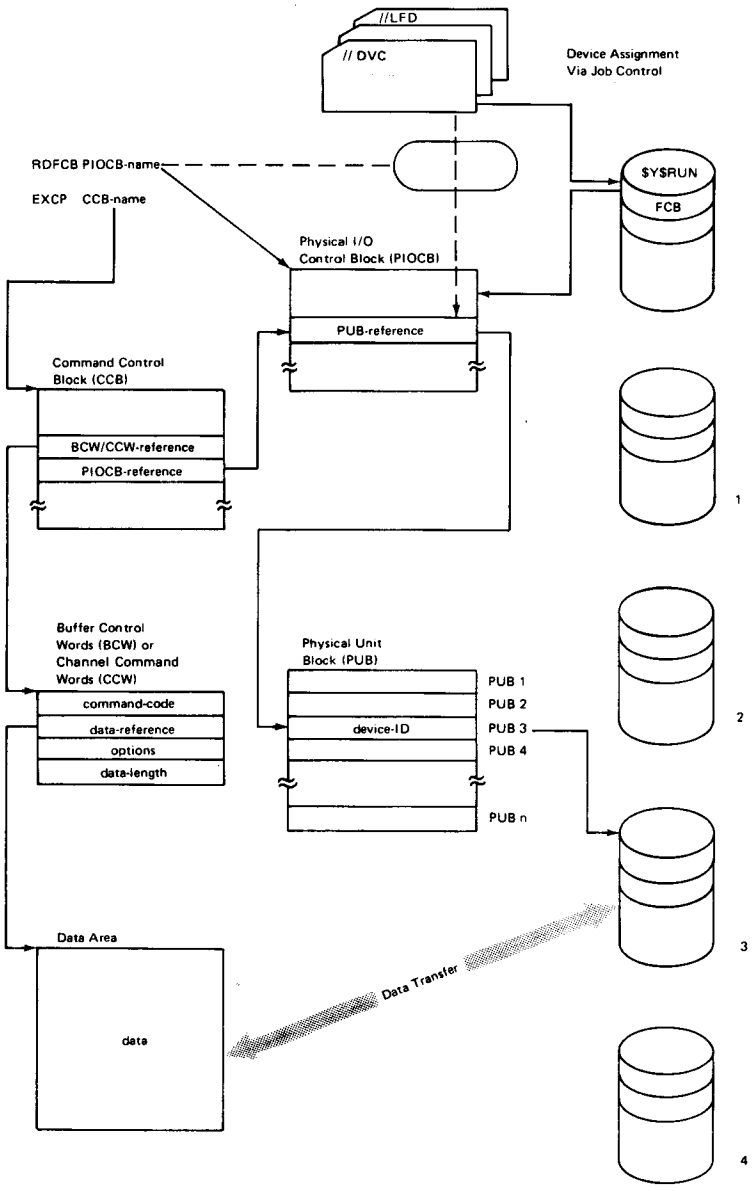


Figure 4—1. Relationship of Basic PIOCS Macro Instructions

4.2.2. General I/O Usage Requirements

The users of I/O facilities are required to perform certain prerequisites for I/O communication. These include:

- Description of the file to the operating system through DVC, LBL, or LFD statements.
- Description of the file to the data management system through file description tables and file control routines.

Description of the file to the operating system is through job control statements which describe the device to be used, the volume which contains the file, and the logical name assigned to the file.

Description of the file to the data management system includes the option of linking to a standard data management file control module, using a resident module, or assembling and/or linking a special tailored module with the user program.

The file description table must be included with the user program.

The macro instructions used in the I/O system are best described at the levels at which they are employed.

- User level macro instructions

  The execution of imperative macros (EXCP, RDFCB, SWAP) results in control being passed to the appropriate control routine within the operating system. You specify the name of the file, which is the name that was assigned to the file control block by an entry in the label field of the PIOCB macro instruction.

  Example:

```
          RDFCB     MASTER
          EXCP      FILEIN
          WAIT      FILEIN
            .
            .
            .
FILEIN    CCB
MASTER    PIOCB
```

The PIOCB declarative macro instruction reserves an area which is the repository of the file control block. The name assigned to the PIOCB must be a duplicate of the character string in the LFD job control statement.

- Data management level macro instructions

  Execution of the imperative I/O macro instructions results in the data management file control routine reducing your macro to a new level of imperative macro instructions. These include the RDFCB (read file control block), the EXCP (execute channel program), and the WAIT (wait for channel program completion) macro instructions.

  The primary parameter to the EXCP and WAIT macro instructions is the CCB. The CCB macro provides the ability to specify a particular command to a particular device.

4.2.3. Generate Buffer Control Word (BCW)

Function:

The BCW macro instruction generates a buffer control word which provides the hardware parameter interface to the integrated disk adapter, integrated peripheral channel, multiplexer channel, and the integrated line adapters for use by the PIOCS routines. Also, the BCW macro instruction provides you with a limited device-independent interface across selector channel devices. In this case, the PIOCS routines construct a CCW chain by using the information provided in the BCW. The formats of the BCW are shown in Figures 4—2, 4—3, and 4—4.

Note that the BCW of formatting commands sent to the 8411 and 8414 disk subsystems must specify a single record.

This is a declarative macro instruction and must not appear in a sequence of executable code.

Format:

LABEL	ΔOPERATIONΔ	OPERAND
symbol	**BCW**	**device-cmd-code[,data-addr] [,data-flag] [,data-byte-count] [,repl-addr] [,repl-flag] [,repl-byte-count] [,control-flag]**

Label:

symbol
Specifies the symbolic address of the buffer control word. This name is used to refer to the BCW.

Positional Parameter 1:

device-cmd-code
Specifies the actual device command code that directs the operation of the I/O device. (For a complete description of the command codes for a particular device, refer to the appropriate subsystem programmer reference manual.)

If omitted, 16 bytes containing 0's are reserved for the BCW, and the assembly listing will contain an error note.

Positional Parameter 2:

data-addr
Specifies the symbolic address of the data being transferred. This is the active buffer for the system console and the integrated line adapters. For the read/punch, it is the address of the punch output buffer. This parameter is required if data is being transferred to or from storage.

If omitted, the data address field in the BCW is set to 0's, and the assembly listing will contain an error note.

Positional Parameter 3:

data-flag
Specifies the flag byte associated with the address of the active buffer. This is written in the form X'xx' as follows:

For the integrated disk adapter:

X'40' Indicates a search operation is to be performed on an entire cylinder rather than a track.

X'80' Indicates no data to be transferred.

For the integrated peripheral channel:

X'20' Indicates no data to be transferred. (This entry can also be used for the multiplexer channel.)

X'80' Indicates a replacement operation is to be performed. If this entry is used, positional parameters 5, 6, and 7 are also required.

If omitted, X'00' is assumed, indicating normal operation as specified by the device command code, data address, and data byte count fields in the BCW.

Positional Parameter 4:

data-byte-count
Specifies the number of bytes to be transferred or the number of sectors to be transferred for a sectored IDA device.

If omitted, zero is assumed. For a search on the integrated disk adapter, this indicates the maximum number of bytes or sectors are to be transferred; and for a read or a write, this indicates no data is to be transferred. For the integrated peripheral channel, this indicates the maximum number of bytes are to be transferred.

NOTE:

Positional parameters 5, 6, 7, and 8 apply only to the integrated peripheral channel.

Explanations:

1. Read one 80-column card in EBCDIC mode.

2. Read/punch 80-column card in EBCDIC mode. Punch buffer is IOAREA1; read buffer is IOAREA2.

3. Print 132 positions and advance one line.

4. Read one sector on 8416/18 disk.

NOTE:

The cylinder half word (BCW name+12), the head address byte (BCW name+10), and the record (sector) number byte (BCW name+14) can be set statically by use of the ORG assembler control directive, or dynamically via instruction execution.

4.2.4. Generate Channel Command Word (CCW)

Function:

The CCW macro instruction generates a channel command word which provides the hardware parameter interface to the selector channels for use by the PIOCS routines. The format of the CCW is shown in Figure 4—5. The format of the CAW, which contains the first CCW address, is shown in Figure 4—6.

The supervisor can only handle command chains on selector devices through two levels of transfer in channel (TIC) within command chain. This limitation is due to the lack of hardware address relocation on CCWs and the need to have a software function perform the absolutizing and relativizing of CCW addresses.

This is a declarative macro instruction and must not appear in a sequence of executable code.

Format:

LABEL	Δ OPERATION Δ	OPERAND
symbol	**CCW**	**[device-cmd-code] [,data-addr] [,flag] [,data-byte-count]**

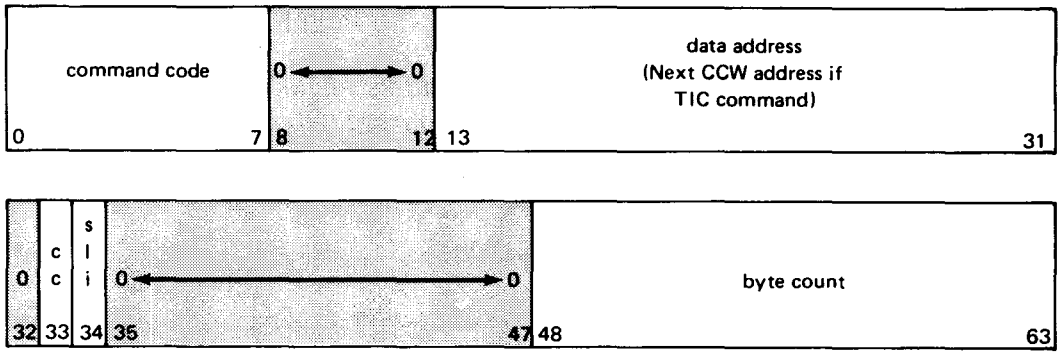
Label:

symbol
Specifies the symbolic address of the channel command word. This name is used to refer to the CCW.

Positional Parameter 1:

device-cmd-code
Specifies the actual device command code that directs the operation of the I/O device. (For a complete description of the command codes for a particular device, refer to the appropriate subsystem programmer reference manual.)

If omitted, eight bytes containing 0's are reserved for the CCW, and the assembly listing will contain an error note.



Bits	Allocation	Function
0–7	Command code	Specifies operation to be performed by device and channel
8–12		Unassigned; must be set to zero
13–31	Data address	Address of location in main storage into or from which first byte of data is transferred
32		Unassigned; must be set to zero
33	cc (chain command flag)	When valid ending device status received, new CCW fetched and operation specified by new command code initiated
34	sli (suppress length indication flag)	If set to 1, incorrect length condition not indicated to program; if cc = 1 also, command chaining not suppressed
35–47		Unassigned; must be set to zero
48–63	Byte count	Byte count required for all data transfer operations

LEGEND:

System-supplied data
Data supplied by the user via the macro instruction that directs the supervisor to generate the control block

Figure 4–5. Channel Command Word (CCW) Format for Selector Channel

Label:

symbol
Specifies the symbolic address of the command control block. This name is used to refer to the CCB.

Positional Parameter 1:

PIOCB-name
Specifies the symbolic address of an associated physical input/output control block generated by the PIOCB macro instruction. (The address furnished will be modified by this macro instruction to be the address of the PUB address within the PIOCB.)

Positional Parameter 2:

BCW-name
Specifies the symbolic address of a BCW.

CCW-name
Specifies the symbolic address of a CCW, or a list of CCWs if command chaining is used.

When you use data management macro instructions, the BCWs and CCWs are generated automatically. When using PIOCS macro instructions, you must specify each BCW and CCW according to the I/O functions desired.

Positional Parameter 3:

PUB-entry-number
May be 0, 2, 4, 6, 8, 10, 12, or 14 indicating one of eight 2-byte fields in the PIOCB containing the absolute address of the PUB for the device involved in the I/O operation. (Zero indicates the first entry, 2 the second, 4 the third, etc.)

If omitted, zero is assumed (indicating the first PUB address).

Positional Parameter 4:

error-option

Specifies error acceptance options elected at assembly time. This is written in the form X'xx' as follows:

X'00'	Indicates that no error conditions are acceptable to the problem program.
X'01'	Block number area is reserved in buffer.
X'02'	Reserved for system use.
X'04'	Reserved for system use.
X'08'	Indicates system access CCB. Device independence can be achieved by furnishing a BCW for an integrated peripheral.
X'10'	Indicates a diagnostic request. Reserved for system use.
X'20'	Indicates that, following the normal error recovery attempts by the supervisor, those errors classified as unique are acceptable to the problem program. See note 1.
X'40'	Indicates that all unrecoverable error conditions are acceptable to the problem program following the normal error recovery attempts by the supervisor. See note 2.
X'80'	Indicates user has own error code. No recovery will be attempted by the supervisor, and device status and sense are communicated to the user in the CCB.

NOTES:

1. *Accept Unique Errors (byte 3, bit 2). Unique errors may be considered as recoverable errors. The meaning of unique errors is different for different devices.*

   For a disk, unique error means record not found. Your program may expect that certain records you are looking for in a file may not be there. An example of this is an update-add program. If the record is found, it is updated; if it is not found, it is added to the file. In this case, you should set the accept unique errors bit (byte 3, bit 2) in the CCB. If you receive a no record found condition (byte 2, bit 3), PIOCS will retry the error twice. If the record is still not found and the CCB is marked to accept unique errors, no error message is displayed on the console and control is returned to your program with the no record found bit set in the CCB.

   For tape, unique error means a tape that is busy rewinding. If you issue an EXCP to a magnetic tape which is rewinding, the CCB will be returned with the unique error bit (byte 2, bit 2) set. This occurs whether or not accept unique errors is set in the CCB. The EXCP should be reexecuted until the status does not occur. At that time, the EXCP is considered completed.

For printers, unique error means character mismatch. This means that there is no match between a code in the load code buffer (LCB) and a character in the print line buffer. When you generate the LCB for your printer, you may choose whether or not to report character mismatches. If you choose not to report character mismatches, they will be ignored and no console error message will be displayed. If the LCB is generated so that character mismatches are to be reported and a character mismatch occurs, an error message will be displayed on the console. If the accept unique errors bit is set in the CCB, the options on the error message will be R (retry) or I (ignore).

If the operator responds I, control will be returned to your program with the unique error bit set in the CCB. If the accept unique errors bit is not set, the options on the error message will be R (retry) or C (cancel).

There are no unique errors for readers and punches. Note that, except for tape, if a unique error occurs and the CCB Is not marked to accept unique errors, PIOCS will treat the unique error as an unrecoverable error.

2. *Accept Unrecoverable Errors (byte 3, bit 1). If you set this bit in the CCB and an unrecoverable error occurs, the console message will appear with the R (retry) and U (accept unrecoverable) options. If the operator responds R, the command will be retried. If the operator responds U, control will be returned inline following the command, and the unrecoverable error bit (byte 2, bit 1) will be set in the CCB. If you do not set the accept unrecoverable bit in the CCB and an unrecoverable error occurs, the console message will appear with the R (retry) and the C (cancel) options. After successive retries, if the error still is unrecoverable, the operator may choose to respond C and the job will be cancelled.*

If omitted, the entry X'00' is assumed, indicating that no error conditions are acceptable to the problem program.

The CCB is used to communicate with the functional IOCS routines executing the I/O operations. The generated CCB forms the logical connection between the PIOCB and the CCW or the BCW. The PIOCB references the actual peripheral device and the CCW or the BCW defines and controls the function of the particular device and its data transfer. The CCB also specifies user options pertinent to the I/O request in the event of an error, and reflects the status of the request. When the related I/O interrupt occurs, the IOCS also stores status information pertinent to the interruption in the associated CCB.

Because the CCB serves as a 2-way communications medium between the IOCS and the problem program, it is used for one active I/O request at a time; therefore, every active I/O request must have a unique CCB.

Byte	0	1	2	3
0	control byte 1	I/O error count	transmission byte	control byte 2
4	TCB address① or next CCW address			
8	CCB link	address② or residual CCW byte count		
12	CCW address			
16	PIOCB pointer (PUB address)			
20	sense byte 0	sense byte 1	sense byte 2	sense byte 3
24	sense byte 4	sense byte 5	device status	channel status

NOTES:

→ ① During the I/O command execution, bytes 4—7 contain the address of the TCB associated with this CCB. At I/O command termination, PIOCS inserts the address of the next CCW in the chain.

→ ② During I/O command execution, bytes 8—11 contain the address of the next CCB in the chain at this job level. At I/O command termination, PIOCS inserts the number of bytes remaining in the CCW byte count (when the I/O command terminated) into bytes 10 and 11.

Figure 4—7. Command Control Block (CCB) Format (Part 1 of 2)

The EXCP macro instruction communicates directly with the I/O scheduler for the purpose of submitting I/O requests to the system. Before the EXCP macro instruction is executed, you must construct an I/O packet consisting of the following:

- Use a CCB macro instruction to define the CCB.
- Use a PIOCB macro instruction to define the physical I/O control block.
- Use one or more CCW macro instructions or a BCW to construct the channel program.
- Use an RDFCB macro instruction to identify the I/O device and to obtain file information specified by job control.

Linkage between these components is as follows:

- The EXCP macro instruction passes the address of the CCB to the PIOCS routines. ←
- The address of a 2-byte field in a physical I/O control block is stored in the CCB. This field contains the address of the PUB for the peripheral device concerned. ←
- The address of the first CCW or BCW is stored in the CCB.
- Each CCW or BCW contains the address of an input/output data area.

Whenever an EXCP macro instruction is executed, the I/O request counter in the task control block of the requester is incremented and a status indicator in the CCB is set to signify that the order is outstanding. Control is returned to the calling program immediately by the supervisor with the degree of completion of this I/O order uncertain. You must use the WAIT or WAITM macro instruction for synchronization with this I/O.

An EXCP issued to a magnetic tape which is rewinding will result in the posting of the CCB with unique error status. The EXCP should be reexecuted until the status does not occur. At that time the EXCP is considered completed.

4.3. INPUT/OUTPUT SYNCHRONIZATION

Macros are available that provide the means by which a task can await the completion of one or more outstanding I/O operations. Specifically the task can await one, several, or all outstanding I/Os; however, the I/O being waited for must have been requested by the task doing the waiting.

Tasks are waited by setting a unique wait bit within that task control block (TCB). These wait bits signal the switcher that this task is nondispatchable and indicate the reason for the wait. Upon clearing the wait bits, the task becomes dispatchable and can be activated.

Two macro instructions are available for I/O synchronization:

- WAIT

  Wait for one or all I/O requests to complete.

- WAITM

  Wait for one of several I/O requests to complete.

These macro instructions can also be used (with different parameters) to synchronize a task with the execution of other tasks. For I/O synchronization, the macro instruction references a CCB; and for task synchronization, the macro instruction references an event control block. Task synchronization is described in 7.4.

It must be remembered when you use these macro instructions that only the task having executed an I/O request can await its completion; and when you await a task, it is not valid to await the executing task.

4.4.2. Tape Restrictions

The 3-byte block number fields are added to standard labels on block numbered tapes. The three bytes precede the label identifier (VOL1, HDR1, etc.) making the label 83 bytes long. This is true for tapes written in ASCII as well as EBCDIC. Note that in the case of ASCII tapes, the 83-byte label is nonstandard. It can be used for internal processing, but cannot be used for information interchange. Block number processing will be exactly the same for both EBCDIC and ASCII tape files. Tape label formats for block numbered EBCDIC tapes are shown in Figures 6—17 through 6—21.

Block numbers will be volume dependent and file independent. Files on a volume and volumes in a multivolume file must be all numbered or all unnumbered, not mixed.

Block number processing is available for magnetic tapes on selector or multiplexer channels. These may be 9-track tapes, or 7-track odd parity tapes operating in data conversion mode. Block size of 7-track tapes operating in data conversion mode must be a multiple of 3.

4.4.3. Input/Output Buffer

When processing block numbered tapes you must reserve a 4-byte storage area immediately preceding your input/output area for supervisor processing of the block number. This 4-byte block number area, and the input/output area, must be aligned on a full-word boundary. Do not include these four bytes in either the location or the block size of the input/output area.

Block numbers will be checked when reading in either direction. When reading backward, you must be sure your input/output area is large enough to hold the entire block of data. If the data is truncated on a backward read, the block number will be lost and incorrect positioning of the tape may result.

4.4.4. Processing

A number of software components are affected by block number processing; these include system generation, tape preparation, job control, automatic volume recognition, PIOCS, data management, and system access technique (SAT) on tape files. Several control tables in main storage are also affected, including the systems information block (SIB), the device PUB trailer, and the CCB. These tables contain fields that are updated and bits that are set, tested, and cleared to reflect user options and processing events.

PIOCS will perform block number processing for data management, tape SAT, and EXCP-level PIOCS users. A general description of required and optional parameters and processing performed is contained on the following pages. Details pertinent to PIOCS users are contained in 4.4.5. Details of the requirement for tape SAT are contained in 6.5 to 6.10 of this manual. For data management details, refer to the data management user guide, UP-8068 (current version).

The supervisor must be configured to process block numbered tapes, in which case, the generated supervisor can process both numbered and unnumbered tapes. A bit in the SIB is set to indicate that the supervisor supports block numbering. If the supervisor does not have the block numbering capability, only unnumbered tapes can be processed; otherwise, misalignment and possible truncation of data will result because of the block number field.

To use the block numbering capability of the supervisor, you must also reserve a 4-byte storage area, aligned on a full-word boundary, immediately preceding the input/output area. If you are a data management user, you indicate that you have reserved this 4-byte area by using the BKNO=YES parameter in the DTFMT macro instruction. If you are a tape SAT user, you do this by using the BKNO=YES parameter in the TCA macro instruction. If you are a PIOCS user at the EXCP level, you must also indicate that you have reserved the 4-byte area by setting a bit in the CCB (4.4.5).

You have the option not to use block number processing even though the supervisor has the capability and you have indicated there is a block number field preceding the input/output area. If you enter N as the first parameter in the VOL job control statement, block numbers will not be written on output tapes and will be ignored if present on input tapes.

Automatic volume recognition will read and store volume serial numbers and will set appropriate bits in the PUB trailer to indicate whether or not it is processing standard labeled tapes and block numbered tapes.

The PUB trailer for a block numbered tape file will contain an expected block number. This number will reflect the next block number anticipated in a forward read and will be adjusted accordingly for backward reads. When the tape is read in either direction, the block number read from tape is stored in the PUB trailer and compared with the expected block number. If there is no discrepancy (and no other errors), control is returned to the user program. If there is a discrepancy, PIOCS attempts to find the correct block by moving the tape backward or forward the number of blocks implied by the discrepancy. If the correct block is found, control is returned to the user. If the correct block cannot be found, the tape is left positioned where it was on the last attempt and an error message is sent to the console.

When processing control macros, block number processing will not be performed, because no data transfer is involved. However, for commands involving single blocks (FSB, BSB), the block number count will be updated.

On block numbered tapes, CCW chains with more than one tape movement command and multiblock BCW commands can be processed only through the first tape movement command.

4.4.5. PIOCS Requirements and Options

PIOCS users at the EXCP level have an additional requirement. Before issuing any EXCP macro instruction for a block numbered tape, you must set byte 3, bit 7, in the CCB. This indicates that the 4-byte block number field preceding the input/output buffer has been reserved. If this bit is not set, the job will be cancelled.

You can request that block numbering be ignored on input tapes by setting byte 0, bit 3, in the CCB before issuing an EXCP. In this case, block numbered tapes will be read, but the block numbers will not be verified. You must set this bit each time you want to ignore block number processing on a read.

Another option available at the PIOCS level is to accept unrecoverable errors. You can do this by setting byte 3, bit 1, in the CCB. You don't have to reset this bit for each EXCP; it need only be set once and stays set.

On a read, if PIOCS detects a variance between the expected block number and the actual block number and is unable to resolve this variance after 10 retries, a console message is issued. If byte 3, bit 1 (accept unrecoverable errors) is set, the console message gives the operator opportunity to request a retry or accept the error. If retry is requested but is still unsuccessful, the operator will again be asked to request a retry or accept the error. If he accepts the error (or if he first requests retry and it is still unsuccessful), PIOCS sets byte 2, bit 1 (unrecoverable error). PIOCS then sets byte 2, bit 0, to indicate that CCB processing is complete and returns control to your program. On input, you should test byte 2, bit 5, after the WAIT is executed to ensure that the correct block has been processed.

If byte 3, bit 1 (accept unrecoverable errors) was not set, the operator has the option only to request a retry or cancel. If retry is requested but is still unsuccessful, the operator will again be asked to request a retry or cancel.

On a write, if byte 3, bit 1, is set and the tape cannot be positioned correctly, a console message gives the operator the opportunity to accept the error or cancel. If this bit is not set, he must cancel.

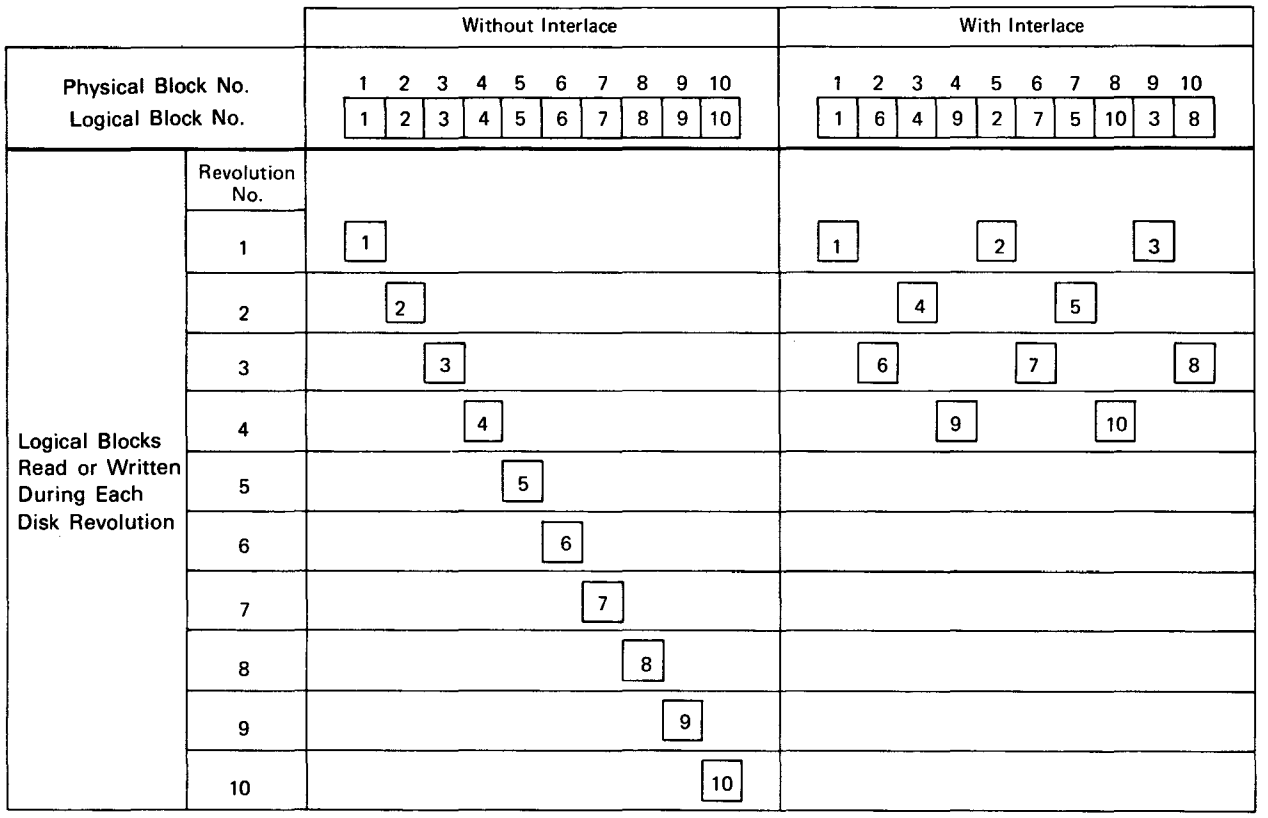


Figure 6—4. Interlace Accessing

Successful interlace operation requires that the I/O orders must be issued within a specific time frame. The lace factor, therefore, determines how blocks are to be spaced on the track to ensure that the actual time frame (which includes both user and SAT overhead) is equal to or greater than your estimate of required time between block accesses.

A lace factor of 4 means that the blocks will be spaced in sufficient intervals (every 4th block) to produce an actual time frame that is equal to or greater than the estimated required time frame.

To calculate the lace factor, use the formula described in 6.2.5.2. Although the formula is based on the use of the 8416 disk subsystem, all lace factor calculations must be performed by using this formula, regardless of the actual disk subsystem being used. When the file is opened by the OPEN macro instruction, the specified lace factor will be applied to the performance of the particular disk subsystem being accessed. If necessary, SAT will adjust the lace factor to the capacity and speed of the specific device so that a similar time frame will be maintained for interlaced files processed on all supported disk subsystems.

6.2.5.2. Lace Factor Calculation

The lace factor is calculated in two steps by using the following formula:

1. $\frac{\text{BLKSIZE}}{256}$ x .535 = Calculated Sector Time

2. $\frac{\text{Required Time Frame}}{\text{Calculated Sector Time}}$ + 1 (rounded high) = Lace Factor

For example, if you are using a block size of 1024 bytes, first calculate the sector time in milliseconds:

1. $\frac{1024}{256}$ x .535 = 2.14 ms

Then calculate the lace factor using an estimate of the processing time required between block accesses. For this example, let us use a required time frame estimate of 7.48 ms:

2. $\frac{7.48}{2.14}$ = 3.49 + 1 = 4.49 rounded to 4

The result is a lace factor of 4. In the PCA macro instruction statement for this partition, enter the keyword parameter LACE=4.

NOTE:

When the time frame exceeds 21.4 ms, it should be divided by 21.4 and the remainder should be used as the time frame in the foregoing calculation.

6.2.6. Accessing Multiple Blocks

When you are engaged in sequential processing (SEQ=YES specified in PCA macro instruction), you can read or write more than one block with each SAT imperative macro instruction that is issued. This is done by specifying the number of blocks you wish to access together by using the LBLK keyword parameter of the PCA macro instruction. However, when you use multiple buffer accessing, be certain that your I/O buffer area has enough contiguous space to contain the blocks. Also, if you are creating the partition by using the format write option, (FORMAT=NO), an additional 8-byte area, used to construct the count field, must immediately precede the first buffer area. During input operations, fewer than the requested number of blocks may be read if the end of data ID is encountered. The I/O count field (bytes 44 and 45) of the DTF (Figure 6—5) will contain the number of buffers not acted upon.

Normally, SAT makes a single reference to PIOCS for the number of blocks requested. If an end-of-track condition is enccuntered for any block other than the last block of the request, SAT makes an additional reference to PIOCS to access the next track. For interlaced files, SAT makes one reference to PIOCS for each block requested. If an end-of-block condition is encountered on the last, or only, block requested, an information bit will be set in the error status field (byte 50, bit 0, of the DTF) to indicate the last block on that track has been accessed.

The LBLK keyword parameter specifies the number of blocks required, within a range from 1 to 255; however, the total size of the buffer cannot exceed 32,767 bytes.

Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	CLOSE	{ filename-1 [,...,filename-n] / (1) / *ALL }

Positional Parameter 1:

filename-1
Specifies the symbolic address of the DTFPF macro instruction in the program corresponding to the file to be closed.

(1)
Indicates that register 1 has been preloaded with the address of the DTFPF macro instruction.

***ALL**
Specifies that all files currently open in the job step are to be closed.

Positional Parameter n:

filename-n
Successive entries specify the symbolic addresses of the DTFPF macro instructions in the program corresponding to the additional files to be closed.

6.5. SAT FOR TAPE FILES

The OS/3 tape system access technique (TSAT) is a generalized input/output control system that provides a standard interface to PIOCS for magnetic tape subsystems. It performs the basic functions of a tape data management system and provides block level I/O for sequential tape files. ←

Interface with TSAT files is through declarative and imperative macro instructions. You use the SAT and TCA declarative macro instructions to define the characteristics of the file and the data management technique to be used to process the file. The SAT macro instruction creates the DTF table for the file, and the TCA macro instruction creates the appendage to the table. These macro instructions are described in 6.8. You use the OPEN, GET, PUT, CNTRL, WAITF, and CLOSE imperative macro instructions to control file processing. These are described in 6.9.

All files processed by TSAT are written in a forward direction, and can be read forward and backward. The CNTRL macro instruction initiates nondata operations on the device and can be issued whether or not the file is open.

To use TSAT, you must observe tape label conventions (described in 6.6) and tape volume and file organization conventions (described in 6.7).

If you are processing block numbered tapes, you must also observe the special conventions applicable to these tapes. Requirements and processing for block numbered tapes are summarized in 6.10.

6.6. SYSTEM STANDARD TAPE LABELS

Magnetic tapes may be labeled or unlabeled, and a labeled tape may contain either standard or nonstandard labels. You indicate this using the FILABL parameter in the TCA macro instruction. TSAT assumes that tapes have standard labels. If nonstandard labels exist on input files, TSAT bypasses them.

All standard tape labels are in blocks of 80 bytes and are always recorded at the same density as the data. The first three bytes of each label identify the type, and fourth byte indicates its position within the group. For example, VOL1 indicates this is the first volume label for this file.

For block numbered tapes, each label includes a 3-byte block number field as the first three bytes of the label, making the label 83 bytes long.

There are five tape label groups; three are required and two are optional. The tape label groups are:

- Volume label group — VOL
- File header label group — HDR
- User header label group (optional) — UHL
- File trailer label group — EOF or EOV
- User trailer label group (optional) — UTL

TSAT does not process user header (UHL) or user trailer (UTL) labels. No provision is made for creating these labels on output files; if they exist on input files, TSAT bypasses them.

TSAT label processing is limited to one volume label (VOL1), two file header labels (HDR1 and HDR2), and two file trailer labels (EOF1 and EOF2 or EOV1 and EOV2). No provision is made for creating additional labels on output files; if they exist on input files, TSAT bypasses them.

Tape label formats for block numbered files are shown in Figures 6—17 through 6—21. Tape label formats for files without block numbers are shown in Figures 6—6 through 6—10 and are described on the following pages.

Positional Parameter 1:

filename
Specifies the symbolic address of the corresponding SAT macro instruction in the program.

(1)
Indicates that register 1 has been preloaded with the address of the SAT macro instruction.

Positional Parameter 2:

code
Is a mnemonic 3-character code specifying the tape unit function to be performed:

BSF	Backspace to tape mark*
BSR	Backspace to interrecord gap*
ERG	Erase gap (writes blank tape)
FSF	Forward space to tape mark*
FSR	Forward space to interrecord gap*
REW	Rewind tape
RUN	Rewind tape with interlock (unloads tape)
WTM	Write tape mark

6.9.6. Close a Tape File (CLOSE)

Function:

The CLOSE macro instruction performs the required termination operations for a file; for example, construction of the EOF label group. Once the CLOSE macro instruction has been issued for a file, only the OPEN macro instruction may reference that file.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	CLOSE	{filename-1[,...,filename-n] / (1)}

Applies only to input files.

Positional Parameter 1:

filename-1
Specifies the symbolic address of the SAT macro instruction in the program corresponding to the file to be closed.

(1)
Indicates that register 1 has been preloaded with the address of the SAT macro instruction.

Positional Parameter n:

filename-n
Successive entries specify the symbolic addresses of the SAT macro instructions in the program corresponding to the additional files to be closed.

6.10. BLOCK NUMBER PROCESSING

TSAT can process magnetic tapes with or without block numbers. The use of block numbers reduces the possibility of incorrect tape positioning and, therefore, incorrect tape processing. This is especially helpful for error recovery on read and write commands and for restarting at a checkpoint.

→ Processing of block numbered tapes for TSAT files will be executed by PIOCS. The general requirements and processing are the same as detailed for PIOCS in 4.4.1 to 4.4.4. Some of these are noted here for convenience.

- When the block numbering capability is being used, all blocks on tape except tape marks will include a 3-byte block number field as the first three bytes of the block. This 24-bit block number field is composed of a 4-bit tape mark counter and a 20-bit block number → counter. PIOCS uses both of these counters when reading and writing block numbered tapes. The format of the tape block number field is shown in Figure 4—9.
- The first block on tape that is not a tape mark will contain a block count of 1 plus the number of tape marks preceding it.
- Block numbers are incremented sequentially by 1. All label, data, and checkpoint blocks are counted and numbered. Tape marks are counted, but no number is written.
- For both EBCDIC and ASCII tapes, the 3-byte block number field is added to a standard label immediately preceding the label identifier (VOL1, HDR1, etc.), making the label 83 bytes long. The 83-byte ASCII label is nonstandard for information interchange. Tape label formats for block numbered EBCDIC tapes are shown in Figures 6—17 through 6—21.
- Block number processing will be exactly the same for both EBCDIC and ASCII tape files.
- Block numbers will be volume dependent and file independent. If a volume contains more than one file, the block count is continued from the preceding file on the volume and the blocks are consecutively numbered to the end of the tape.

- Files on a volume and volumes in a multivolume file must be all numbered or all unnumbered, not mixed.

- The 7-track odd parity tapes operating in convert mode may be block numbered if the block size is a multiple of 3.

The PUB trailer for a block numbered tape file will contain an expected block number. This number will reflect the next block number anticipated in a forward read and will be adjusted accordingly for backward reads. When the tape is read in either direction, the block number read from tape is stored in the PUB trailer and compared with the expected block number. If there is no discrepancy (and no other errors) control is returned to the user program. If there is a discrepancy, PIOCS attempts to find the correct block by moving the tape backward or forward ← the number of blocks implied by the discrepancy. If the correct block is found, control is returned to the user. If the correct block cannot be found, the tape is left positioned where it was on the last attempt and an error message is sent to the console.

6.10.1. Facilities Required for Block Number Processing

To process block numbered tape files, three conditions (called preliminary conditions) are required:

1. So that the generated supervisor can process both numbered and unnumbered tapes, you must operate with a supervisor configured to process block numbered tapes.

2. You must reserve a full-word aligned, 4-byte storage area immediately preceding your input/output area for supervisor processing of the block number. Do not include these four bytes as part of either the address or the length specifications (IOAREA and BLKSIZE keyword parameters of the TCA declarative macro instruction).

3. You must indicate to TSAT that you have reserved the 4-byte block number area by specifying BKNO=YES in the TCA macro instruction (6.8.2).

If these three preliminary conditions exist, you may then control block number processing through either job control (JCL) or automatic volume recognition (AVR). This permits you to leave the 4-byte storage area and the BKNO parameter in your program even though you may at times be processing unnumbered tapes.

6.10.2. Specifications for Block Number Processing

Several factors determine when and how block number processing is employed. If a tape is not at load point when the file is opened, the file will be handled according to the specifications existing when the tape was opened at load point. Therefore, you cannot have both numbered and unnumbered files on the same volume.

If a tape is at load point when it is opened, processing will proceed as described on the following pages.

The various methods of tape file processing can be divided into two categories: processing with tape initialization, and processing without tape initialization. These will be referred to simply as initialized or noninitialized processing.

6.10.2.1. Initialized Processing

Initialized processing includes:

- TPREP utility routine processing, described in the system service programs user guide, UP-8062 (current version);
- processing output files with standard labels (FILABL=STD specified in the TCA macro instruction) and PREP specified in the VOL job control statement; or
- processing input or output files with nonstandard labels (FILABL=NSTD) or no labels (FILABL=NO specified in the TCA macro instruction).

For initialized processing, you control the presence or absence and the processing of block numbers by the first parameter of the VOL job control statement as follows:

You Specify	Preliminary Conditions	Result
Nothing	All present	Block number processing
	Some missing	No block number processing
N	Ignored	No block number processing

6.10.2.2. Noninitialized Processing

Noninitialized processing includes:

- processing output files with standard labels (FILABL=STD specified in the TCA macro instruction), but without PREP specified in the VOL job control statement; or
- processing input files with standard labels (FILABL=STD specified in the TCA macro instruction).

For noninitialized processing, TSAT ignores the first parameter of the VOL job control statement. Instead, the specification is obtained from the tape content (which was detected by AVR), as follows:

Tape Content	Preliminary Conditions	Result
Block numbers	All present	Block number processing
	Some missing	No block number processing
No block numbers	Ignored	No block number processing

For processing of multivolume files, you must ensure that all volumes have (or do not have) block numbers. You cannot mix numbered and unnumbered volumes within a file.

Positional Parameter 5:

DA
Specifies that the 8-byte phase name specified in positional parameter 1 will be overwritten with a read pointer during the first execution of this macro instruction. This read pointer is used to find the phase on the second and all subsequent executions of this macro instruction.

If omitted, a search is performed on the phase name specified in positional parameter 1 each time this macro instruction is executed, and the 8-byte phase name is not overwritten.

8.2.7. Load a Program Phase and Relocate (LOADR)

Function:

The LOADR macro instruction locates a program phase in a load library on disc, loads it into main storage, and transfers control to the calling program immediately following the LOADR macro instruction.

After execution of this macro instruction, register 0 contains the job-relative address at which the phase was loaded, and register 1 contains the job-relative entry-point address. This entry point address is determined at linkage edit time. If an alternate load address is provided (positional parameter 2), the load point address specified to the linkage editor is overridden and the phase is loaded at the specified address. This new override address is returned in register 0.

The format and operation of the macro instruction is identical to the LOAD macro instruction except that all address constants in the phase are relocated if an alternate load address is specified (positional parameter 2).

This macro instruction is used to load a phase at an address other than that at which it was linked.

Format:

LABEL	Δ OPERATION Δ	OPERAND
[symbol]	LOADR	{phase-name / (1)}[,{load-addr / (0)}][,{error-addr / (r)}][,R][,DA] ←

Positional Parameter 1:

phase-name
Specifies the name of the program phase to be loaded. This may be either the 1- to 6-character user-assigned alias phase name or the 8-character linker-assigned phase name in the format nnnnnnpp where nnnnnn is the program name and pp is the phase number.

(1)

Indicates that register 1 has been preloaded with the address of the 8-character phase name.

Positional Parameter 2:

load-addr

Specifies the symbolic address at which the phase is to be loaded.

(0)

Specifies that register 0 has been preloaded with the load address.

If omitted, the program phase will be loaded at the address specified by the linkage editor.

Positional Parameter 3:

error-addr

Specifies the symbolic address of an error routine that is to be executed if a load error occurs.

(r)

Specifies that the designated register (other than 0 or 1) contains the address of the error routine.

If omitted, the calling task will be abnormally terminated if a load error occurs.

Positional Parameter 4:

R

Specifies that only the system load library is to be searched for the phase.

If omitted, a full search is to be performed (8.2.3).

Positional Parameter 5:

DA

Specifies that the 8-byte phase name specified in positional parameter 1 will be overwritten with a read pointer during the first execution of this macro instruction. This read pointer is used to find the phase on the second and all subsequent executions of this macro instruction.

This option is designed to reduce the search time for separately linked load modules which are loaded repeatedly. When using this option, you must ensure that there is no possibility of another job deleting or moving the load module you are trying to load. For example, if another job uses the librarian to pack the library, this may cause a load error in your job. If you can be sure this doesn't happen, you may be able to reduce considerably the load time for some modules, particularly in large libraries.

If omitted, a search is performed on the phase name specified in positional parameter 1 each time this macro instruction is executed, and the 8-byte phase name is not overwritten.

```
                LABEL    OPERATION     OPERAND          COMMENTS
                1        10            16               72
19.             *                      ERROR IF COMPUTATION NOT DONE BEFORE TIME ELAPSES
20.             TOOLONG  EQU           *
21.                      STXIT         IT               DETACH ISLAND CODE
22.             *                      PRINT ERROR MESSAGES, ETC
23.                      .                              } error print routine
24.                      .                              }
25.                      .                              }
26.             *                      TIMER ISLAND CODE - ACTIVATED WHEN TIME ELAPSES
27.             ILANDCOD EQU           *
28.                      OI            FLAGBYTE,TIMEFLAG SET FLAG
29.                      EXIT          IT
30.             *                      WORK AREAS
31.             ICSAVE   DS            18F              REGISTER SAVE AREA REQUIRED
32.             FLAGBYTE DC            X'00'            INITIALLY ZERO
33.             TIMEFLAG EQU           X'01'            BIT = 1 WHEN TIME ELAPSES
```

Figure 8—7. Example of Interval Timer Island Code Linkage Using Symbolic Addresses (Part 2 of 2)

In this example, the SETIME macro instruction (line 6) requests a timer interrupt in 25 seconds so that a time limit of 25 seconds can be placed on the computation (lines 8 to 16) that follows. The STXIT macro instruction (line 4) attaches the interval timer island code routine (lines 27 to 29) to the task. The routine sets a flag when the time interval expires. The STXIT macro instruction is used again (lines 18 and 21) to detach the island code routine. The EXIT macro instruction (line 29) returns control from the island code routine to the interrupted task. Line 18 is the normal exit from the compute loop, which occurs if computation is completed before the timer elapses. Lines 20 and 25 are the error routine which is executed if the time elapses before the computation is completed. Line 31 defines the save area needed when the interrupt occurs.

8.6.8. Operator Communication

Your operator communication island code routine receives control when the operator enters an unsolicited message at the system console or workstation. He does this by typing the job number and a zero, followed by the message text. For additional details of the operating procedure at the system console or workstation, refer to the appropriate operations handbook for your system.

You can use the WTLD and OPR macro instructions to communicate with the operator. In these cases, your program displays a message on the system console or workstation and requests a reply. However, the use of operator communication island code routines permits the operator to enter a message for the attention of your program at any time during the execution of a job step without being prompted by your program. He could enter one of several predefined messages to acknowledge an event or a condition external to your program, for example, an infrequent request for statistics at the end of a particular job step.

The island code routine gains control at the entry point specified in the STXIT macro instruction that linked the island code routine to the job step. At this time, register 0 contains the length (including the character under the cursor) of the message entered by the operator. Register 1 contains either a zero, indicating that the operator communication was initiated at the console, or a negative sign, indicating that the operator communication was initiated at the workstation. (Register 1 would not contain an ECB address because operator communication island code routines always execute under the primary task TCB.)

To exit from operator communication island code, use the EXIT macro instruction to return to the interrupted task.

If the operator attempts to enter an unsolicited message for a job step for which there is no operator communication island code routine, or the island code routine has been detached, the message is rejected.

Figure 8—8 is an example of the use of the STXIT and EXIT macro instructions for operator communication island code routine using symbolic addresses. The general operation is similar to that described for program check (8.6.5). However, you will note that, in addition to the entry point and save area, the STXIT macro instruction also specifies a message area and message length.

Following the format, the STXIT macro instruction in line 21 specifies that it is attaching an operator communication island code routine (OC). The island code routine's entry point is SYSCON, the save area address is OC1, and the message from the system console is stored in a reserved 60-byte area whose address is OPRMSG.

```
                LABEL      ΔOPERATIONΔ          OPERAND                    Δ
                1          10                   16
1.                         LR                   R3,R7
.
.
.
21.                        STXIT                OC,SYSCON,OC1,OPRMSG,60
.
.
.
75.             SYSCON     LR                   R5,OPRMSG+3
.                                                                          } island code routine
.
.
89.                        EXIT                 OC
90.             OC1        DS                   18F
91.             OPRMSG     DS                   15F
```

Figure 8—8. Example of Operator Communication Island Code Linkage Using Symbolic Addresses

9.3.1. How to Call the Monitor Routine

There are two ways to call the monitor routine into main storage. Which one you use depends on whether you want to trace instructions from the beginning of the job or wait until after the job begins executing.

Whenever you use the monitor routine, keep this in mind: it occupies 3K bytes of main storage. If you specify the minimum main storage as a parameter of the JOB control statement, make sure you do not overestimate the storage size needed by your job, because it is possible that there might not be enough main storage available for the monitor when you combine your job needs plus the 3K bytes needed by the monitor.

Another point to remember: the monitor routine cannot be run in a strict spooling environment, because the job being monitored always requires the sole use of a printer. You can accomplish this through the *addr* parameter of the DVC job control statement which, in effect, dedicates a printer strictly to this job. It's coded immediately following the logical unit number (separated by a comma). Every device has a hardware address number associated with it. Your site manager can provide you with the number you need. (In most cases, however, this number can be physically found on the device itself, generally on some type of label.) This number is then coded in the device assignment set for the print file in your job.

Assume the printer you want to dedicate has a hardware address number of 170. Using 20 as the logical unit number, the DVC job control statement would be:

```
1        10        20        30        40        50
// DVC 20,170
```

It is also recommended that the job be run as the first job immediately after the system is intialized (initial program load) to ensure that the job is scheduled; otherwise, you might have to wait for the job to be scheduled, depending on the work load.

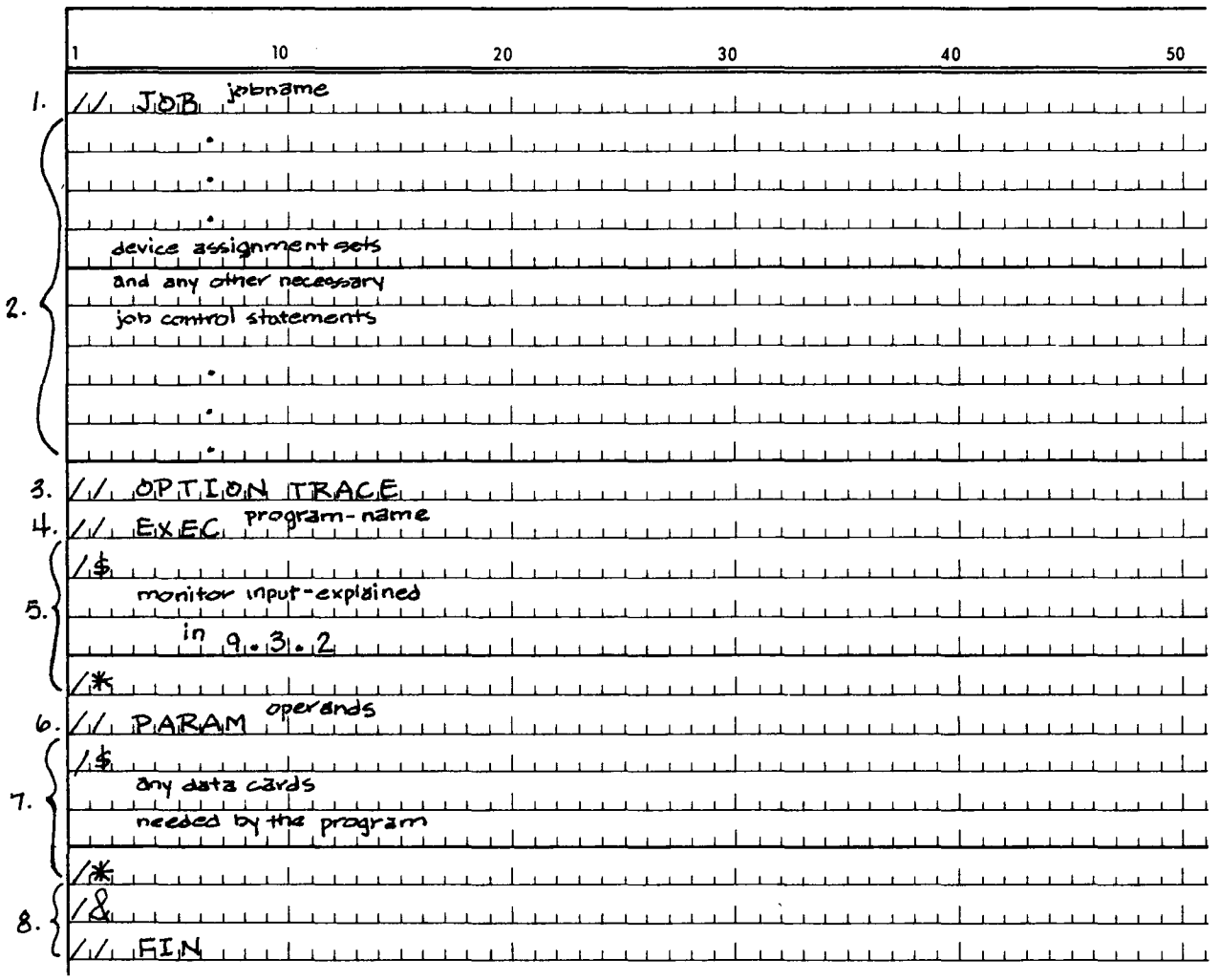
9.3.1.1. Monitoring From the Beginning of the Job

If you want to begin monitoring with the first instruction executed, you must call the monitor routine into main storage before the job to be monitored is run. In this case, the monitor input is entered as embedded data in the control stream.

The system operator types MO at the system console, which brings the monitor routine into main storage. The monitor initializes itself and awaits activation.

If you want to use the monitor beginning with the first instruction of the program, you must enter the monitor statements as embedded data in the job control stream. The job step that contains the program to be monitored must include an OPTION job control statement with the *TRACE* parameter specified. This parameter activates the monitor routine by setting the monitor bit in the PSW and creates a link between this job step and the monitor statements. If the OPTION job control statement is not present in the proper job step (the one with the monitor statements— — the one you want to trace), it will not activate the monitor routine because an OPTION job control statement is effective only in the job step in which it is encountered. As soon as the program begins executing, monitoring begins, and it continues until the program completes or until the monitor is deactivated by meeting the conditions that accompany a Q action (9.3.5.3).

The control stream you submit when you want to monitor from the beginning of the job would look something like this:



The explanation for each job control statement is the same as for the corresponding job control statement in 9.3.1.1. Notice the absence of an OPTION job control statement and the monitor statements, and the presence of the ALTER job control statement.

After the monitor statements have been read, the operator must issue the GO command, using the same job name as that on the JOB control statement. This resumes program execution under monitor control.

The second method for suspending the executing program is the use of an OPR macro instruction with a REPLY parameter (10.3.2). By placing it in a location near the area you want to monitor, you can use the halt when the program is suspended and the message it generates to instruct the operator to activate the monitor. Once again, the operator must have the monitor statements ready in the card reader (no /$ or /*). He then enters 00 MO R, to activate the monitor. After the monitor statements have been read, he enters the reply you requested with the OPR macro instruction to resume processing under monitor control. The monitor input is exactly the same as when using the first method. That is, no /$ or /* enclose it, and an OPTION TRACE job control statement is not submitted in the control stream. (And, in this case, no ALTER job control statement is submitted.)

The third method is to instruct the operator to type in the PAUSE command at some specific place in the program execution. This could be after a certain time limit has expired, or when a certain milestone is reached, such as the end of an input tape file. The operator places the monitor statements in the card reader and, when the system halts, types 00 MO R to activate the monitor routine. After the monitor statements are read, he finally types GO and the job name from the JOB control statement to resume program execution under monitor control. When activating the monitor in this way, the *P=phase-name entry cannot be used to specify the type of task to be monitored. Use either the *U=jobname or *S=symbiont-name entry in the monitor input deck. These entries to the monitor input format are described in 9.3.3.

There might be a situation when there is no card reader available to read in the monitor statement (or no keypunch readily available to prepare the monitor statements). If this is the case, the operator can type in 00 MO C at the system console. The C indicates to the system that the monitor statements are going to be input via the console, not via a card reader. (This applies to entering the monitor statements during all three methods of suspending program execution.) In this way the operator can enter the task, options, and actions at the console. He enters one card at a time, a line on the screen corresponding to a card in the monitor statement input, and indicates the end of each card by pressing the TRANSMIT key. After all monitor statements are sent, he enters the GO command followed by the job name.

9.3.2. Monitor Input Format

The monitor statements define what to monitor (task), when to monitor (option), and what to do when you monitor (action). This applies to monitor statements submitted via the control stream as embedded data before the job begins, and to the monitor statements used by the operator after program execution was begun. (Remember, the /$ and /* job control statements are only needed when the monitor statements are submitted as embedded data.)

For the program you want to monitor, only one task can be specified. It must be coded as the first monitor statement of the input, and no options or actions can share this card with the task. These tasks are explained in 9.3.3. For the task, however, you can specify up to 15 different options. (Each option must be on its own card; no two options can be present on the same card.) Each option can specify as many actions as will fit on a single card. A space must be used to separate the option from the first action on the card, and each succeeding action is separated from the previous action by a semicolon (;).

So, if you want to specify one option and one action, it would be coded as:

```
1        10        20        30        40        50
option action
```

If you wanted three different options, each with two actions, it would be coded as:

```
option-1 action-1;action-2
option-2 action-1;action-2
option-3 action-1;action-2
```

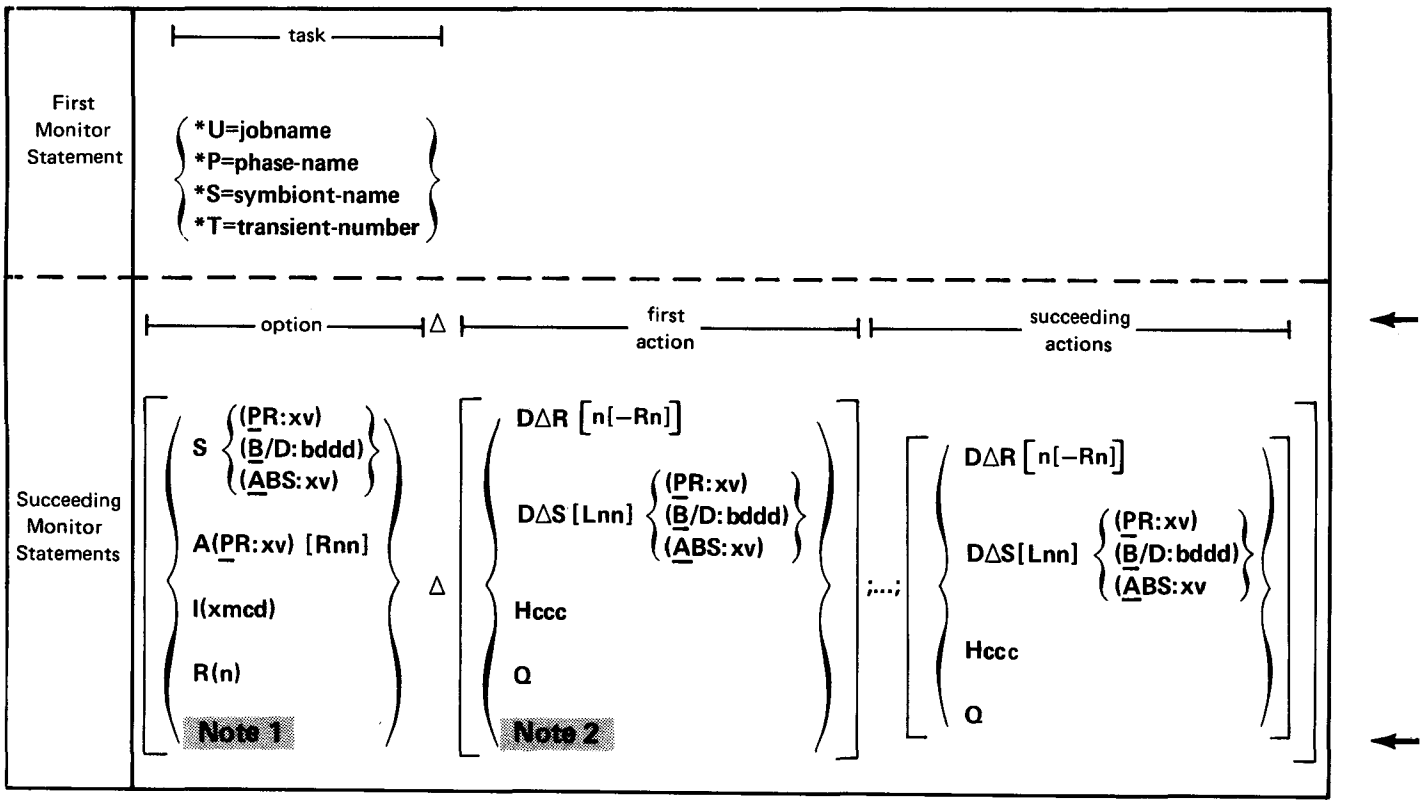
The last card used in the monitor input stream is a $ card. (Do not confuse this with the /$ job control statement, which indicates start of data.)

So, the order of a monitor input stream is:

- the task statement;
- the first option statement with its actions;
- any other option statements and their actions; and
- the $ card.

The options are described in 9.3.4, and the actions are defined in 9.3.5.

Figure 9—1 shows the format of the monitor statements.



NOTES:

1. If no option is specified, the monitor routine assumes a default option (9.3.4.5) and default display (9.3.5.1.3).

2. If no action is specified, the monitor routine produces a default display (9.3.5.1.3). Also, remember that the first action is separated from the option by a blank space, and any succeeding actions are separated from the previous action by a semicolon.

Figure 9—1. Monitor Input Format

9.3.3. Defining What You Want to Monitor

The task you want to monitor can be one of four types:

1. Your entire program
2. A certain phase of your program
3. A symbiont, which is a system utility routine
4. A transient, which is an OS/3 routine that is nonresident and is called into a transient area when needed.

In this format:

***U=jobname**

***P=phase-name**

***S=symbiont-name**

***T=transient-number**

you can see that each type has its own specification, and each type is preceded by an asterisk.

If you want to monitor all the phases of your program, use the **U=jobname* entry. The jobname is the same as the *jobname* parameter on the JOB control statement. (Remember, if you have the operator enter the monitor statements after the program has started, you can limit monitoring to a part of the job step; otherwise, the job step is monitored from the beginning.)

For example, if the JOB control statement is:

```
1         10        20        30        40        50
//JOB POCO
```

the monitor task statement would be:

```
*U=POCO
```

Since a program can consist of more than one phase, it can be useful to use the specific phase name with the **P=phase-name* entry. (A program can also have more than one phase.) If you want to monitor a phase, you have to know its name. The names of the phases used in a program are listed on the allocation map provided by the linkage editor. (Remember, operator input can limit the monitor to a portion of a phase.)

If the phase name you want is this:

```
                                   ** ALLOCATION MAP **

                LOAD MODULE -   LNKLOD      SIZE -   00000SCC

PHASE NAME      TRANS ADDR    FLAG      LABEL      TYPE      ESID      LNK ORG     MIADDR      LENGTH      OBJ ORG
LNKLOD00          NODE - ROOT                                         00000000    00000SCA    00000SCC
*** START OF AUTO-INCLUDED ELEMENTS -
        - 75/10/04 05.59 -              PR$IOE      OBJ
                                        PR$IOE     CSECT      01      00000000    00000A4F    00000480    00000000
                                        DP$COM7      ENTRY    01      00000000                            00000000
                                        DP$COM0      ENTRY    01      00000000                            00000000
                                        DP$COM1      ENTRY    01      00000000                            00000000
                                        DP$COM6      ENTRY    01      00000000                            00000000
                                        DP$COM2      ENTRY    01      00000000                            00000000
                                        DP$COM5      ENTRY    01      00000000                            00000000
                                        DP$COM4      ENTRY    01      00000000                            00000000
                                        DP$COM3      ENTRY    01      00000000                            00000000
```

For example, if you want the monitor routine to take action when the program reaches an instruction that references storage at absolute address 34AE, you would code:

```
1         10        20        30        40        50
S(ABS:34AE)
```

9.3.4.2. Instruction Location Option (A)

This option requests the monitor routine to take action when the specified instruction location is reached. Just as with the storage reference option, it uses the program relative address. However, you can also add a range to continue *this* monitor action for a specific number of bytes. It has only one format:

A(PR:xv) [Rnn]

The *xv* is the 1- to 6-hexadecimal-character program relative address (0_{16} to $FFFFFF_{16}$). If the program reaches an instruction at this location (program relative), monitor action begins. You can also continue monitor action for this option for a length of up to 255 bytes by specifying a range *(Rnn).* The allowable values for this range field are 02_{16} to FF_{16}.

For example, if you coded either:

```
A(PR:C02)
```

or

```
A(P:C02)
```

the monitor takes action for this option if the instruction at program relative address is reached.

If you coded (notice the convenient form P instead of PR):

```
A(P:C02)R0E
```

monitor action begins when the instruction at program relative address C02 is reached, and continues for 14 bytes (0E). This means the monitor action is to continue until program relative address C10 is reached. Note that you must use two hexadecimal characters for the range even when it can be expressed in one. In the last example, if the leading 0 of 0E was omitted, and it was coded as this:

```
A(P:C02)RE
```

monitoring would continue for 224 bytes to program relative address CE2.

9.3.4.3. Instruction Sequence Option (I)

This option requests the monitor routine to take action when the exact instruction sequence specified is reached. The monitor routine compares the machine code specified in the option entry to the actual instruction sequence of each instruction to be executed in the program being monitored, and takes action when an exact match occurs. The format for the instruction sequence option is:

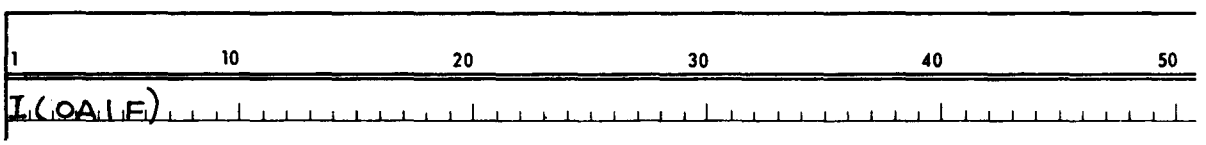
I(xmcd)

The *xmcd* stands for hexadecimal machine code. It may consist of from 2 to 64 hexadecimal characters (1 to 32 bytes). This is the value you want compared to the actual machine code being processed.

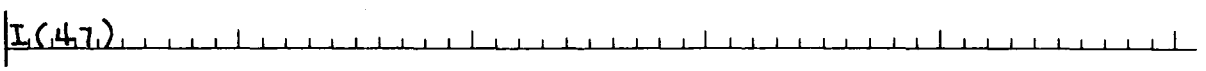
There are three different types of machine code sequences you can select:

- A single instruction
- Just the operation code of an instruction
- A string of instructions

For example, if you want monitor action to start when a supervisor call instruction for supervisor routine 31 occurs (SVC 31 in machine code = 0A1F), code it as:



If you want monitor action whenever *any* branch on condition instruction is reached (hexadecimal code = 47), you would code:



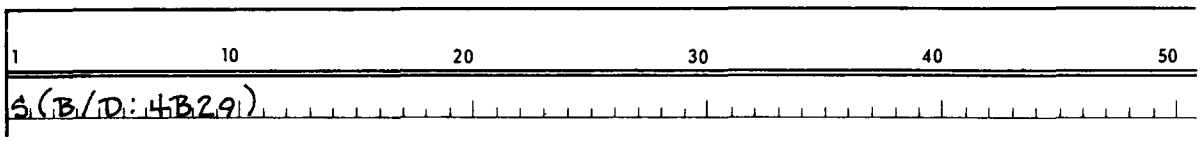
But if you want monitor action to occur whenever the following sequence of instructions occur (even though we are showing a series of inline expansion codes):

```
 LOC+  OBJECT CODE     ADDR1 ADDR2  LINE   SOURCE STATEMENT

000000                               1 PROG     START 0
000000 0560                          2 BEGIN    BALR  6,0
000002                               3          USING *,6
000002 47F0 6010              00012  4 BRANCH   B     *+16
000006 C1C2C3C4404040404040          5          DC    CL8'ABCD'
00000E C5C6C7C4                      6          DC    CL4'EFGH'
000012 4110 6102              00104  7          LA    1,LIST
                                     8          SNAP  (1)

000028 0A26                     A    18+        SVC   38 ISSUE SVC
00002A D207 60F2 6004 00004 00006    19         MVC   BUF(8),BRANCH+4
                                     20 TAG3    PUT   OUT
000030                          A    21+TAG3    DC    0Y(0) SLT ALIGNMENT
000030 5810 6116              00118  A  22+      L     1,=A(OUT) LOAD R1$, FILENAME ADDRESS
000034 9220 1031        00031        A  23+      MVI   49(1),X'20' SET FUNCTION CODE
000038 58F0 1034              00034  A  24+      L     15,52(,1) LOAD ADDR OF COMMON I/O
00003C 05EF                          A  25+      BALR  14,15 LINK TO COMMON
                                     26 TAG4    CLOSE OUT
00003E                          A    27+TAG4    DC    0Y(0)
00003E 5810 6116              00118  A  28+      L     1,=A(OUT) LOAD R1$, FILENAME ADDRESS
000042 0A27                          A  29+      SVC   39 ISSUE SVC
```

For example, assume that the following option statement was the only input to the monitor routine (and the task statement):



When the program reaches an instruction that references an address using base register 4 and a displacement of B29, a default display is given.

Remember, you can also get a default by omitting the option statement (9.3.4.5.). The only difference between the default display caused by omitting the option and the default display caused by omitting the action is that the omission of the option means that the option causing the display is not printed.

9.3.5.2. Halt Action (H)

This action, like the other actions, prints out items 1, 2, 3, and 4 (detailed in 9.3.5.1.1). It then prints a halt message on the system console and suspends program execution until a reply from the console operator allows execution to continue.

The halt message sent to the system console has the following format:

HALT ccc. TYPE-IN GO jobname TO RESUME

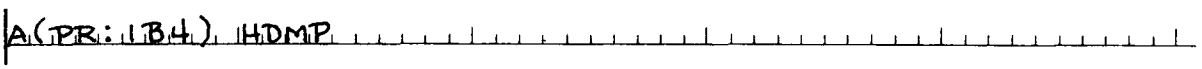
Program execution is then suspended until the operator issues the GO command followed by the job name (same as that on the JOB control statement). You can then provide the operator with special instructions about what to do before entering the GO command, such as taking a main storage dump. After he completes these special instructions, and enters the GO command, the instruction causing the halt is executed, and program processing continues under monitor control.

The format for the halt action is:

Hccc

The *ccc* is a 3-character EBCDIC code that you specify to identify the halt, and corresponds to the *ccc* in the halt message displayed to the operator.

For example, assume that your JOB control statement has a job name of TWESTMON, and uses the following monitor statement:



When the program reaches the instruction at program relative address 1B4, the monitor routine prints out the program information and displays the following message on the system console:

HALT DMP TYPE-IN GO TWESTMON TO RESUME

You would instruct the operator to take your desired action when he sees this message. In this case, assume it is a dump. After issuing the DUMP command (and a dump of main storage is given), the operator would then type:

GO TWESTMON

to reactivate the interrupted job. The instruction at program relative address 1B4 is then executed, and program processing continues under monitor control.

9.3.5.3. Quit Action (Q)

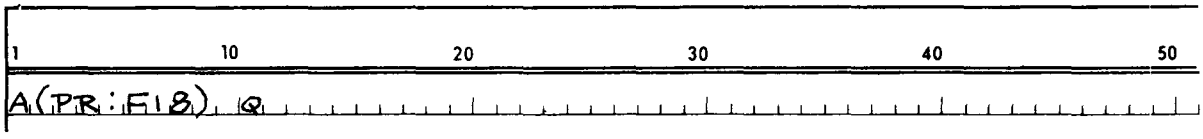
The quit action (Q) prints out items 1 through 4 and nothing else. The instruction causing the printout is then executed, and program processing continues without any further monitor intervention (pertaining to the option to which this action applies).

This action is useful when you want to monitor a problem area in the beginning of your program, and then exit from the monitor routine without tracing all the remaining instructions in the program (thus not wasting execution time).

The format for the quit action is:

Q

For example, if you coded:



the monitor routine would print out the program information when program execution reaches the instruction at program relative address F18. This instruction is then executed, and program processing continues without monitor intervention.

When the quit action is not used as one of the actions for an option, monitor processing continues until the end of the job step.

Table 9—2 summarizes the program information that is displayed by each action.

Table 9—2. Summary of Actions and Program Information Printed

Program Information Printed	Action				
	Display Register (D R)	Display Storage (D S)	Default Display	Halt (H)	Quit (Q)
Job name*	x	x	x	x	x
TCB address*	x	x	x	x	x
Program base address*	x	x	x	x	x
PSW contents	x	x	x	x	x
Next instruction to execute	x	x	x	x	x
Option causing this printout	x	x	x	x	x
Contents of specified registers	x				
Contents of specified storage		x			
Contents of changed registers			x		
Contents of referenced storage			x		
HALT message				x	

*These items are included for only the first option that causes a printout.

9.3.6. Cancel of Monitor

If the monitor routine is terminated abnormally, either by a CANCEL command or by a program exception within the monitor routine, all programs requesting the monitor routine will continue normal program processing without any type of monitor intervention. The monitor routine itself will dump and leave the system. A CANCEL command should not be issued while transcent monitoring is in progress.

9.4. SYSTEM DEBUGGING AIDS

Several debugging aids are built into the OS/3 supervisor to aid in solving system problems which cannot be identified through a normal SYSDUMP. These aids are useful only with some knowledge of the internal supervisor structure and are therefore not intended for general use. This section is provided for informational purposes only.

Table 9–3 summarizes the debugging aids described on the following pages.

Table 9—3. Summary of System Debugging Aids (Part 1 of 2)

Function	Use	Console Command	Results
Pseudo monitor*	To identify the routine changing a particular byte	SET HA,PM,address [,job-name]	HPR code 99130202 (Press RUN to continue.)
Resident monitor*	To identify the instruction changing a particular byte	SET HA,RM,address [,job-name]	HPR code 99130404 (Press RUN to continue.)
Mini monitor	To identify the instruction changing a particular byte	MM value,address, RTUE	HPR code 991200 (Press RUN to continue.)
Verify bytes 0-B*	To identify the routine destroying low-order storage	Included in supervisor debug option	HPR code 99130303 (Press RUN to continue.)
History tables*	To provide some recent history in SYSDUMPs	Included in supervisor debug option	Continuous updating of resident tables
Halt on transient load	To halt if and when a particular transient is loaded	SET HA,TL,hex-id	HPR code 990C0C (Press RUN to continue.)
Halt on transient call*	To halt if and when a particular transient is called	SET HA,TC,hex-id	HPR code 990C0D (Press RUN to continue.)
Halt on transient exit*	To halt if and when a particular transient exists	SET HA,TE,hex-id	HPR code 990C0E (Press RUN to continue.)
Halt on shared code call*	To halt if and when certain (or all) shared code modules are called.	SE HA,SC[,{module-name prefix.}]	991D01 (Press RUN to continue.)
Halt on shared code return*	To halt if and when certain (or all) shared code modules return.	SE HA,SR[,{module-name prefix.}]	991D02 (Press RUN to continue.)
Halt on shared code return with error*	To halt if and when certain (or all) shared code modules return with error.	SE HA,SE[,{module-name prefix.}]	991D03 (Press RUN to continue.)
Halt on symbiont load	To halt if and when a particular symbiont (or symbiont phase) is loaded	SET HA,SY,idnn	HPR code 997C (Press RUN to continue.)
Pause on shared code call*	To pause a task if and when certain (or all) shared code modules are called.	SE PA,SC[,{module-name prefix.}]	SE25 console message (Enter 'C' to continue.)
Pause on shared code return*	To pause a task if and when certain (or all) shared code modules return.	SE PA,SR[,{module-name prefix.}]	SE25 console message (Enter 'C' to continue.)
Pause on shared code return with error*	To pause a task if and when certain (or all) shared code modules return with error.	SE PA,SE[,{module-name prefix.}]	SE25 console message (Enter 'C' to continue.)

*Supervisor debug option required at IPL

Table 9—3. Summary of System Debugging Aids (Part 2 of 2)

Function	Use	Console Command	Results
PIOCS debug option	To identify checksum errors or internal PIOCS problems	SET DE,IO	HPR code 990F
Transient debug option	To halt on transient errors (100-1FF)	SET DE,TR	HPR code 99080800
Loader debug option	To halt on loader errors (52-5F)	SET DE,LD	HPR code 991500 (Press RUN to continue.)
Shared code debug option	To halt on errors detected during the execution of shared code.	SET DE,SC	HPR 990809 on shared code errors (Press RUN to take a SYSDUMP and to continue.) HPR 99130A when dynamic buffer pool links are destroyed.
Dynamic buffer debug option*	To halt on dynamic buffer overflow	SET DE,DB	HPR code 99130D
Screen format coordinator input/output debug option	To take a snapshot dump of all input and output buffer blocks when using the screen format coordinator	SET DE,INO	Writes snapshot dump to job log
Screen format coordinator format/input/output debug option	To take a snapshot dump of the format block; the input buffer (on input operations); the output buffer (on output operations) blocks; and, if errors occur, the screen format coordinator blocks	SET DE,FS	Writes snapshot dump to job log or system printer
Reset pause options	Resets all SE PA commands	SE PA,OFF	None
Reset halts	Resets all SE HA commands	SE HA,OFF	None
Reset debug options	Resets all SE DE commands	SE DE,OFF	None

*Supervisor Debug option required at IPL.

9.4.1. Supervisor Debug Option

The supervisor debug option is set at initial program load (IPL) time by entering D as the final character (following the comma) of the initial IPL message. This is described in the operations handbook. Use of this D option causes the supervisor being loaded to be expanded in size to support the supervisor debug option.

The following functions are provided:

- A normal halt (HPR code 99130101) between IPL and supervisor initialization. This allows changes to be made to the supervisor (via the maintenance panel) prior to loading the supervisor initialization load module. Normally, however, you should simply press the RUN switch on the maintenance panel to continue.

- A pseudo monitor to detect when any byte within the supervisor has been changed. When activated this function checks the byte on every interrupt and on every pass through the switcher. When the byte is changed, the supervisor halts (HPR code 99130202) without restoring the original contents of the byte. If you want to continue, press RUN. The *new* value becomes the *original* value and the supervisor halts if the byte is changed again.

- The console command to activate the pseudo monitor is:

  SET HA,PM,address[,job-name]

  where address is the address of the byte to be monitored either absolute (no job-name specified) or relative to the preamble of a currently active job if you specify one with job-name. After the pseudo monitor is activated you use this same command to change the address of the byte being monitored.

- Verification of low-order main storage (locations 0–B) on every interrupt and every pass through the switcher. When changed, the supervisor saves the incorrect setting, restores the correct setting and halts (99130303). Although you may continue past this HPR by pressing RUN, you should take a SYSDUMP here to determine why these bytes are being altered.

- A resident supervisor monitor to detect when any byte in main storage has been changed. When activated, this function checks the byte upon executing every instruction in supervisor critical code (interrupt processing), transients, symbionts, and job control. The only code not monitored is code being executed under a key other than 0 (i.e., user jobs). Monitoring user jobs is unnecessary because the hardware key protection feature of the processor prevents user jobs from destroying any part of the supervisor.

  When the specified byte is changed, the resident monitor halts (99130404) without restoring the original contents. The double word at location 80 contains the PSW at the time the byte was changed. If you want to continue, simply press RUN. The *new* value becomes the *original* value and the supervisor will halt if the byte is changed again.

9.4.6. Shared Code Halts and Pauses

SET console commands are available to interrupt or halt processing when shared code modules are called or when they return. These commands allow the operator to request an interrupt or halt on the call or return for:

- a specific module
- a specific group of modules which have a common prefix; or
- all modules.

The format of these commands is:

$$SE\ \left\{\begin{matrix}HA\\PA\end{matrix}\right\},\left\{\begin{matrix}SC\\SR\\SE\end{matrix}\right\}\left[,\left\{\begin{matrix}prefix.\\name\end{matrix}\right\}\right]$$

The first and second parameters form individual commands which are discussed in the following paragraphs. The third parameter determines what modules these commands affect. You specify an individual module by its full name, a module group by its prefix followed immediately by a period, or all modules by omitting the parameter completely. For example, the command SE HA,SC,DM. would cause an HPR upon a call to any module whose name begins with DM.

You can continue past any HPR resulting from these commands by pressing RUN. The supervisor debug option is required at IPL time for all of these functions. The individual commands are:

- Halt on shared code call. The SE HA,SC command causes an HPR of 991D01 when a module is called.
- Halt on shared code return. The SE HA,SR command causes an HPR of 991D02 when control returns from a module.
- Halt on shared code return with error. The SE HA,SE command causes an HPR of 991D03 when control returns from a module with an error condition.
- Pause on shared code call. The SE PA,SC command interrupts processing and displays the following message when a module is called:

  SE25 SC PAUSE ON shared-code-name. CONTINUE? (Y, HELP)

  This message shows which shared code module has been called. A reply of Y causes processing to resume. A reply of HELP displays the following information: the job or symbiont name, the name of the calling module, the TCB address, the base address of the calling module, and the local store address.

- Pause on shared code return. The SE PA,SR command interrupts processing and displays the SE25 message when control returns from a module by execution of the SRETURN macroinstruction. If requested to, this command displays the same shared code information as SE PA,SC does except that it shows what module is being returned to rather than what module called the shared code.

- Pause on shared code return with error. The SE PA,SE command interrupts processing and displays the SE25 message when control returns from a module in which an error has occurred. If requested to, this command displays the same shared code information as SE PA,SR does.

9.4.7. Soft-Patch Symbiont (PT)

The PT symbiont is used to temporarily patch transients (transient overlays), load modules, and shared code modules at the time they are loaded in main storage (soft patch) instead of permanently patching the disk (hard patch). This is useful if you want to test a patch to see if it is effective before hard-patching or if you want to trap a problem by temporarily applying a patch. To use the PT symbiont, you must have included the supervisor debug option at IPL time.

When initiated, the PT symbiont builds a patch table from input read from cards or keyed in from the console. The PT symbiont then locks itself into the supervisor so it can scan this table on every load of a transient, load module, or shared code module. During this scan, if the module name matches an entry on the patch table, the specified patches are applied. These patches are temporary. Patches to transients remain in effect until the PT symbiont is cancelled. Load and shared code modules that are loaded in main storage while the PT symbiont is active remain patched until reloaded.

The PT symbiont is also used to apply patches to the resident supervisor; however, these patches remain in effect until you IPL the system again.

9.4.7.1. Soft-Patching Using Card Input

When using card input to soft-patch, you must create the card deck containing the desired patches. Once prepared, the deck is placed in the system reader prior to initiating the PT symbiont. The input deck consists of four card types:

1. The first card is provided for compatibility purposes. It is necessary when using the transient patch (TRNPAT) program, which applies core to transients.

   Format:

   1 D=R

   The card must have a 1 in column 1, followed by a blank in column 2, and then D=R.

2. The second card defines the type and the id (or name) of the module to be patched. The form of this card depends upon the module type.

   Formats:

   2 T=decimal-id (for transients)

   2 S=module-name (for shared code modules)

   2 L=module-name (for load modules — the load module can be the resident supervisor, a symbiont, or a module loaded from a user library)

   2 O=module-name (for resident supervisor modules specifying the csect or object module name — this format can only be used when operating in a mixed or CDI mode environment)

   In all the formats, a 2 must appear in column 1, followed by a blank in column 2. Each module to be patched must be defined with one of these cards.

3. The third card defines the patch. Each card contains only one patch, and the patch is applied only to the module specified in the preceding 2 card.

   Format:

   P addr,patch

   A P must appear in column 1, followed by a blank in column 2. Starting in column 3, the hexadecimal address (relative to the start of the transient or module phase to be patched) is entered. The address must be within the module specified or the card will be ignored. The address is followed by a comma and then the patch. (The patch is also given in hexadecimal, and embedded blanks are not permitted.) The patch character string can be any length, though the entire P entry must fit on one card (or one line of the console, if using console input).

   More than one patch can be made to a module by entering more than one P card. All patches to be applied to a given module should be specified in successive P cards following the 2 card that defines the module.

4. The last card signifies the end of the patches. The symbols are entered in columns 1 and 2.

   Format:

   ```
   /*
   ```

The following is a sample deck of cards:

1 D=R	Can be eliminated if not using TRNPAT
2 L=MYSAL	Defines a load module MYSAL
P 1A,47000000	Defines a patch to be applied to MYSAL
2 T=1539	Defines a transient
P 94,C0	Defines two patches to be applied to the transient
P 12A,4780F2E49966	
2 S=MYSHRCOD	Defines a shared code module MYSHRCOD
P 24,07C0	Defines a patch to be applied to MYSHRCOD
2 O=SM$DEBUG	Defines an object module SM$DEBUG
P 27,FF	Defines a patch to be applied to SM$DEBUG
/*	Indicates the end of the patches

Once the card deck is created and placed in the system reader, initiate the PT symbiont by keying in the following command from the console:

PT

Once initiated, the PT symbiont accepts the patches on the card deck from the system reader and applies them to the specified modules as they are loaded.

9.4.7.2. Soft-Patching Using Console Input

Soft-patching can also be accomplished by entering the required input directly from the keyboard of the system console. When using this method, the PT symbiont must be initiated before entering any input. To initiate the PT symbiont, key in the following console command:

PT C

Once initiated, the PT symbiont solicits input from the system console. The input you key in is entered in the same card-image format as that of the four card formats described in 9.4.7.1. The PT symbiont builds a patch table from your input and applies the patches as the specified modules are loaded.

There are some optional features available to you when soft-patching directly from the console. For example, you can key in the following console command to initiate the PT symbiont:

PT[dev-addr] C

This form of the command not only solicits patch input from the console, but it also punches that input on the device specified. The card deck produced contains the patches that you can reuse at some later time.

You can also enter all the information for a single patch as part of the PT command format when patching from the system console. The following is the format of this option:

```
PT { T,transient-id                  }
   { S,shared-code-module-name       } ,addr,patch
   { L,load-module-name              }
   { O,object-module-name            }
```

Although this form eliminates the need of separate entries for 2 and P type card-image inputs, it can only be used to make a patch at one location (module address). To patch more than one location, use one of the other forms of the command, or key in this form one time for each location to be patched.

Examples:

```
PT L,MYSAL,1A,47000000
PT T,440,F0,45A0F220
PT S,MYSHRCOD,24,07C0
PT O,SM$DEBUG,27,FF
```

NOTE:

The object-module-name entry can only be used in systems with mixed or CDI mode environments.

9.4.7.3. Using the PT Command

Whether you use card input or console input, you can enter the PT symbiont command more than once and the input is simply added to the end of the patch table. In addition, any combination of the various forms can be used. For example, you can key in PT and a patch table is built from the card input. Later in the same session, you can key in PT C and enter additional patches. These additional patches are added to the existing patch table.

9.4.7.4. Cancelling the PT Symbiont

Regardless of how the input is entered, the PT symbiont can be cancelled at any time by keying in the following console command:

```
CA PT,S,N
```

Cancelling the PT symbiont eliminates all the patches entered, except those that changed the resident supervisor. (These will remain in effect until you perform the IPL again.) Shared code and load modules that were loaded while the PT symbiont was active will remain patched until reloaded. Subsequent loads of modules, however, will not be patched.

9.4.7.5. PT Symbiont Error Messages

Error messages are produced by the PT symbiont and appear on the console screen. The following is a list of the error messages that might occur, the condition that caused the error, and the corrective action to be taken.

PT01 TWO 2 IMAGES IN A ROW

Two 2 cards have been entered in a row. This is invalid because a module has been specified to be patched, but no patches have been entered. The first 2 card is ignored, and the PT symbiont continues. This could result in incorrectly applied patches. To avoid this, cancel the PT symbiont, correct the input deck, and begin a new PT symbiont session.

PT02 INVALID CHARACTER STRING, CHARACTER ON CARD

A non hexadecimal digit (other than 0—9 and A—F) was entered in a field requiring a hexadecimal digit. This message is also produced if an odd number of characters was entered for a patch (patches cannot be half bytes in length). Cancel the PT symbiont, correct the input deck, and begin a new PT symbiont session.

PT03 PATCH TABLE OVERFLOW — SOME PATCHES LOST

Too many patches have been entered. There is a limited amount of space that can be allotted to the patch table, and the PT symbiont will stop accepting input when this limit is exceeded. This could result in a patch table that contains only part of the patches you intended to apply. To avoid this, cancel the PT symbiont. Limit the number of soft patches you enter, and begin a new PT symbiont session.

PT04 INVALID PT — NEEDS SUPV DEBUG OPTION SET AT IPL

The supervisor debug option, which is required if the PT symbiont is used, was not specified at IPL time. The PT command is ignored, and the symbiont cannot be initiated. You must IPL the supervisor again, specifying the debug option; then begin a new PT symbiont session.

PT05 PUNCH SPECIFIED BUT NO CONSOLE INPUT

The form of the PT command specifying a punch device was used, but the input was not specified as coming from the console. This will occur if the C following the device address was not entered. The PT command is ignored under this condition. Reenter the command, including the final C.

PT06 csect-name NOT FOUND

The object module or csect name specified on the 2 O= card was not found on the supervisor currently loaded. The 2 O= card and all the P cards until the next 2 card are ignored. If an incorrect object module or csect name was entered, you can enter the correct name later in the session and the input will be added to the patch table.

PT07 SYSRDR NOT AVAILABLE

The form of the PT command used requires the system reader device, but in this case it is unavailable. The PT command is ignored. When the system reader becomes available, reenter the command.

PT08 INVALID INPUT FORMAT

An error was made in entering the information for a patch on a single line from the console. The PT command is ignored under this condition. Check to make sure that all commas are in the right place, and reenter the command.

11. Other Services

11.1. SPOOLING

11.1.1. General

Spooling is the technique of buffering data files for low speed input and output devices to a high speed storage device independently of the program that uses the input data or generates the output data. Data from card readers or from remote sites is stored on disk for subsequent use by the intended program. Data output by the program is stored on disk for subsequent punching or printing. The spooling function also handles diskette files. It treats input from diskette as though it were from a card reader, and output to a diskette as though it were to a card punch. In this description of spooling, any reference to a card reader, card input, or card file also includes diskette input; any reference to a card punch, card output, or card file also includes diskette output. The data management user guide, UP-8068 (current version) shows the formats for diskette records.

Spooling enhances system performance by releasing large production programs and system software from the constraint of the slower speed devices, thereby freeing the main storage occupied by these programs sooner; and by driving the slower speed devices at their rated speed on a continuous basis, thereby making full use of the devices during the time that is normally lost to systems overhead or to job steps not using printers.

The spooling function comprises five elements: initialization, input reader, spooler, output writer, and special functions. These elements are described on the following pages. Figure 11—1 gives a simplified picture of the relationship between the slow and high speed input/output devices and the software components of the spooling function and the supervisor.

11.1.1.1. Initialization

Spool initialization provides for the establishment, data recovery, or reestablishment of the spoolfile at supervisor initialization. Based on system generation parameters or operator specified options at supervisor initialization, it allocates the spoolfile and builds the system spool control table, or it recovers an existing spoolfile. In the case of an existing spoolfile, it clears the file, recovers closed subfiles, or recovers and closes all subfiles.

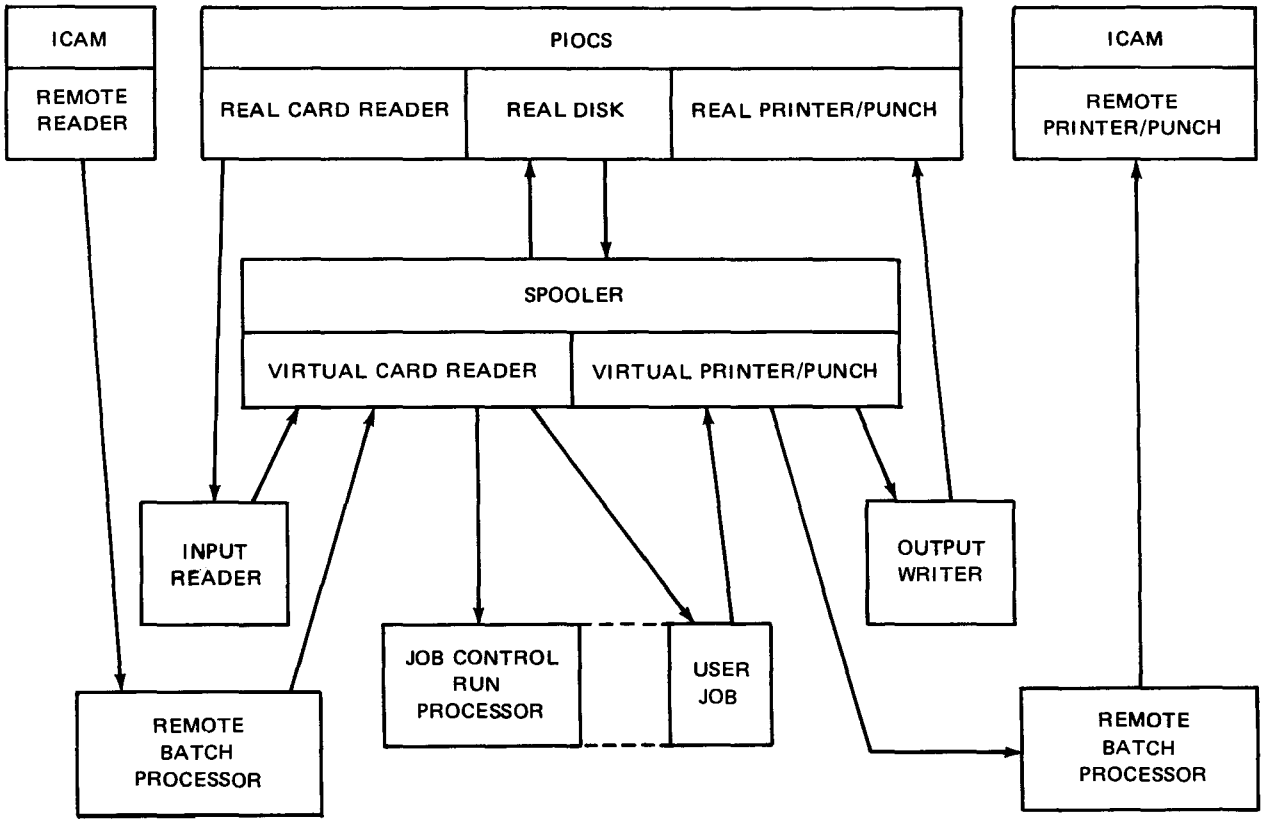


Figure 11—1. Relationship of Spooling Devices and Programs

11.1.1.2. Input Reader

Using PIOCS the input reader reads cards from a real card reader or records from a diskette and writes these images to the spoolfile via a virtual card reader and the spooler. It closes the previous subfile if one exists and opens a new subfile. A given input reader can handle only one card reader or diskette at a time; however, any number of input readers can be active.

11.1.1.3. Spooler

The spooler is the hub of the spooling package and is linked as part of the resident supervisor. It provides record level input and output to and from the spoolfile for each element in the system needing access to that file. It intercepts all input/output commands to virtual printer, punch, and card reader devices, and accesses the disc when necessary using the system access technique (SAT) for accesses to the spoolfile. All input/output requests (EXCP macro instructions) addressing virtual devices are trapped and routed to the spooler for processing rather than PIOCS. The spooler supports both reads and writes to virtual devices while simulating the action of PIOCS as far as error handling, page spacing, and synchronization are concerned. It allocates tracks to subfiles and maintains control of the user's spool control tables. It can handle any number of print, punch, and read files simultaneously, including multiple files per job.

Job control options for spooling are entered using the JOB, SPL, DATA, and DST job control statements. These are described in the job control user guide, UP-8065 (current version). Initialization options are also entered by the system operator. These are described in the appropriate operations handbook for your system.

There are no changes required to a user program to use spooling. You can define your files using either data management macro instructions, or PIOCS macro instructions. A job that runs on a nonspooling system will also run on a spooling system, and vice versa. If you use the BRKPT macro instruction in your program, it will be ignored if your job is run on a nonspooling system.

NOTE:

Spooling always permits redirected output to tape. To redirect output to disk, however, you must include dynamic buffer management either explicitly or using other parameters at system installation time.

11.1.3. Create a Breakpoint in a Spool Output File (BRKPT)

Function:

The BRKPT macro instruction creates a breakpoint in a printer or punch spoolfile. It closes and reopens the subfile as it is being generated by the spooler. Each segment created at this breakpoint is considered a logical subfile so that output to the physical device can be started prior to job step termination.

If this macro instruction is included in a program executing in a system that does not have the spooling capability, the macro instruction is ignored.

Format:

LABEL	△OPERATION△	OPERAND
[symbol]	BRKPT	{filename / CCB-name / (1)}

Positional Parameter 1:

filename
Specifies the symbolic address of the DTF macro instruction in the program which defines the file in which a breakpoint is to be created. Use this parameter if you are using data management macro instructions to define and access the file.

CCB-name
Specifies the symbolic address of the command control block (CCB) associated with the file in which the breakpoint is to be created. Use this parameter if you are using PIOCS macro instructions to define and access the file.

(1)
Specifies that register 1 has been loaded with the address of the DTF macro instruction or CCB associated with the file to be breakpointed.

11.2. JOB ACCOUNTING

11.2.1. General

The job accounting package consists of resident routines which are linked with the supervisor and elements of the job step processor at system generation time. These routines provide a count of the facilities utilized by each job step during its execution within the system. The message logging facility of the spooling function transfers this data from main storage to disk as part of the output spoolfile. The output writer prints the job step and job values as part of the normal message log output for each job. Optionally, the output writer can write the accounting information to a standard SAM magnetic tape file for offline processing by user-developed accounting routines or by OS/3 data utility routines. You can assign an account number using the JOB job control statement which is carried along with the accounting records. This enables you to accumulate statistics from the SAM file for computer time and resources charged against an account number, which could represent a project, department, cost center, etc. The job accounting function requires the use of the spooling package and the optional timer facilities. These must be included at system generation time. Also, the job accounting versions of SVC decode and the switcher must be included within the supervisor at link edit time.

11.2.2. Accounting Data

Accounting data is accumulated in a job accounting table (Figure 11—2) in the job prologue. Fields in this table serve as counters for job step and job statistics.

Byte	0	1	2	3
0	count of SVCs in job step			
4	count of SVCs in job			
8	count of transient calls in job step			
12	count of transient calls in job			
16	CPU time used by job step			
20	CPU time used by job			
24	length of largest job step (in bytes)			
28	time of day that job step started			
32	time of day that job started			
36	accumulated time of day of all job steps			

Figure 11—2. Job Accounting Table Format (Part 1 of 2)

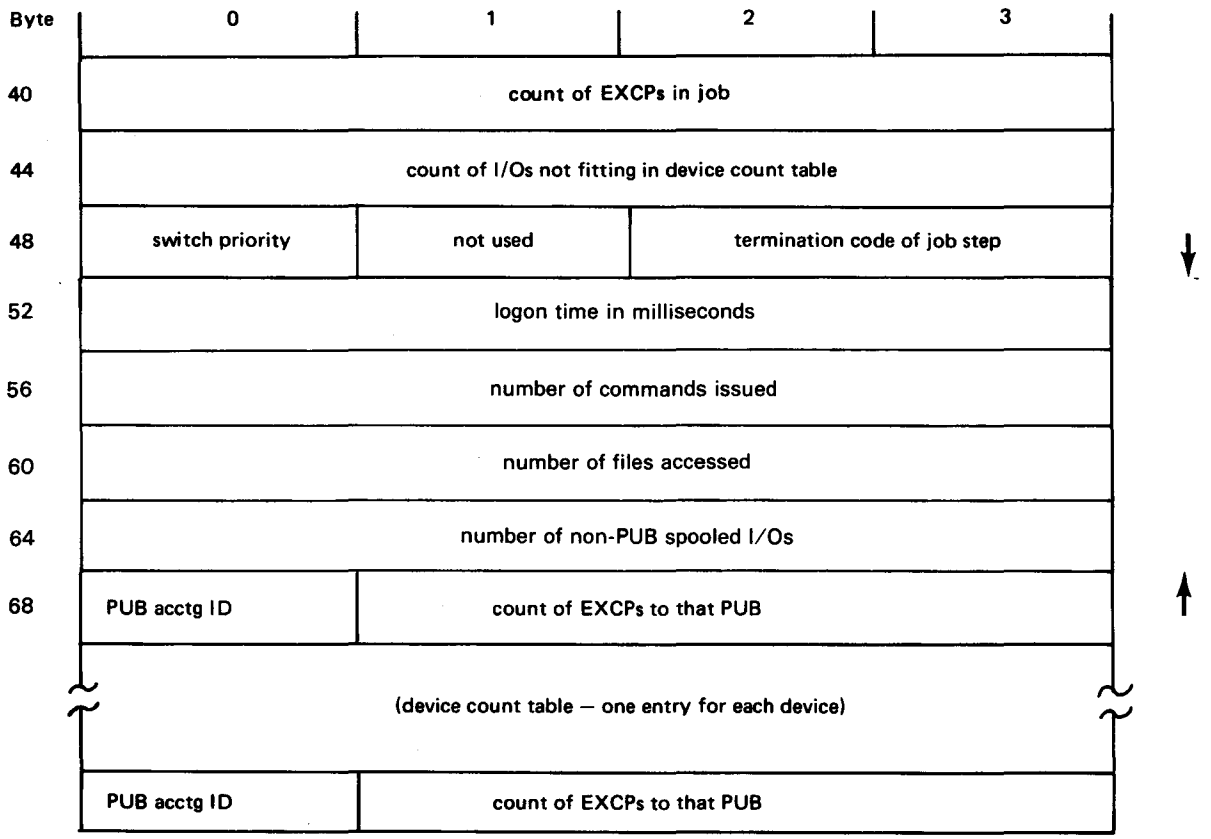


Figure 11—2. Job Accounting Table Format (Part 2 of 2)

11.2.2.1. Job Step Level Data

Counters in the job accounting table are dynamically incremented during job step execution. The following data is collected for each job step:

- Central processor time

  This consists of the total time in milliseconds charged to tasks of this job or supervisor tasks working for this job. This means that all supervisor overhead, such as processing SVCs and the processing of supervisor tasks is charged to the requesting job. Supervisor idle (wait) time is not charged to any job.

- Total SVCs executed

  This consists of the total number of SVCs executed by the job's tasks or by supervisor tasks working in behalf of the job.

- Total transient functions

  This consists of the total number of transient functions executed by the job's tasks or by supervisor tasks working in behalf of the job. This does not include overlays to transients.

- Total I/O requests

  This consists of the total number of I/O requests executed for each device by the job's tasks or by supervisor tasks working in behalf of the job. I/O requests per device include spooling activity in terms of the number of cards read from the spool file and print lines written to the spool file by this job step.

In addition to the counts dynamically maintained in the job accounting table, the job step processor furnishes the following values for job step accounting:

- Total wall clock time required for the job step to execute. This does not include time during which the job step was rolled out, nor does it include the period between the time a checkpoint was taken and the job step was restarted from the last checkpoint.

- Total main storage into which programs were loaded by the loader.

  This value represents only that amount of main storage used by the job step as recorded by the loader, and does not include the prologue or those available areas within the job region which are used but not for loading.

- Initial switch priority of the job step.

- Termination code of the job step. Normal termination code is 000.

- Value of the User Program Switch Indicators (UPSI) at job step termination.

11.2.2.2. Job Level Data

Some of the data collected for the job steps of a particular job is totalled for the job's accounting record. In addition, data is collected on the job level which cannot be acquired by just summing the job step values. That data which is collected solely for the job is recorded at job termination time and consists of the following:

- Size of the largest job step.

- Job date

  This is the date from the job preamble representing the date the job was run.

- Total job main storage including prologue.

- Total wall clock time for the job, including all of the job step processor overhead.

  Wall clock time is defined as the point in time when a job is initiated to execute up to the point in time when the job termination message is displayed, and does not include spool time.

- Total wall clock time for all job steps.

  This is a sum of the total wall clock time for each job step and does not include job control time.

- Total CPU time for all job steps.

  This is a sum of the CPU time for each job step and does not include job control time.

- Total SVC count for all job steps.

  This is a sum of the SVC counts for each job step and does not include job control counts.

- Total transients called for all job steps.

  This is a sum of the transients called by the job steps and does not include job control counts.

- Total I/O count for all job steps.

  This is a sum of the I/Os executed by the job steps and does not include job control counts.

11.2.3. Data Printout

When printing the job's log, the output writer also prints the accounting records for that job. Also, the output writer can write all the job log records to a magnetic tape for offline processing, or only the log records, or the accounting records. This gives you the ability to create a system log file and a system accounting file for subsequent statistical processing and evaluation. Figure 11—3 shows the format of the job accounting record printout.

SPERRY UNIVAC

PRINTER FORMAT CHART
160 PRINT POSITIONS

APPLICATION: _____ TYPE OF PRINTER _____

PROGRAMMER _____ DATE: _____

Job Header	AC01	JOB job-name ACCT. NO. xxxx ASSIGNED MEMORY=xxxxxxxx BYTES (PLUS xxxxxx BYTE PROLOGUE) yy/mm/dd JOB #nn A hh:mm:ss
Spool Header	AC02	machn supnam vv.r.sqq A hh:mm:ss
Step Header	AC10	LFD - LFD-NAME FORM NAME - FORMNAME, COPIES - xxxx, PAGES - xxxxxxxx, STEP =xxx A hh:mm:ss
	AC11	STEP #xxx (stepname) USED xxxxxxxx BYTES ELAPSED WALL CLOCK TIME=hh:mm:ss.mil TOTAL SVC CALLS=xxxxxxxx A hh:mm:ss
	AC12	TERM CODE=xxx SWITCH-PRIORITY=xx CPU TIME USED =hh:mm:ss.mil TRANSIENT CALLS=xxxxxxxx A hh:mm:ss
	AC13	UPSI SETTING X'nn' A hh:mm:ss
Device Counts	AC19	DEVICE EXCP'S ddd=xxxxxxxx ddd=xxxxxxxx ddd xxxxxxxx ddd=xxxxxxxx ddd=xxxxxxxx A hh:mm:ss
Job Totals	AC21	JOB TOTALS USED xxxxxxxx BYTES TOTAL ELAPSED WALL CLOCK TIME=hh:mm:ss.mil TOTAL JOB SVC CALLS=xxxxxxxx A hh:mm:ss
	AC22	WALL CLOCK TIME OF ALL STEPS =hh:mm:ss.mil JOB TRANSIENT CALLS=xxxxxxxx A hh:mm:ss
	AC23	TOTAL CPU TIME OF ALL STEPS =hh:mm:ss.mil TOTAL JOB EXCP's =xxxxxxxx A hh:mm:ss

UD1-1505

Figure 11—3. Job Accounting Record Printout Format

SPERRY UNIVAC

USER COMMENT SHEET

Your comments concerning this document will be welcomed by Sperry Univac for use in improving subsequent editions.

Please note: This form is not intended to be used as an order blank.

_____

(Document Title)

_____ _____ _____

(Document No.) *(Revision No.)* *(Update No.)*

Comments:

Cut along line.

From:

_____

(Name of User)

_____

(Business Address)

Fold on dotted lines, and mail. (No postage stamp is necessary if mailed in the U.S.A.)
Thank you for your cooperation

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 21 BLUE BELL, PA.

POSTAGE WILL BE PAID BY ADDRESSEE

SPERRY UNIVAC

ATTN.: SYSTEMS PUBLICATIONS

P.O. BOX 500
BLUE BELL, PENNSYLVANIA 19424

CUT

FOLD