IBM

Programming Systems Analysis Guide

IBM 7030 Master Control Program (MCP)

PREFACE

This manual was prepared by Programming Systems to provide detailed information on the internal logic of the IBM 7030 Master Control Program (MCP). It is intended for systems programming personnel who are responsible for diagnosing system operation or for adapting the programming system to special usage.

Certain knowledge is prerequisite for the full utilization of this manual. It is assumed that the reader has a basic knowledge of the IBM 7030 and its basic language, STRAP II. Such background knowledge can be obtained from the IBM 7030 Data Processing System Reference Manual, Form A22-6530-2, and from the IBM Reference Manual, STRAP II 7030 Assembly Program, Form C28-6129. It is also assumed that the reader has a general knowledge of MCP as described in the IBM 7030 Data Processing System Master Control Program Reference Manual, Form C22-6678.

CONTENTS

ILLUSTRATIONS

The IBM Programming System for the IBM 7030 Data Processing System consists of the Master Control Program (MCP), the assembly program (STRAP II), FORTRAN, and the System Macro Language, SMAC I. MCP is the system control program; the other three programs are language processors. The programs constituting the system are described in detail in their respective Programming Systems Analysis Guides. This manual, then, is one of a set of four Analysis Guides which describe the IBM 7030 Programming System completely.

How MCP is used with respect to computer operations and problem programs is described in the IBM 7030 Data Processing System Master Control Program Reference Manual, Form C22-6678. Detailed information about the functions performed by MCP is described in the Analysis Guide, together with miscellaneous programming information. This, together with the program listing, enables the systems programmer to determine how MCP accomplishes any given task, and to adapt MCP to such special usage as the installation may require. The Analysis Guide, designed to complement the program listing, is a tool for the systems programmer to help him make most effective use of the MCP program listing.

For purposes of exposition, MCP has been divided into the following five basic functional areas:

1. Interrupt control
2. System operation
3. System I-O
4. Service routines
5. Debugging aids

Each area is described in a section of the Analysis Guide. Additional sections provide a general description of MCP, and descriptions of the supporting programs.

Flow charts have been included in context for ease of reference. They are drawn at a functional level, and conform to the standards specified in Appendix A. Program symbols are used freely in both the text and the flow charts in order to relate the Analysis Guide and the listing as closely as possible.

The subprograms of MCP are closely interrelated. In order to avoid diversionary discussions throughout the Analysis Guide, each section has been written presuming knowledge of the other sections whenever necessary. The reader with the responsibilities outlined in the preface should read the entire Analysis Guide once to become familiar with content, terminology, and level of discussion, before attempting to put it to use.

GENERAL DESCRIPTION OF MCP

The Master Control Program (MCP) for the IBM 7030 is composed of many small programs whose overall purpose is to expedite the flow of work through the 7030 system. They total approximately 11,000 words, most of which remain in core storage at all times. They rely on a set of supporting programs to perform peripheral I-O operations and system update.

THE LOGICAL STRUCTURE OF MCP

MCP is composed of three kinds of programs:

1. Interrupt control programs
2. Service routines
3. Major packages

These three categories constitute distinct logical levels within MCP. They operate within the framework of a few basic priority and control concepts.

Interrupt Control Programs

One of the design features of the 7030 is its interrupt system. MCP takes all interrupts, although some may be subsequently routed to a problem program (PP). Included in the category of interrupt control programs are those programs which receive control directly or almost directly from an interrupt table. These programs operate with the interrupt system disabled, and, except for interrupts due to non-recoverable errors, they route control to some other program. Since MCP uses the IF interrupt as a basic means of communication, the interrupt control programs concerned with the IF interrupt are referred to collectively as the dispatcher. Similarly, the interrupt programs concerned with handling I-O interrupts are referred to as the receptor.

Service Routines

MCP permits the problem program (PP) to use symbolic I-O, and performs I-O operations on the proper unit when requested by the PP. The programs which actually accomplish these services (i.e., read, write, rewind, etc.) are called service routines. The group of service routines concerned exclusively with symbolic I-O operations is called the actuator. Additional service routines perform such functions as suppression and release of I-O interrupts, setting of the interval timer, etc.. The service routines may be explicitly defined as those routines which accomplish pseudo-ops for pseudo-op codes less than 64.0. They operate disabled, and are effectively an extension of the requesting program. They receive control from the dispatcher and return to the point of request.

Major Packages

The remaining programs in MCP are major packages. In terms of pseudo-ops, a major package is a program which services a pseudo-op whose code is 64.0 or greater. A major package has characteristics similar to those of a problem program. It may operate enabled, may suppress I-O interrupts, may perform I-O (but must use symbolic I-O and the service routines), may use other major packages, and has priority over a problem program and its interrupts. A major package may be primed; that is, have its pseudo-op and parameters entered into a queue for execution at a later time.

Program Control and Priority

MCP receives control only when an interrupt occurs. Pseudo-op requests enter MCP via the IF interrupts. Since the service routines are considered a logical extension of the requesting program (MCP or PP), it is necessary to introduce the concept of level of program operation. The program level, or instruction counter owner, is MCP when an MCP major package is in control or about to receive control. At all other times the level is PP. The system level (SL) bit is used by MCP for control logic. Note that the service routines operate at either level, depending on the level of the program requesting the service pseudo-op. Thus, an MCP service routine may be in control, but may be operating at PP level. However, all portions of MCP in which this is possible (i.e., non-major packages) operate with the interrupt system disabled.

Interrupts other than IF may occur only while the PP or an MCP major package is operating. Considering only I-O interrupts, four situations are possible:

1. An MCP I-O unit interrupts an MCP major package.
2. A PP I-O unit interrupts an MCP major package.
3. An MCP I-O unit interrupts the PP.
4. A PP I-O unit interrupts the PP.

These situations may arise in any order and at any rate. In addition, one interrupt may immediately follow another interrupt, in which case response to

the second interrupt must be deferred until the response to the first is complete. An I-O interrupt stacking mechanism is required, and is provided by MCP.

Because of the functions performed by MCP, the MCP program level must have priority over the PP level. The PP must not be allowed to interrupt MCP. Thus, a stacking mechanism is required for PP interrupts occurring at MCP level.

The MCP receptor places a program (MCP or PP) in the auto-stack mode (in which successive interrupts are automatically stacked) when it passes an I-O interrupt to that program. When the receptor receives an I-O interrupt and the program owning the interrupt is auto-stacked, or is, by request, suppressing I-O interrupts ($SIO), the interrupt is stacked in a queue. Two such queues are maintained: one for PP and one for MCP. Considering the SIO mode and the/auto-stack mode as identical, the disposition of I-O interrupts may be tabulated as shown.

When a fixup routine (a program responding to an I-O interrupt) has finished, control must ultimately return to the point of interrupt; however, intervening interrupts must first be unstacked. Since PP interrupts are stacked when the IC is at MCP level, these must be unstacked whenever the level is to change to PP. The return service routine ($RET) sends control to the proper place. Every fixup routine, PP or MCP, and every major package, terminates with $RET. The return routine must also attend to the unpriming of primed major packages. Consider the sequence of events shown in Figure 1.

This sequence of events could be further complicated by adding simultaneous interrupts of different levels, and primed major packages.

In summary, MCP has priority over PP; all MCP tasks must be completed before control is given to PP. All major transfers of control are the result of interrupts: major package to major package, PP to major package, major package to PP, mainstream to fixup, and fixup to mainstream.

Interrupt Owner	Program Level	Program Mode	Interrupt Disposition
MCP	MCP	Auto-stack	Stack for MCP
MCP	MCP	Non-auto-stack	Give to MCP auto-stacked
PP	MCP	Either	Stack for PP
MCP	PP	Either	Give to MCP, change level
PP	PP	Auto-stack	Stack for PP
PP	PP	Non-auto-stack	Give to PP auto-stacked

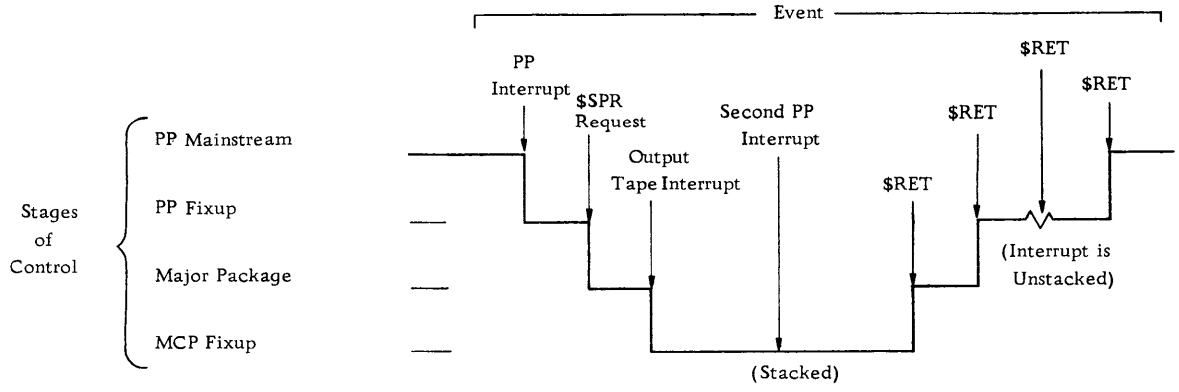


Figure 1. Function of Return Routine

PROGRAM ORGANIZATION

In general, the order of the MCP programs are interrupt control, service routines, then major packages. All of MCP resides permanently on the Permanent Read Only Storage Area of the disk (PROSA). Most of it also resides permanently in upper core storage, the rest being called in from the disk as needed. Some of the part called in from the disk operates in the overlay area at the upper end of the permanent core storage portion, while part of phase 4 of job control operates in PP memory, since it is required only between jobs.

Memory Allocation

MCP occupies five type areas on PROSA. One is the main portion of MCP, the others are major packages called into core when required. They are:

Type Area	Content
11D11MCP	Main portion of MCP
22SCOMD	Command package
22LOADER	Loader and resume load
22$DUMP	Debug and dump packages
22$EOJ	JC4, assign move

An overlay area is provided for major packages which are infrequently used, for example, once per job. Once a program is in the overlay area, it need not be called in again unless it has been overlayed by another program. The overlay area is controlled by the major package fetcher, a subroutine of JC4, and is also used as the buffer for the disk fetch ($FETCH) program.

The core region from the overlay area to the top of memory is available for patches and special installation packages.

The MCP Communication Region

The MCP communication region consists of 14 words from SMCP to SMCP + $(16)_8$ reserved for control bits and words to be used by any part of MCP. The symbols associated with this region are listed in the early pages of the listing. They are further defined in subsequent sections of the Analysis Guide as they are used.

CONTROL OF SYMBOLIC I-O

To utilize its symbolic I-O feature, MCP requires the following:

1. Precise knowledge of the I-O equipment configuration of the machine.
2. A mechanism for translating IOD reference numbers (logical I-O units) into the specific channels and units used to perform I-O requests.
3. A mechanism for translating a channel and unit into an IOD reference number to determine the I-O table of exits for which an interrupt is intended.
4. A mechanism which can differentiate between interrupts resulting from I-O operations performed by the owner of a unit and set-up operations performed by MCP for the owner of the unit.
5. A bookkeeping system to keep track of activities both at the physical unit level and at the logical unit level.

These functions are performed by the I-O control tables. MCP uses six distinct types of I-O control tables:

1. A Channel Status Table (CST) exists for each channel; it contains the operational status for the channel. For single unit channels, the CST also contains the operational status of the unit, and locates the unit area table. For other channels, the CST locates each unit status table.
2. The Unit Status Table (UST) provides the operational status for units on a multi-unit channel. It locates the unit area table. Each such unit has a UST.
3. The Unit Area Table (UAT) serves as an interface between tables oriented toward physical units and tables oriented toward logical units. It contains current information on unit usage, and locates both the file area table and the channel status table. One UAT exists for each physical unit defined by IOD's.
4. A File Area Table (FAT) exists for each logical I-O unit: that is, each unit requested by an IOD card. The FAT contains information in terms of operations on the logical unit. It contains the information from the IOD card, and locates the proper entry in the symbolic I-O location table.
5. The Symbolic I-O Location Table contains one entry for each IOD reference number. The entry locates the UAT and FAT.
6. The Reel Pool Table is used to record the sequence of reels used by a physical unit. It contains the labeling information from the REEL cards.

The channel and unit status tables are originally set up by the initialization program at initial program load. They may be changed by the IOCHANGE command, and are initialized by job control. They are updated by the actuator and receptor.

The unit and file area tables, along with the symbolic I-O location table and reel pool table, are set up by phase 4 of job control, immediately prior to

running a job. The area tables are updated by the actuator and receptor.

The tables may need to be referenced starting either with a channel number (as in the receptor) or with an IOD reference number (as in the actuator). The tables are chained (Figure 2) so that either number serves to locate all related tables. The channel number locates the CST entry, and the IOD reference number locates the proper entry in the symbolic I-O location table. These tables are defined in detail in Appendix B.

The basic characteristics of the tables are as follows:

Table	Size	Frequency	Total Storage	Location
Channel status	1 word	1 per channel	32 plus 1 words for each BX channel	MCP core
Unit status	1 word	1 per unit of multi-unit channels	Varies with I-O configuration	MCP core
Unit area	9 words	1 per unit used	Varies	User's core - protected
File area	7 words	1 per IOD	Varies	User's core - protected
I-O location	1 word	1 per IOD	Varies	User's core - protected
Reel pool	2 words	1 per PP reel	Varies	PP core

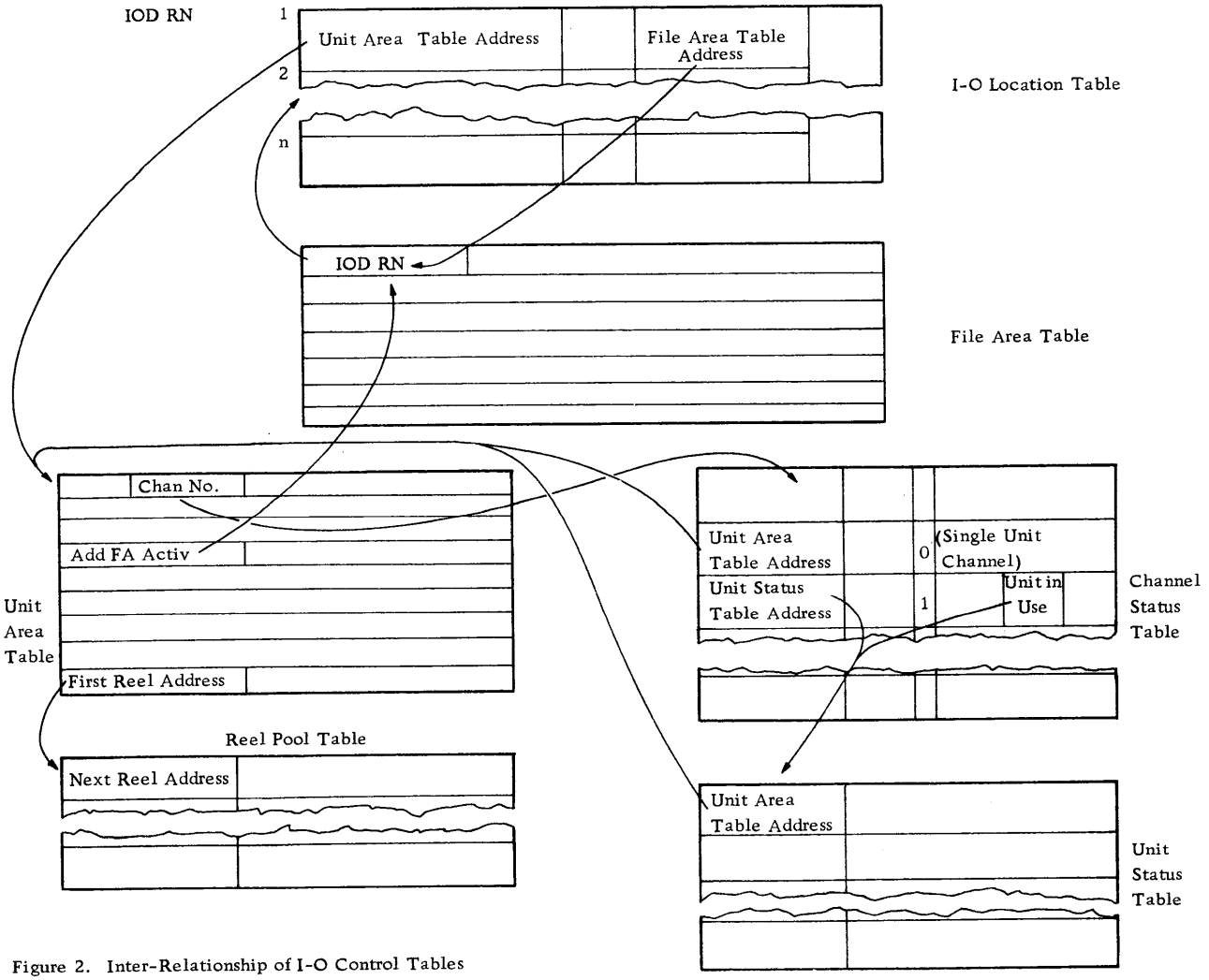


Figure 2. Inter-Relationship of I-O Control Tables

INTERRUPT CONTROL

MCP design is centered around the IBM 7030 interrupt system. All interrupts are taken by MCP. All major transfers of control and all I-O functions depend upon the interrupt system and the MCP interrupt control programs for their success.

MCP divides the IBM 7030 interrupt ($IND) indicators into four categories:

1. The error indicators: 0-3, 5-8, 15-17, 19
2. The maskable indicators: 4, 18, 20, 22-47
3. The I-O indicators: 9-13
4. The IF indicator: 21

The IF indicator is placed in a category by itself because, in addition to indicating a potential error, it is used for major transfers of control.

Figure 3 shows the distribution of interrupts among the MCP programs. All interrupts are received by the MCP interrupt table and from there, control flows to the specified MCP programs.

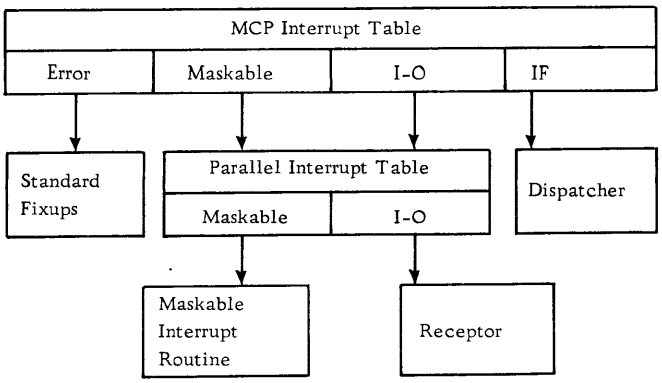


Figure 3. Distribution of Interrupts Among MCP Programs

THE INTERRUPT TABLES

MCP provides the following two interrupt tables:

1. The interrupt table (SIT)
2. The parallel interrupt table (SIPT)

All IBM 7030 interrupts are received by the interrupt table (SIT). It consists of 48 full word instructions of the form SIC,---; BD,---; one for each indicator 0 through 47. Indicator 14 is not used, the interrupt table containing two NOP instructions in the word corresponding to that indicator. The interrupt table branches to one of the following three basic areas of MCP (Figure 3):

1. The standard fixups
2. The parallel interrupt table
3. The dispatcher

I-O interrupts are all handled by a common routine, as are the maskable interrupts, and control is passed to these routines through the parallel interrupt table. This permits retaining the identity of the individual interrupt while branching to a common routine.

The parallel interrupt table consists of 48 full words. All 48 words are not used, because all interrupts do not require use of this table. In positions corresponding to the maskable indicators, the parallel interrupt table contains the full word instruction SIC, ---; B,---. In positions corresponding to the I-O indicators, it contains BZB1, SPSIOI+ (indicator number), KSTORE, providing for entry to the receptor with a bit set corresponding to the interrupt.

The action taken by the interrupt tables for the various interrupts may be tabulated as shown on the following page.

ERROR INTERRUPTS

MCP divides error interrupts into two groups that are both handled by standard fixup routines:

1. Machine errors (indicators 0-3, 5)
2. Program errors (indicators 6-8, 15-17, 19)

Machine Errors

Indicators MK, IK, IJ, EK and CPUS are set by machine errors. The interrupt table, on a machine error, branches respectively to one of five routines all of which have the following form:

```
YFX**   SIC, YMFL
        B, YMFP
        (IQSX)DD(BU,32),     **X
        (AX)DD(BU,32),       **X
        VF,XX.
```

(** denotes the indicator mnemonic)
(XX. denotes the ABEX codes 65.-69.)

Thus, control passes to a common fixup routine, YMFP, with the location of the indicator mnemonic in the address YMFL. Lower registers are stored in the backup buffer (SLRBU) and the level bit (SL) is tested to determine the source of the error.

If the error was caused by PP (SL=0), the level bit is set to MCP, the IC converted to A8 and IQS and placed in the appropriate PP error messages along with the proper indicator mnemonic, and the ABEX code is stored in YAXTYP. The disabled entry to the commentator is used to type the message:

```
$ OPTR PP 'interrupt'. IC = 'location'
CNSL CS TO CONTINUE.
```

Interrupt Category	Interrupt Indicators	Interrupt Table (SIT) SIC IN	BD TO	Parallel Interrupt Table (SIPT) SIC IN	B TO
ERROR	MK, IK, IJ EK, CPUS (0-3, 5, machine error)	YMFIC	YFX** (standard fixup; ** denotes indicator mnemonic)		
	EKJ, UNRJ, CBJ, OP, AD, USA, DS, (6-8, 15-17, 19, program error)	YSFIC	YFX** (standard fixup; ** denotes indicator mnemonic)		
MASKABLE	TS(4, the only asynchronous maskable interrupt)	STIC	SIPT+4.0	SINTO	JWLODE (special TS interrupt routine)
	All other maskable (18, 20, 22-47)	STIC	SIPT+ (indicator number)	WINTOR	WLODE (maskable interrupt routine)
I-O	EPGK, UK, EE EOP, CS (9-13)	STIC	SIPT+ (indicator number)	Not saved. Bit SPSIOI + (indicator number) set.	KSTORE (receptor)
IF	IF (21)	STIC	SIFAE (IF analyzer)		

If the console channel signal is given, a test is made for the MK interrupt. An MK interrupt at PP level causes core storage to be searched for the MK location. The location is restored to prevent possible parity errors when the system continues. ABEOJ is primed and MCP mask and boundaries are selected. Control is then given to the short message routine, which writes the following message on the output tape:

'interrupt' INTERRUPT AT LOCATION 'location'

Then the ABEOJ is unprimed causing termination of the PP and the system continues. (See the dispatcher return routine, the short message routine, and the ABEX routine.)

If the error occurred at MCP level, the contents of the IC in IQS and the indicator mnemonic are placed in the MCP error message:

$OPTR MCP 'interrupt'. IC = 'location'
IPL REQD.

After using hardware I-O instructions to write the error message, the system halts with a BD, $ at YMCPHP and must be reinitialized.

Program Errors

Incorrect program action may set indicators EKJ, UNRJ, CBJ, OP, AD, USA, and DS. The interrupt table, on a program error, branches respectively to one of seven routines all which have the following form:

```
YFX**    SIC, YPREL
         B, YPREFX
         (AX)DD(BU,32),     **X
         (IQSX)DD(BU,32)    **X
         VF, XX.
```

(** denotes the indicator mnemonic)
(XX. denotes the ABEX code 70.-76.)

Control passes to a common fixup routine, YPREFX, where the location of the A8 mnemonic is contained in YMFL.

If PP caused the interrupt, the lower registers are stored in the backup buffer (SLRBU), the program error indicators are cleared, and the EXE and IF indicators are reset to prevent an unanticipated loss of subsequent control. The indicator mnemonics are

placed in PP and MCP error messages and the level bit is retested and set to MCP.

If the error was caused by PP, the IC saved in YSFIC is converted to A8 and placed in the PP error message, the ABEX code is stored in YAXTYP, the MCP mask and boundaries are selected, the ABEOJ is primed with control given to the short message routine that writes the error message. The ABEOJ is then unprimed and the PP terminated.

If the error was caused by MCP, system operation cannot be continued. Hardware I-O instructions are used to write the error message using the same routine as for an MCP machine error. The system halts with a BD, $ at YMCPHP.

MASKABLE INTERRUPTS

The maskable indicators are used at the option of the problem program. They are:

Indicator	Type
4	Time Signal (TS)
18	Execute Exception (EXE)
20	Data Fetch (DF)
22-34	Result Exceptions
35-38	Flagging
39-40	Transits
41-47	Program

Time signal and execute exception are permanently masked on, but to give the problem program more control, they are treated as pseudo-maskable (see the description of the maskable interrupt routine, and the EXE interrupt). The IF indicator, the mask of which must always be 1, is not included in this group.

Interrupts caused by these indicators are routed via the parallel interrupt table to the maskable interrupt routine, or, in the case of TS, to a special entry in the maskable interrupt routine. Control then passes to a program table of exits (PTOE) in either PP or MCP. In the case of MCP, the standard fixup routine is entered from the PTOE, and control returned to the point of interrupt via the return after maskable interrupt routine.

The Program Table of Exits (PTOE)

When any maskable interrupts are masked on, fixup routines must be available to handle the resulting interrupts. The problem program has the option of providing special fixup routines or of using the MCP standard fixup routine, or a combination of both.

If the problem program is to use special fixup routines, it must provide MCP with a program table of exits (PTOE) to these routines. The refill field (RF) of index register 15 ($15) is reserved for this purpose. If the problem program is to use any special fixup routines for maskable interrupts, it must load the RF of $15 with the first word (18 bit) address of a PTOE. The RF of $15 will be zero when MCP initially gives control to PP. It will also be zero when MCP gives control to any PP I-O table of exits. It is the responsibility of the PP that $15 RF be correct whenever PP has control. PP may provide as many special PTOE's as are necessary, changing $15 RF when desired.

The PTOE reserves storage for the IC, the indicator register and the mask register ($IND and $MASK). It consists of 4+k full words as follows:

Word	Content
1	Saved IC
2	Saved $IND
3	Saved $MASK
4	Pattern word, containing k ones
4+1	First instruction for the first special interrupt routine.
4+i	First instruction for the ith special interrupt routine.
4+k	First instruction for the last special interrupt routine.

The pattern word indicates whether the PP furnishes a special interrupt routine or desires to use the standard fixup routine. There is a one-to-one correspondence between positions in the indicator register and in the pattern word. When a pattern word bit is set to one, it means that a special interrupt routine is to be used for the corresponding interrupt. The first instruction for the routine must be in the PTOE at word 4+i, where the interrupt indicator register position corresponds to the ith one in the pattern word. Only bits 4. 18. 20, and 22 through 47 may be set to one in the pattern word. All others are treated as zero. The PP must set the appropriate mask bits in the mask register ($MASK). The IF mask bit, 21, must always be left as one.

When MCP enters the problem program PTOE, the IC, $IND, and $MASK are stored in the word specified. The remaining low registers are unchanged. $IND is cleared to zero, and $MASK is cleared to zero except for the IF bit and those permanently on.

Maskable interrupts $TS and $EXE are considered as masked off unless a PP PTOE is specified, and the corresponding pattern bits are on.

The MCP PTOE and the Standard Fixup

Since PP may elect to take maskable interrupts without providing special interrupt routines, MCP is prepared to take any maskable interrupt. MCP provides its own PTOE, which is entered when PP does not provide a PTOE, or when the appropriate pattern bit in the PP PTOE is off.

The MCP PTOE is located at SFPTE. It has the same structure as the PP PTOE, and contains ones in the pattern word corresponding to each of the maskable interrupts. The last 27 full word entry points are all of the form

```
LVI, $1, YIN**
B, YMSF
(** denotes the location of the A8 indicator
  mnemonic)
```

NOTE: The first two interrupts, TS and EXE, are treated as masked off, and the PTOE branches to the return after maskable routine (WRAM).

Thus, control passes to a standard fixup routine with the location of the A8 indicator mnemonic in $1. The maskable interrupt standard fixup routine, YMSF, saves the low registers in SLRBU, and puts the indicator mnemonic and the IC in a message skeleton. The program changes the level to MCP and branches to the system short message routine to print the following message on the output tape:

'interrupt' INTERRUPT AT LOCATION 'location'

Since the level bit (SL) is set to MCP, the return routine will return to PP rather than to the short message routine. (See the description of the short message routine and the return routine.)

If PP has entered the standard fixup routine 50 times, the dispatcher error control routine (SDISP) is entered with error code 13 in $14. This will result in ABEOJ for the PP.

The Maskable Interrupt Routine

From the parallel interrupt table, the maskable interrupt routine is entered disabled at WLODE for all maskable interrupts except TS. It is entered at JWLODE when a TS interrupt occurs. The routine selects either the PP or the MCP PTOE, performs the necessary set-up, and enters the table.

TS Maskable Interrupt

The TS interrupt is the only maskable interrupt occurring asynchronously with the program in control, therefore, the system modes must be determined before the interrupt is released to a PTOE. If TS interrupts MCP, the TS is stacked until control is returned to PP. If TS interrupts a PP I-O fixup (PP auto-stacked), the TS will be stacked until control is returned to PP mainstream, unless PP has provided a PTOE effective in auto-stack mode. (Recall that MCP sets $15 to zero before entering a PP I-O table of exits.) If TS occurs while a PP PTOE has control (SSPFIX = 1), it will be stacked, requiring that PP leave the PTOE with a $RAM pseudo-op (return after maskable) to unstack TS interrupt.

The low registers are saved in STLR upon entry into the maskable interrupt routine (Figure 4). If the interrupt was TS (entry at JWLODE), the system mode control bits are tested and the TS either stacked or given to a PTOE. It will be given to a PTOE only if it occurs in PP main stream, or in PP auto stack with a PP PTOE provided. Otherwise, it is stacked as follows:

TS Occurs in	Stacked by Setting	Unstacked by
PP special fixup (PP PTOE in use)	STSBIT	$RAM
PP auto-stacked (no PP PTOE provided in A/S)	SLRPP+8.4	$RET
MCP	SLRBU+8.4	$RET

The first case (PP special fixup) is provided because of the asynchronous nature of the TS interrupt, to keep an uncontrollable maskable interrupt from occurring during a maskable interrupt fixup. To prevent this, MCP stacks the TS and releases it to the PTOE when the PP executes $RAM. The second case (PP auto-stacked) provides for holding the TS interrupt until return to PP mainstream, unless PP has specifically provided for taking the interrupt by loading $15 RF in the I-O fixup routine. The third case is simply an exercise of MCP priority over PP. As with any other PP interrupt, the TS interrupt is stacked until MCP returns control to PP.

In the first case, the stacking mechanism is simply a bit (STSBIT) reserved for that specific purpose. In the last two cases, the stacking mechanism involves the basic return logic in MCP. The buffers SLRPP and SLRBU are used to save PP low registers when transfer of control among programs is involved. SLRPP contains PP mainstream registers while PP is in an I-O fixup, or when MCP has control after having interrupted the PP I-O fixup. In any case where the program level is changed from PP to MCP, the PP registers at the point where the level changed are saved in SLRBU. Thus, SLRBU may contain registers from either PP mainstream or a PP I-O fixup. When the level is returned to PP, it always returns

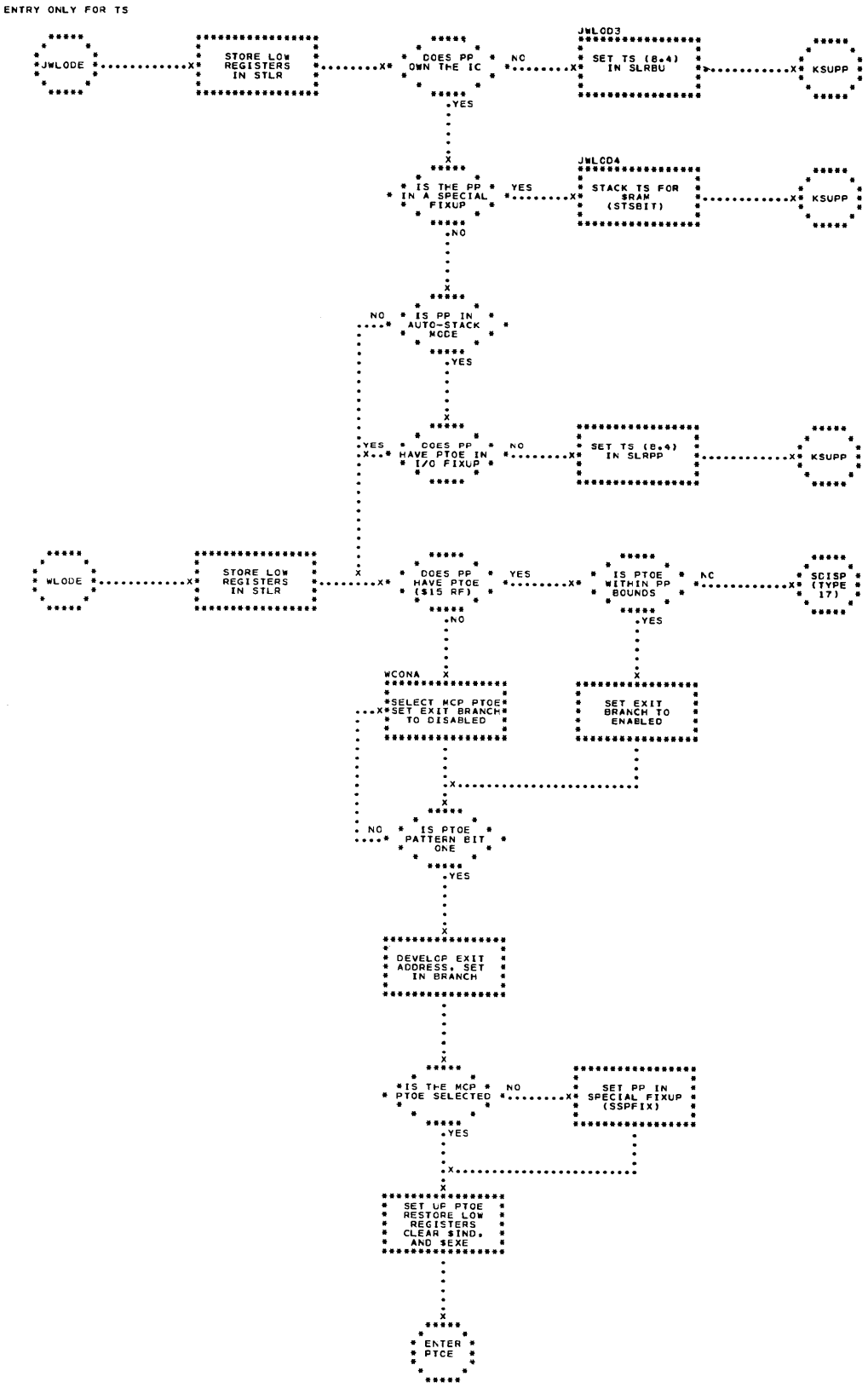


Figure 4. Maskable Interrupt Routine

via SLRBU. Return via SLRPP occurs when a PP I-O fixup returns to PP mainstream.

When the return routine is returning to PP via either SLRBU or SLRPP, it examines bit 8.4 of the buffer to determine if a TS has been stacked. If so, it restores the lower registers and exits to the TS entry in the parallel interrupt table, faking the TS interrupt. Thus, the TS eventually gets unstacked while PP has control.

Other Maskable Interrupts

Maskable interrupts other than TS, together with TS interrupts being taken, are processed by the routine which starts at WLODE (Figure 4). If $15 RF is zero (no PP PTOE provided), the routine selects the MCP PTOE and sets the exit branch to disabled. If $15 RF is not zero, it is checked for being in PP boundaries, and the exit branch set to enabled. If the PTOE is outside PP bounds, control is given to dispatcher error control (SDISP) with error code 17 in $14.

The program checks the pattern word of the PTOE selected. If the required bit is off, it loops back to select the MCP PTOE and change the exit to disabled. (The MCP PTOE pattern word has a one for every maskable interrupt.) It then computes the exit address and stores it in the preset BD or BE. If the required bit is on in the PP PTOE selected, a bit is set (SSPFIX) to indicate to MCP that PP is in a special fixup. The IC, $IND, and $MASK are stored in the PTOE, $EXE is reset (see the following description of the EXE interrupt), the used registers are restored, and the PTOE is entered.

The EXE Interrupt and Combinations

The EXE interrupt, although permanently masked on, is treated as pseudo-maskable by MCP. Since 18 other interrupts have priority over EXE, the simultaneous occurrence of $EXE and the higher priority indicators must be considered. Not all of these simultaneous occurrences are significant, in view of the following analysis:

1. $EXE is off if an interrupt is caused by DS or a lower priority indicator.
2. $EXE will not be set while MCP has control.
3. Interrupts caused by machine errors (0-3, 5) or program errors (6-8, 15-17) result in BD, $ or ABEOJ, so the only concern is that $EXE be off if ABEOJ is to be executed.
4. The remaining interrupts with higher priority than EXE are TS (4) and the I-O (9-13).
5. Since EXE is treated as pseudo-maskable, it is considered as masked off unless there is a PP PTOE with the EXE pattern bit on.
6. If $EXE is on when a higher priority interrupt occurs, it cannot be allowed to remain on and cause the interrupt on any subsequent BE; the BE must be to the point of original interrupt (where $EXE was set). If this will not be the case, $EXE must be turned off, and a mechanism established to fake the interrupt at a proper time.

The combinations of interrupts of concern are as follows: EXE interrupt by itself, TS with $EXE set, I-O with $EXE set, and TS with both I-O and $EXE set. These may occur only in PP; however, they must be considered separately for the three levels of PP: mainstream, auto-stacked, and special fixup.

EXE Interrupt Alone

If the EXE interrupt occurs by itself (no higher priority indicators) in PP mainstream, it is either taken by the PP PTOE (if one exists with the EXE pattern bit on) or is ignored and return is to the point of interrupt in PP.

If the EXE occurs in a PP special fixup, it is treated as though it were mainstream. The PP has control of PTOE specification and should be prepared for this eventuality.

If the EXE occurs in PP auto-stacked, it is again treated as though it were mainstream. If PP may cause EXE in an I-O fixup, then the fixup routine must provide a PTOE if the EXE is to be taken.

TS Interrupt, $EXE Set

If a TS interrupt occurs with $EXE set, the handling of the EXE is influenced by the disposition of the TS. If the TS is stacked (for instance with PP in auto-stack with no PTOE) or in special fixup, $EXE may remain on and be allowed to occur on return to the point of TS interrupt. If the TS is taken, the $IND is saved with the EXE bit set in the PTOE (MCP or PP), $EXE itself turned off, and the EXE interrupt faked when $RAM is executed after the TS fixup (MCP or PP).

I-O Interrupt, $EXE Set

When an I-O interrupt occurs, the receptor will either enter the appropriate I-O table of exits or return to the point of interrupt with the I-O interrupt stacked, depending on the modes (SIO or A/S) of the programs and the owner of the interrupt. If the receptor is to return to the point of the interrupt, then $EXE may remain on to occur when the return is attempted. If a table of exits is to be entered, the PP low registers are saved in SLRPP (including $IND with $EXE on), and $EXE must be turned off to enter the I-O fixup enabled. The EXE will be detected by the return routine ($RET) when attempting to restore low registers from SLRPP, and an EXE interrupt faked at

that point, which reduces to the case of an EXE interrupt by itself.

TS Interrupt, an I-O Indicator On, $EXE On

If a TS interrupt occurs and an I-O indicator and $EXE come on simultaneously, the TS interrupt is processed in the normal manner. If a PTOE is to be entered, $EXE is turned off (on in the PTOE), and the situation is reduced to a simultaneous TS and EXE with a subsequent I-O. If the TS is to be stacked, the I-O occurs on attempt to return to the point of interrupt, and the situation is that of simultaneous I-O and EXE.

Return After Maskable Interrupt

The return after maskable interrupt routine is entered from the MCP PTOE or when the pseudo-op $RAM is executed. Its function is to return to the point at which a maskable interrupt occurred, after first releasing a stacked TS or EXE interrupt. It operates entirely disabled.

When $RAM is executed, the routine is entered from the identifier at WRAMPP (Figure 5). The current PP PTOE specified by $15 RF is compared with the PP boundaries, and if it is not within PP boundaries, dispatcher error control is entered (SDISP) with error code 17 in $14. The PP special fixup bit is reset (SSPFIX), and STSBIT tested for a stacked TS interrupt. If a TS occurred while the special fixup was in progress (STSBIT one), the saved IC is stored in STIC, the used registers are restored, and the TS interrupt faked by a branch to the parallel interrupt table at SIPT+4.0.

If STSBIT is off, bit 18 of the saved indicator register is tested to determine if an EXE had been stacked due to the simultaneous occurrence of TS. If so, conditions are restored and control returned enabled to the point at which the maskable interrupt occurred.

When the routine is entered at WRAM by the MCP PTOE as a result of a TS or EXE interrupt which was logically masked off, the MCP PTOE address is selected, and control is given to the main line of the routine at the point of the EXE test.

I-O INTERRUPTS

The I-O interrupts correspond to the five indicators: $EPGK, $UK, $EE, $EOP, and $CS. When any of these interrupts occur, the MCP receptor is entered. The receptor must identify the owner of the interrupt and either pass it to the proper program or stack it, according to program mode.

Channel signal (CS) interrupts from the console are given by the receptor to the conceptor because the owner of a CS from the console cannot be identified until a read is performed. The conceptor is used to perform this identification.

The Receptor

The receptor is entered disabled from the parallel interrupt table with a bit set in SPSIOI corresponding to the interrupt. The functions of the receptor are:

1. To identify the interrupt and its owner and decide whether to stack the interrupt or pass it to the proper program.
2. To stack all channel signal interrupts occurring simultaneously with other interrupts. CS interrupts will be treated at a later time as separate interrupts.
3. To pass console signals to the conceptor.
4. To update tape file and disk arc records in the control tables.

An I-O interrupt may be owned by PP, an MCP major package, or by a set-up operation being performed by a disabled MCP routine (e.g., change density). If it is a set-up operation, control will be given to the appropriate routine at the address specified in SRETAD in the unit area table, UAT. (See Control of Symbolic I-O.)

Interrupts caused by non-set-up operations will be stacked if the owner is in a stack I-O mode, or if the owner is PP and the system is at MCP level. Stacked interrupts are eventually unstacked by the dispatcher return routine, which enters the receptor by way of the unstack subroutine.

The receptor is entered at KSTORE from the parallel interrupt table, and also at three other points in the following circumstances:

Entry Point	Circumstances
KUNSTC	The return routine is unstacking an interrupt. $RIO has been given, and stacked interrupts must be unstacked.
KQIN	Entered by the conceptor to fake a CS.
KGATE	Entered by the actuator and conceptor to fake an interrupt under certain circumstances.

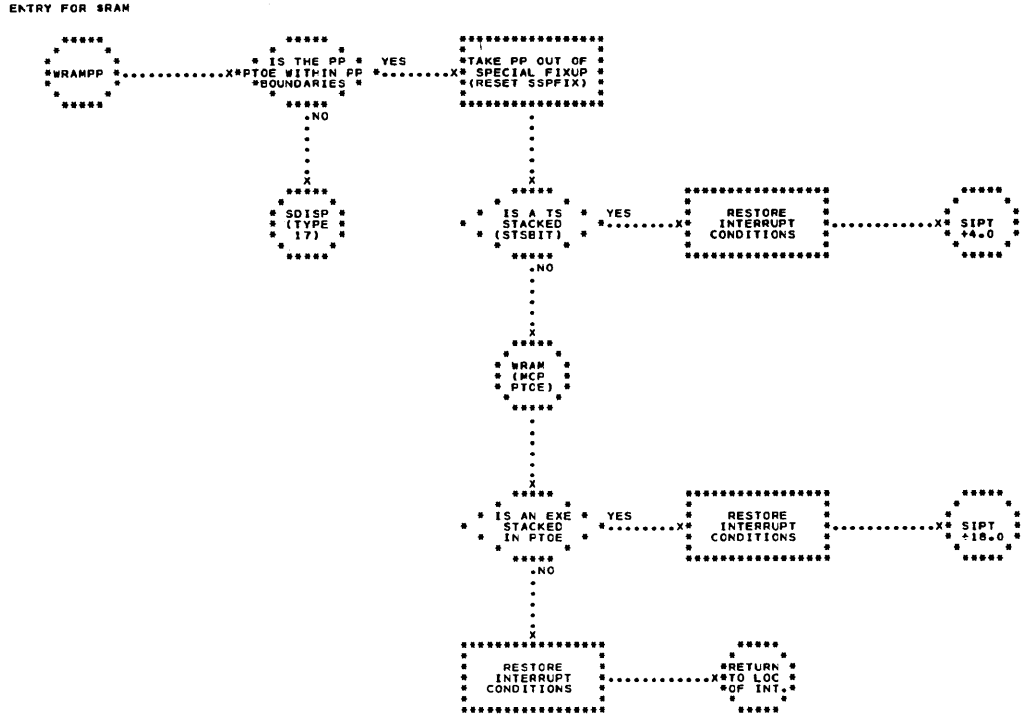


Figure 5. Return After Maskable Routine

The exits from the receptor are as follows:

Exit To	Circumstances
The address specified in SRETAD in the UAT.	The set-up I-O bit (SSETUP) is on in the unit status table (UST), indicating that it was a set-up operation.
PCONE1 (the conceptor)	The interrupt is a channel signal from the console.
The I-O table of exits specified in the FAT	The interrupt owner is MCP, or PP at PP level, and is not stacking I-O.
KSUPP (service-op return)	The interrupt is stacked, because the owner cannot take it now. Either OP, AD, USA, or DS indicators occurred simultaneously with the I-O interrupt, and are being allowed to interrupt. An EOP interrupt occurred for an operation which had been requested in the SEOP mode.
SDISP (dispatcher error control)	Type 5 error, a PP PTOE is out of PP bounds. Type 78 error, there is no more room in the interrupt queue. Type 76 error, a locate or CCW attempted by the receptor has been repeatedly rejected. Type 77 error, an interrupt was not found in the queue when the queue count was non-zero.
RRR (verify label)	A label verification had failed. The remount is handled enabled and asynchronously. The channel signal is handled as a setup interrupt and control is returning to the verification process.

The Receptor Main Flow

The receptor (KSTORE, Figure 6) saves the low registers in STLR, clears the I-O indicators, and computes the address of the channel status table. If the interrupt is CS alone, and occurs in a multi-unit channel, control is given to KSIGNL to find the unit. If the interrupt was not CS, but CS is on, subroutine KCSIN is used to stack the CS and set KSUPP to unstack it. In either case, address of the UST is obtained.

If the channel is a single unit channel, the unit status table (actually the CST) address is set up (KSINGL). If the unit was not assigned, it is assumed that the interrupt was a stray CS from the card reader, printer, or punch, and an exit is made via the service op return routine (KSUPP).

NOTE: The disk and the console are considered as multi-unit channels, since they may have more than one logical unit.

Whether a single or a multi-unit channel, the unit and file area table addresses are set up at KDOOL. The index registers at this point locate the control tables as follows:

$10VF	Address of CST
$11VF	Address of UST
$12VF	Address of UAT
$13VF	Address of FAT

If the set-up bit is on (SSETUP) in the UST, control is given to the address specified in SRETAD in the UAT. Otherwise, $1 is set for an ultimate branch to KNORM at KLEVEL, and control passes to KGATE after performing the following bookkeeping:

1. If the operation is read or write, CCW into SCCW in the FAT.
2. If tape, update the file count (SFILEK) in the UAT according to the interrupt and the I-O operation.
3. If disk, update the current arc address (SCUARC) in the FAT, and the located arc address (SARCAD) in the CST.

The receptor is entered at KGATE from the actuator and the conceptor with the index registers set as though KSTORE had been entered. The actuator enters the receptor either with the index registers set as though an EPGK had occurred (when, for example, a read request is received for a file protected tape) or with EOP faked when a $REL is requested. The conceptor enters the receptor when a console read is given after a console channel signal had been passed to the user. Since the read had already been done by the conceptor, it is necessary to fake an EOP to the using program.

At KGATE (Figure 7) the I-O indicators are saved in the FAT, and the status indicators are cleared in the UST. The release and SEOP bits are examined in the UST. If the release bit is on, the code at W23K is used to return to KGATE with an EOP faked for the release. If the SEOP bit is on, a check is made for indicators other than EOP (at KFREE) and if none are on, the service op return routine is entered (KSUPP). If others are on, or if the SEOP bit was not on, control passes to KQIN.

At KQIN, the interrupt is stacked in the interrupt queue (SQUE), which consists of 128 nineteen-bit

fields containing the 18 bit address of the FAT and a bit identifying the owner of the unit. PP and MCP interrupts are intermixed in the same queue. However, a separate count of stacked interrupts is maintained for the two levels and is stored in the value field of the second half word (SQK) of the program status table (SPROGS) for the level. The unit suppressed bit (SUNSUP) is set, and control given to either KNORM or KHEX, according to $1.

At KNORM, (Figure 7), a decision is made whether or not to unstack the interrupt. If PP is interrupting MCP, the interrupt remains suppressed and exit is made via the service op return routine (KSUPP). If MCP is interrupting PP, the unstack routine (KUNSTC) is entered after moving low registers to the back-up buffer. If the present program level is the same as that of the interrupt owner, and the program is not in $WAIT status or in auto-stack or SIO mode, the interrupt will be unstacked. If the program is in $WAIT status, and is auto-stacked, the service op return routine is entered. The $WAIT routine will unstack this interrupt if it is the one awaited. In all cases where the interrupt is to be unstacked, low registers are saved in the appropriate buffer. (See description of the return routine.)

The unstack routine returns to KINTTY, where the proper mask and boundaries are set up, and the program put in the auto-stack mode. If a PP PTOE is to be entered, it is checked against the PP bounds. The unit suppressed bit is reset, and the table of exits is set up. If either OP, AD, USA, or DS are on, KSUPP is entered to allow the error interrupt. Otherwise, the maskable indicators are cleared, $15 is cleared, $EXE is reset, and the PTOE is entered enabled according to the highest priority I-O indicator that was on.

The Channel Signal Search

Control is given to KSIGNL (Figure 9) when a channel signal is received from a multi-unit channel. (The console and disk are considered logically as multi-unit channels.) If the CS is from the console, control is given to the conceptor at PCONE1. If from tape, control passes to KHEX. The function of the code at KHEX is to determine what change, if any, has been made in the ready status of the tape units on that channel. If the equipment is neither tape nor console, control is returned to the main flow at KKK.

The channel signal will be considered valid only for tape units in a mount or rewind status (SMOUNT, SREW in the UST). For all such units on the channel, the unit will be selected, the control word copied, and the ready bit tested. If the unit is now ready, the UST is updated, and the receptor takes one of the following four courses of action depending on the status bits:

1. SIMNT and SMOUNT bits on -- This signifies initial use of this tape unit. The CS is discarded for a PP unit but delivered to MCP if the unit belongs to the input or output programs.
2. Either SMOUNT or SREW bits on -- This signifies the issuance of a $UNLD or $REW. The CS for the unit is stacked starting at KQIN and control is returned to KHEX.
3. SIMNT and SREW bits on -- The MCP unassign program has issued a SFREE,which in turn issued $REW. This combination is used internally for improved tape handling. Once the bits have been zeroed, the CS is discarded by going directly to KHEX.
4. SMOUNT and SSETUP bits on -- This is a situation where a verify label remount has occurred. If the verify buffer is in use when the remount occurs, an indication of the remount is saved in a queue. If the buffer is not busy, control will go to the verification process. When the verify buffer becomes not busy (signaled by a tape I/O pseudo-op branch to RKSUPP) the unqueuing of remount verifies is initiated on an MCP first basis.

After testing all the units on the channel, control is returned to KNORM or KSUPP according to whether or not a change in ready status was found. First, however, the unit originally in select on the channel (SUNIT in the CST) is relocated.

Search and Unstack

The search subroutine (KSERCH, Figure 11) is used by $WAIT, the return routine, and the receptor to search the interrupt queue for a particular FAT address (the one being waited). It uses the linkage

```
LVI, 1, $+1.0
B, KSERCH
(not found return)
(found return)
```

with the desired FAT address in $13VF. If it finds the address in the queue, it repacks the queue (KPACK) and reduces the proper queue count.

The unstack routine (KUNSTC, Figure 11) is entered by the receptor, the return routine, and the $RIO routine to release interrupts. It will try to obtain the first MCP FAT address from the queue, otherwise the first PP FAT address. Then it enters the search routine in the vicinity of KPACK with $1 set to KINTTY-.32, to force return to KINTTY after repacking the queue. If the queue is empty, the unstack routine returns to KSUPP.

Channel Signal and Console Unstack Control

If the CS indicator is on when a higher priority I-O interrupt occurs for a multi-unit channel, the channel signal is deferred. The higher priority interrupt is

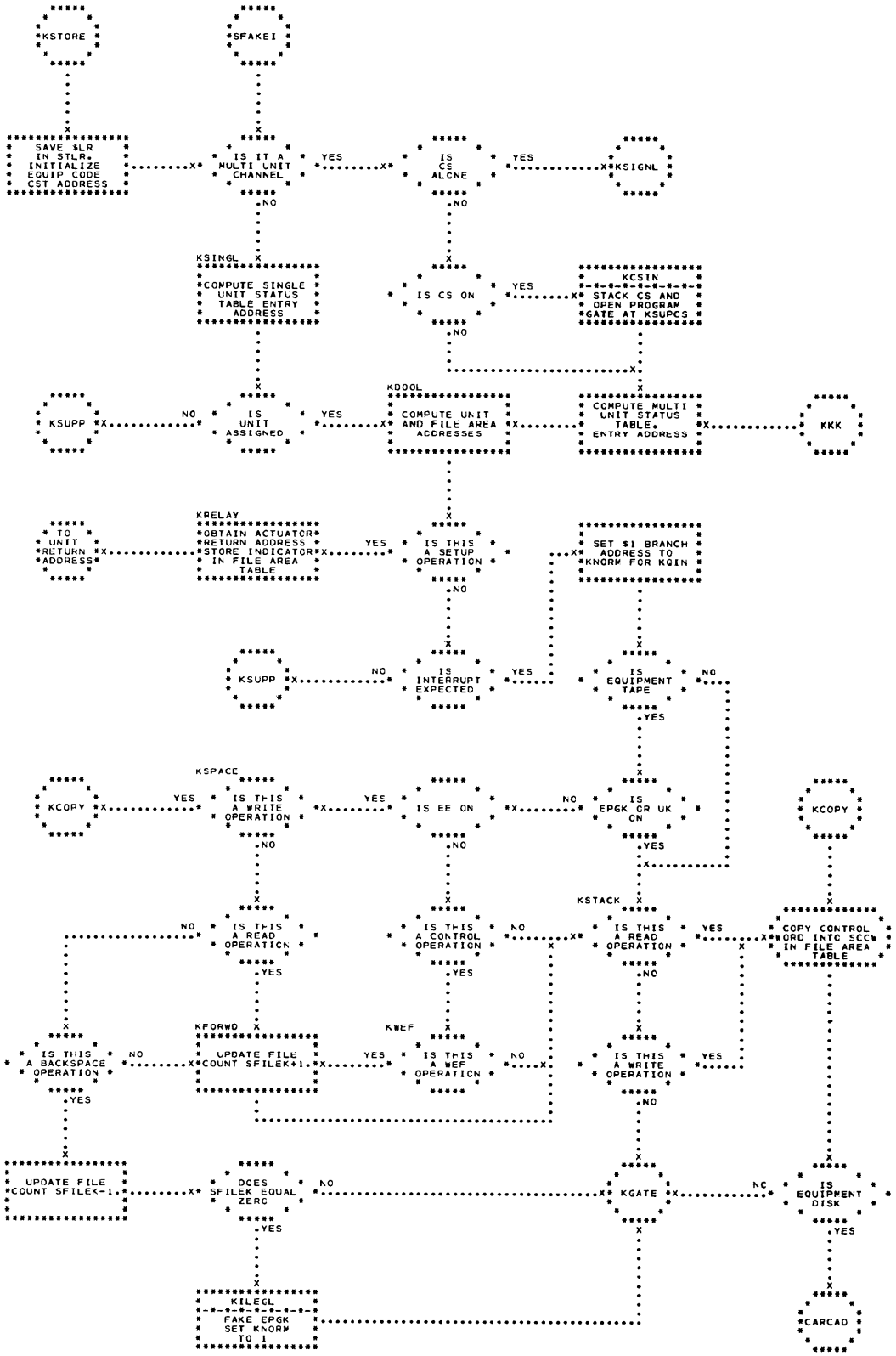


Figure 6. Receptor - Chart 1

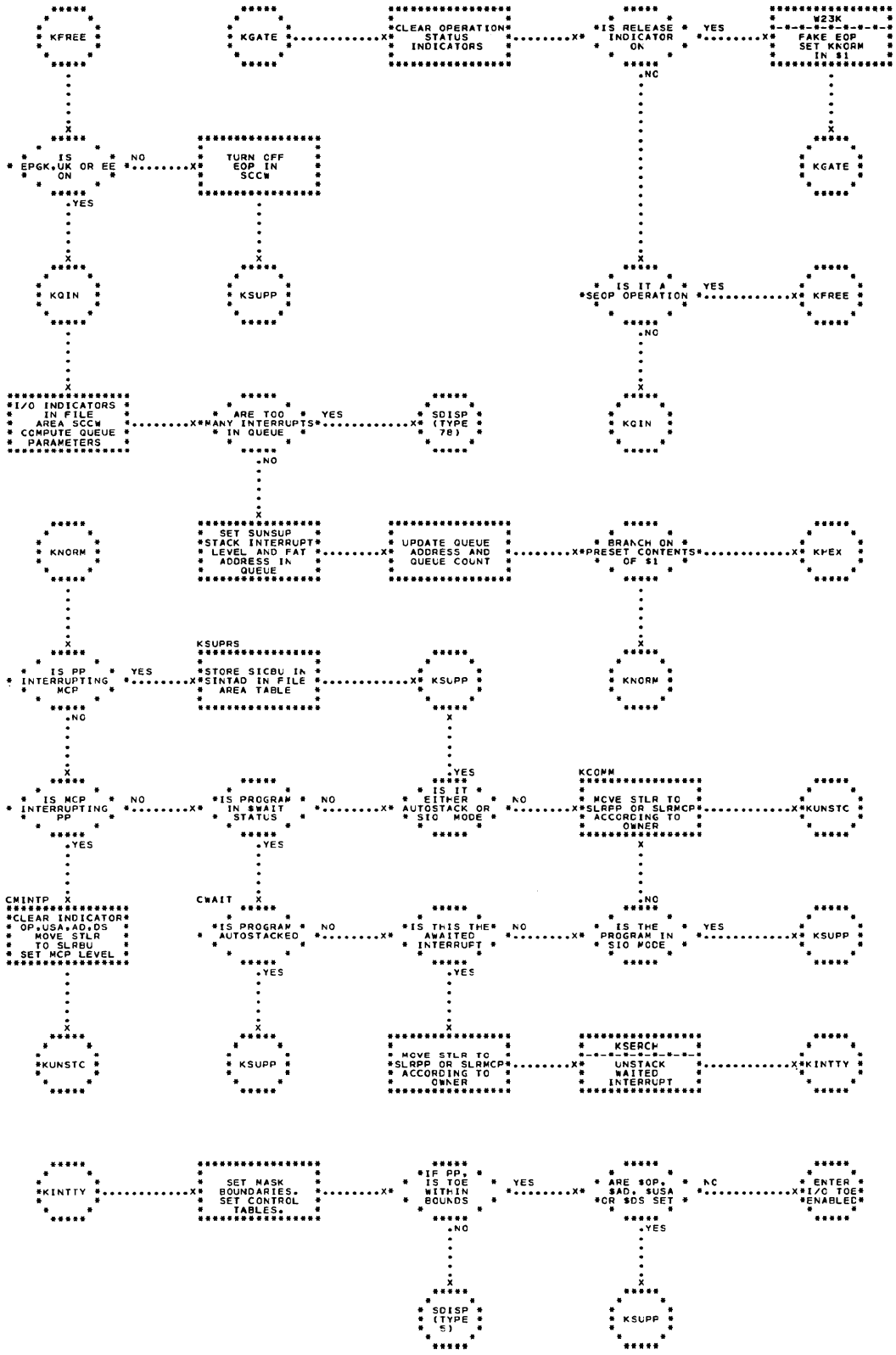


Figure 7. Receptor - Chart 2

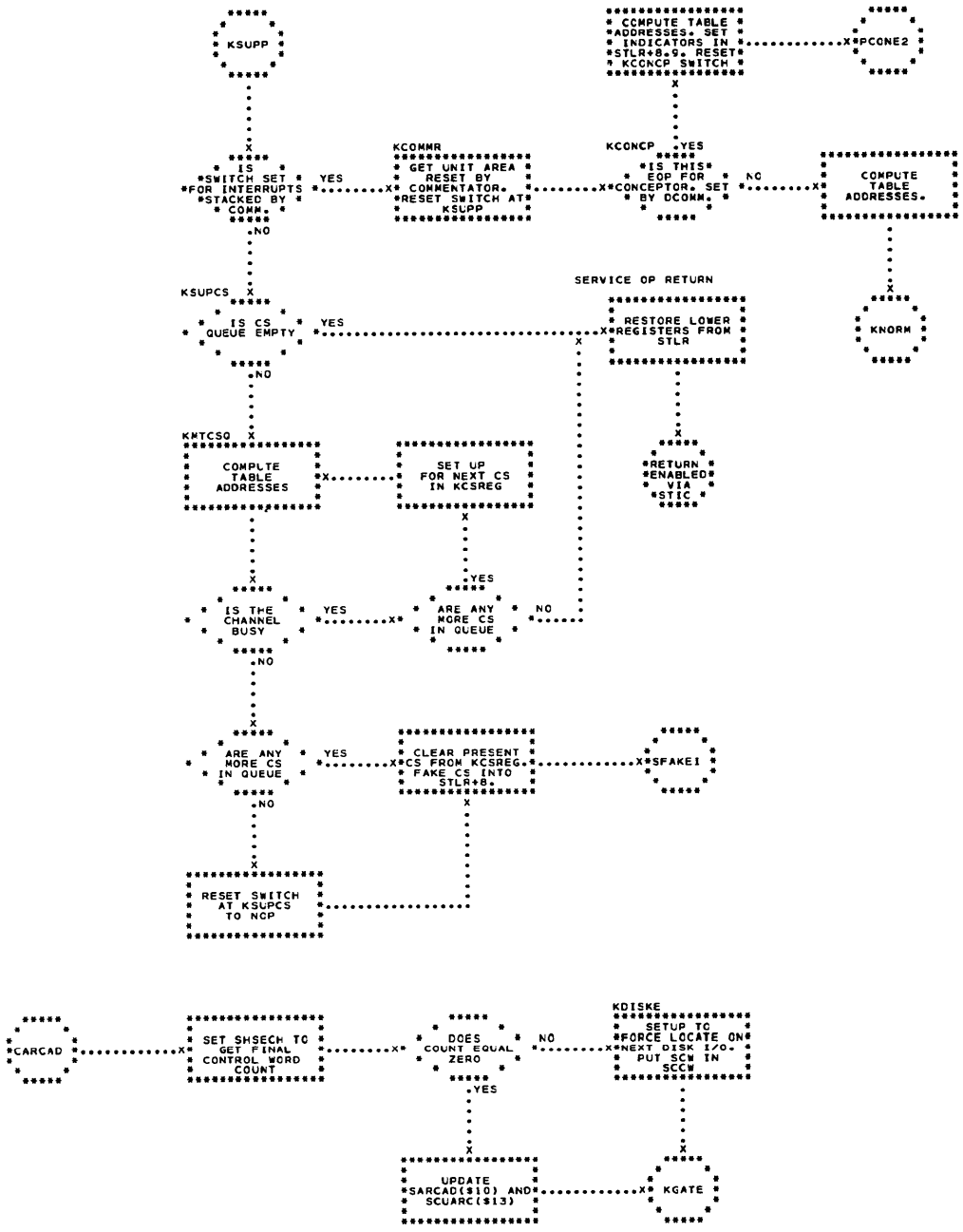


Figure 8. Receptor - Chart 3

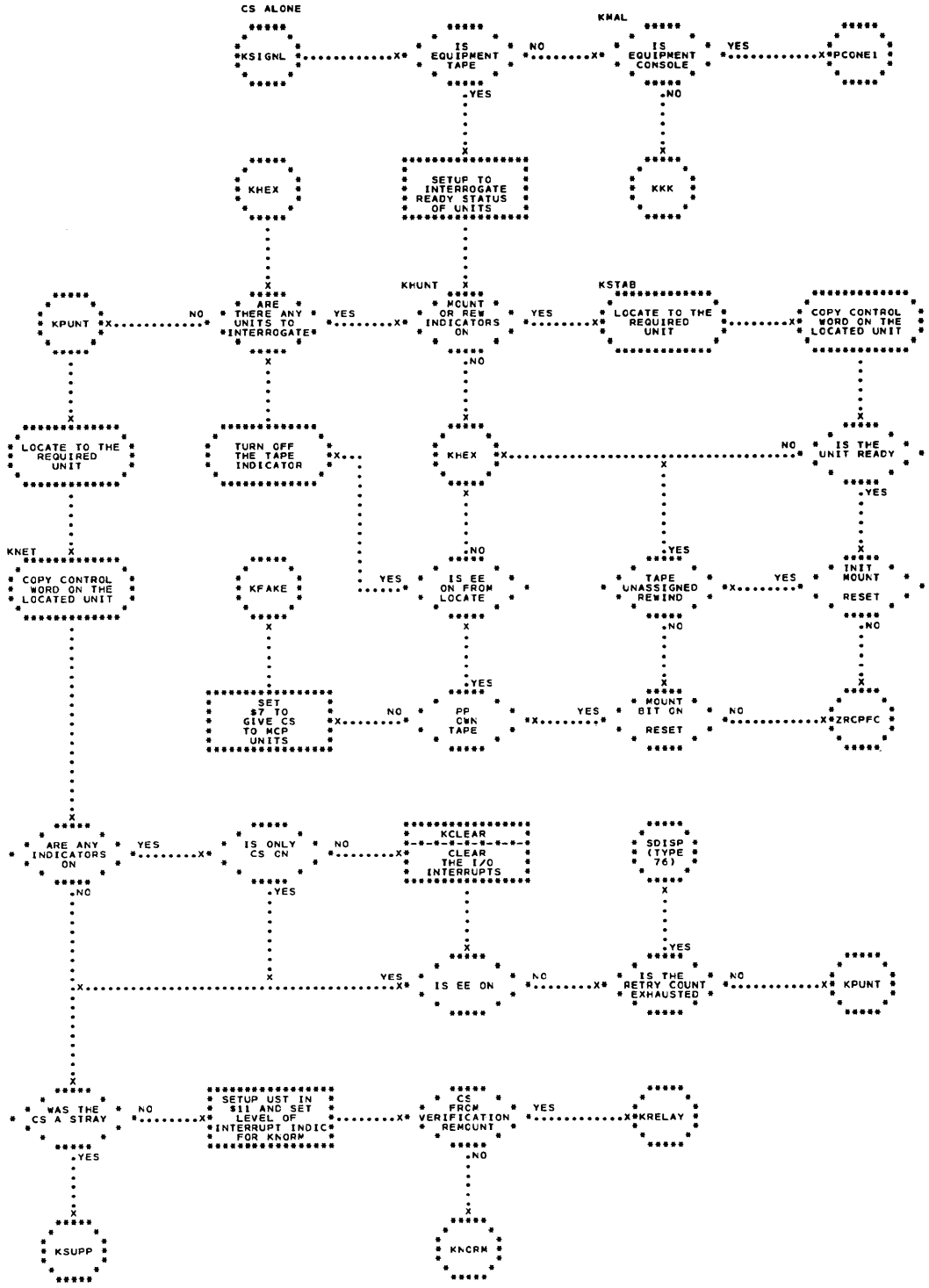


Figure 9. Receptor - Chart 4

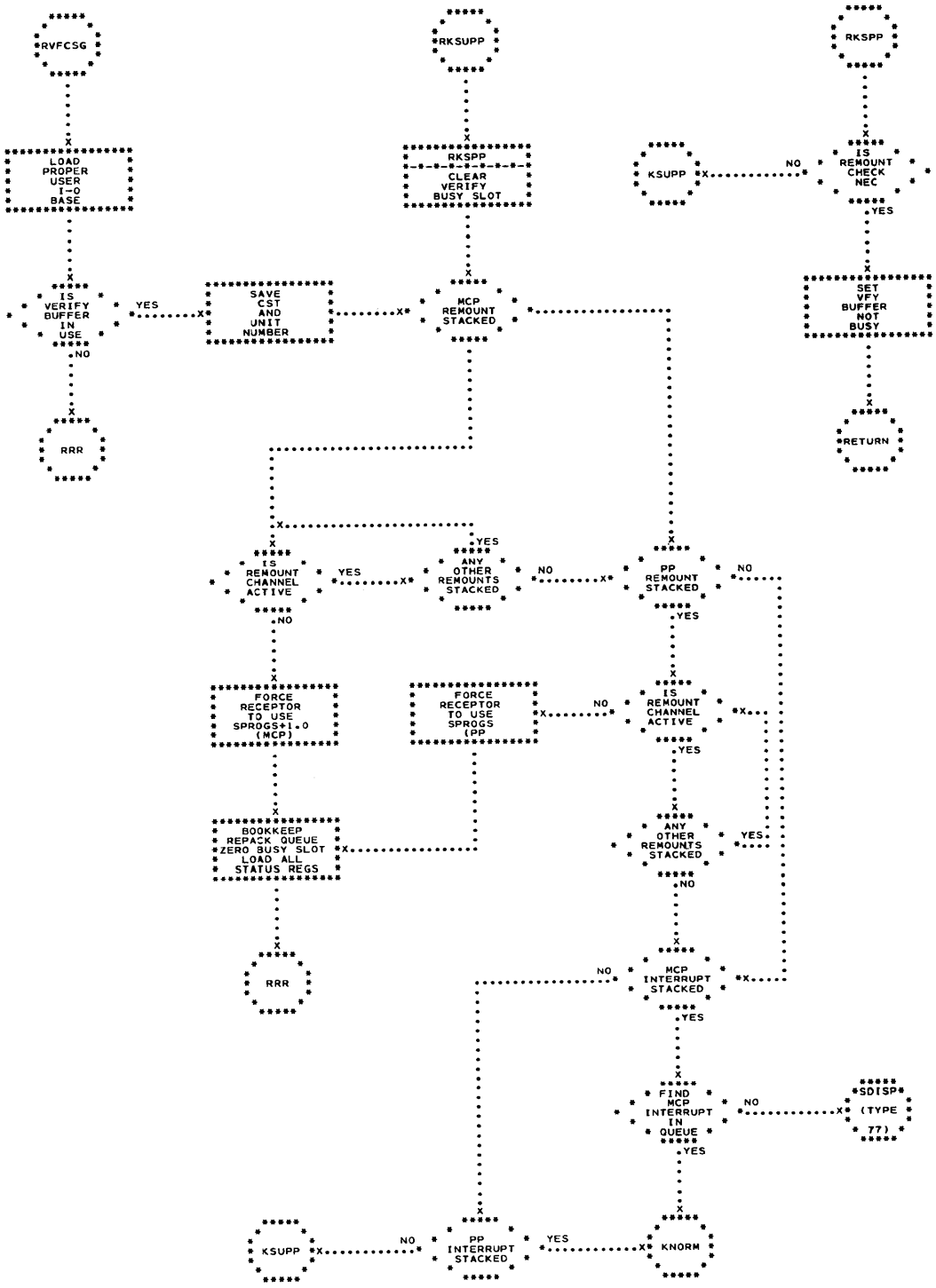


Figure 10. Receptor – Chart 5

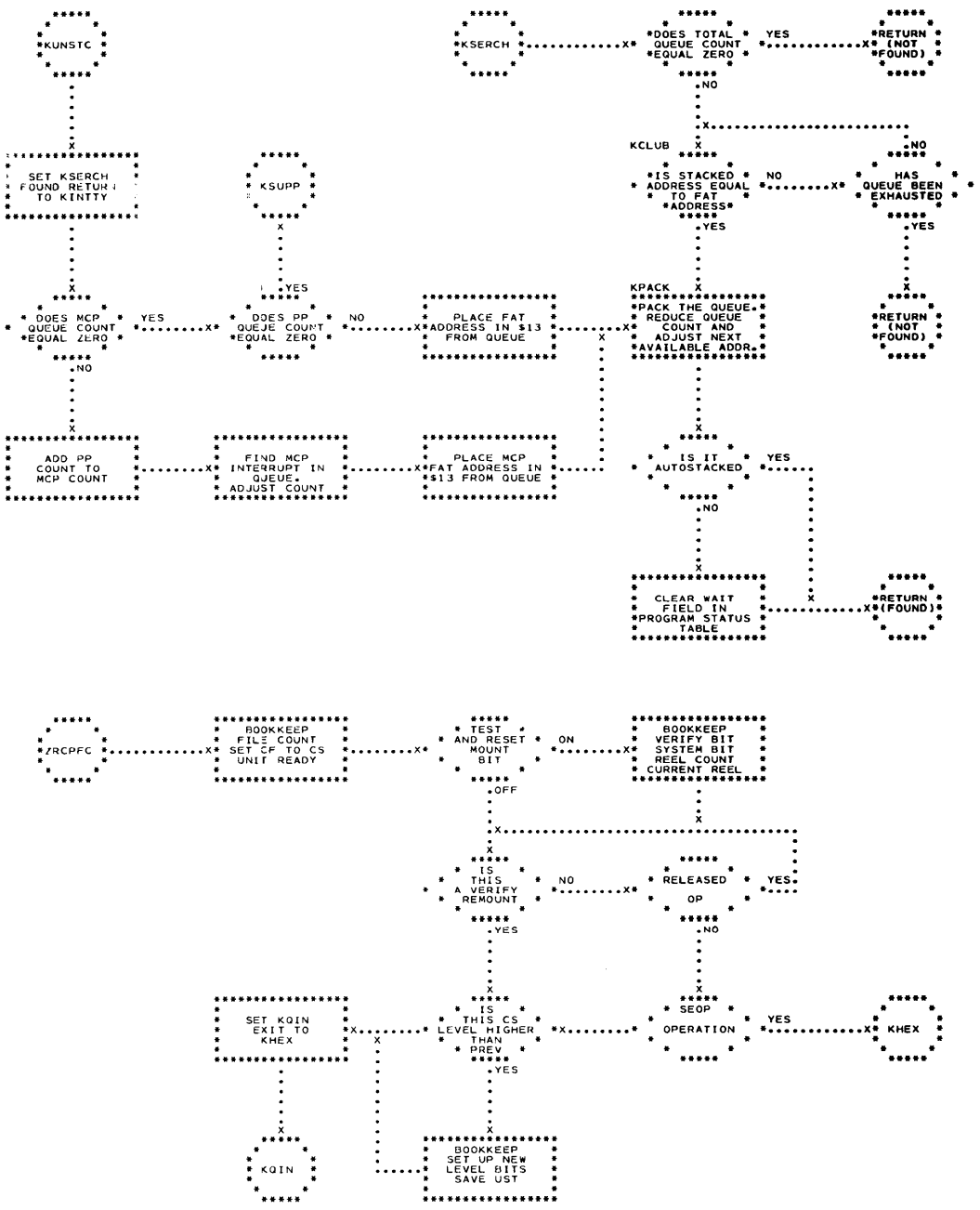


Figure 11. Receptor - Search and Unstack Routines

associated with the unit in select, but the channel signal may have originated at any unit on the channel.

The receptor uses the subroutine KCSIN to stack the CS until a later time. The stacking consists of setting a bit corresponding to the channel number in the word KCSREG. The NOP at KSUPCS in the service op return routine is changed to a branch, and the CS will be unstacked on the next entry to the service op return routine. When this occurs, control passes to KMTCSQ (Figure 8). The channel number for the first 1 in KCSREG is computed, and if the channel is not operating, the receptor is entered at SFAKEI with registers set up as though the CS had just occurred by itself. If, after resetting the bit in KCSREG, the word is all zeros, the instruction at KSUPCS is changed to a NOP.

If the channel is operating the KCSREG is examined for additional 1's. If 1 is found for a non-operating channel, control goes to SFAKEI as described. Otherwise, the service op return routine is used, and unstacking of the CS is deferred until the channel is not busy.

The service op return routine is also used to unstack console interrupts in certain circumstances. If the $COMM pseudo-op had been given while the console was being used for non-commentator I-O, the commentator suppresses the console user's interrupts and changes KSUPP to a branch to KCOMMR to release them. Furthermore, if the commentator suppressed a conceptor EOP, it changed the instruction at KCONCP to a NOP, and stored the copied control word at KCOMCW. If the suppressed interrupt was not the conceptor's, then the commentator stored the UAT address in KCOMMR.

When the service op return routine is entered with KSUPP (Figure 8) set to a branch, the switch is reset and the UAT address (if any) is picked up. If the interrupt is the conceptor's (KCONCP a NOP), the registers are set up as though the interrupt had been received by the receptor and recognized as a conceptor set-up operation. Control is given to PCONE2. If the interrupt is not the conceptor's, then registers are set up as though the interrupt had been received by the receptor, and the receptor is entered at KNORM. The return function of the service op return routine is discussed with the dispatcher programs.

The Conceptor

The conceptor is a special program to receive console channel signals. It performs a set-up console read, and, upon completion, identifies the owner of the CS and sets up a fake CS for that owner. When the subsequent $READ is issued, the actuator enters the conceptor, which performs a data transmission and enters the receptor with an EOP faked for the read.

The three entry points are: PCONE1, when a CS is received; PCONE2, when the read is complete; PCSRD, when the $READ is given.

The Channel Signal Entry

When a console channel signal is received (or unstacked) by the receptor, the conceptor is entered at PCONE1 (Figure 12). The conceptor sets up the addresses of the MCP console tables and gives a set-up console read, with PCONE2 in SRETAD in the UAT. It exits to KSUPP.

The Set-Up Interrupt Entry

When the interrupt occurs from the set up read, the conceptor is entered at PCONE2. The program resets the set up, read, and channel operating bits, and tests the interrupt indicators. An error message is printed if EPGK or UK is on, and control returned via KSUPP. PCONE1 is entered if CS is on, and if EE is on, a message is printed acknowledging the erase.

If EOP is the only indicator on, the message is examined to determine the owner. The owner may be PP, the debugging package, or the command package as follows:

1. The owner is PP if either:
   The keyboard was not enabled (CS key) and the message consists of the switches and keys (the end code is in the third word).
   The first two non-control characters of the message are PP.
2. The owner is the debugging package if the end code is the first character of the fourth word (the ENTER, END sequence).
3. Otherwise, the owner is the command package.

The flow separates according to the owner, and converges again at PMAIN for the two MCP owners, and at PEX1 for all owners. If a typewritten message had been read, it is edited for backspace codes by the subroutine GEDIT, and, according to owner, the proper tables and registers are set up. MCP messages are stored in the buffer PMCPBF, and PP messages are stored in PPPBF. At PEX1, the console signal bit, SCNSSG, is set in the UST, and control given to KQIN with registers set as though a CS had just occurred and with $1 set to KNORM.

The Actuator Entry

The recipient of the faked channel signal is expected to issue a $READ for the console. When the actuator read routine receives a console read request with SCNSSG set in the UST, it enters the conceptor at PCSRD, since the hardware read has already been done.

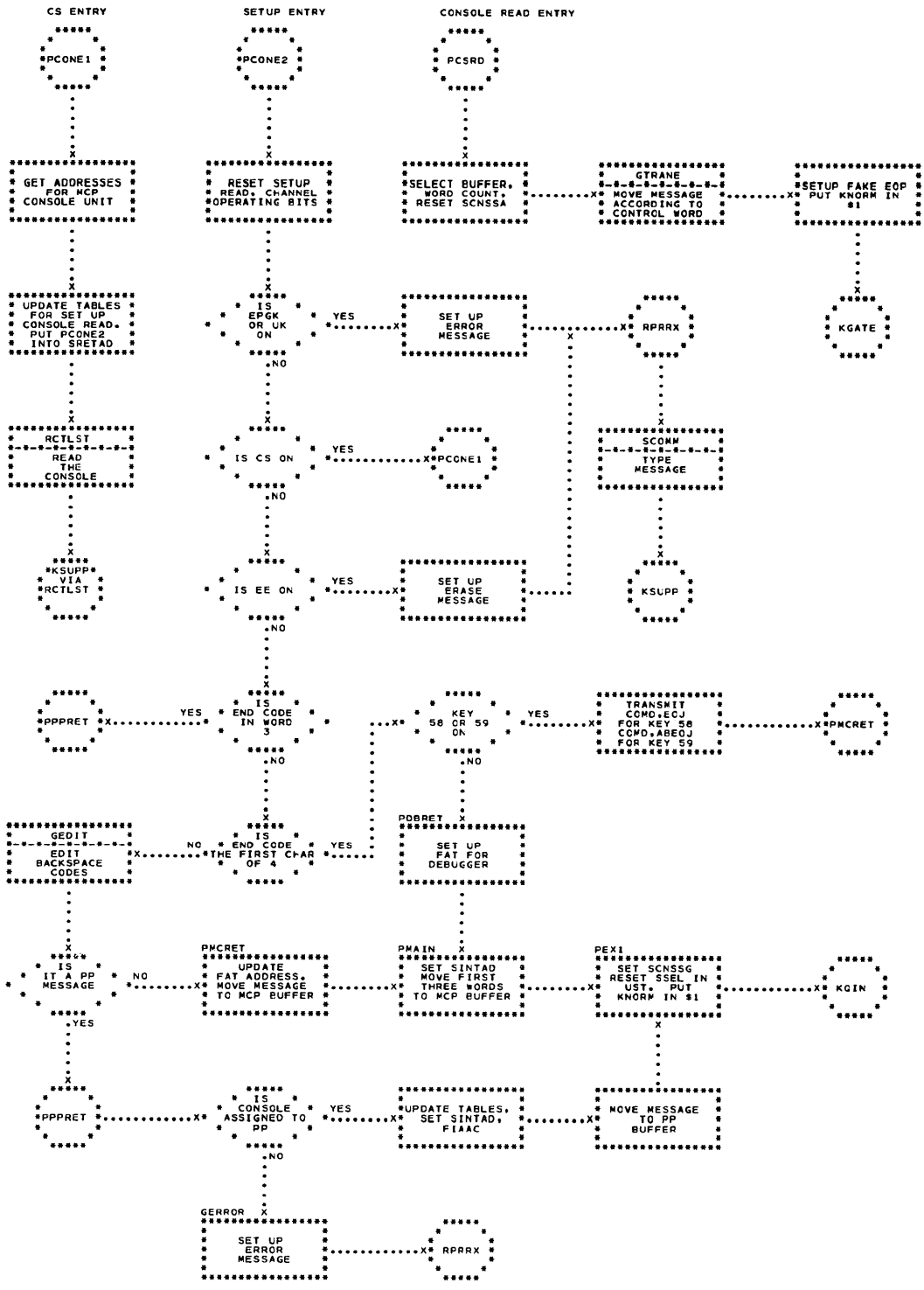


Figure 12. Conceptor

At PCSRD, the SCNSSG bit is reset and the subroutine GTRANF used to transfer the message from the buffer to the user according to the control word. Registers are then set up as though a hardware read had terminated with EOP, and the receptor is entered at KGATE.

THE IF INTERRUPT -- THE DISPATCHER

The IF interrupt initiates major transfers of control within MCP, between MCP and PP, and between the PP I-O fixups and PP mainstream. Control programs concerned with the IF interrupt are referred to collectively as the dispatcher. The dispatcher consists of the IF analyzer, the identifier, the service op return program, the return routine, and error control. All dispatcher programs operate disabled. Figure 13 provides a general picture of flow in and through the dispatcher.

All pseudo-op requests generate an IF interrupt, resulting in control passing to the IF analyzer.

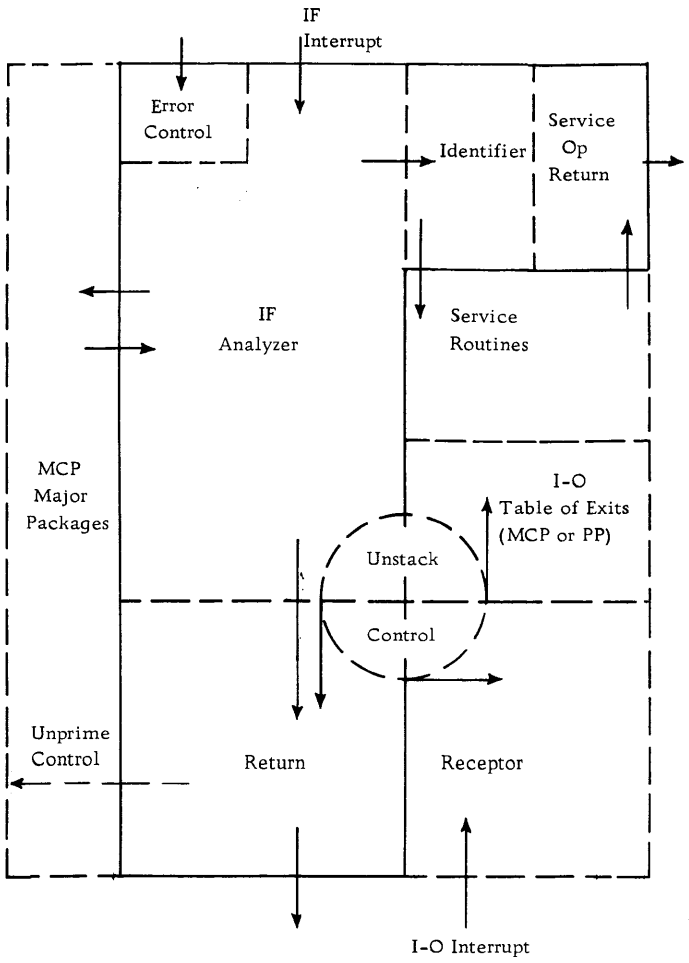


Figure 13. General Flow In and Through MCP Dispatcher (MCP Dispatcher Programs within Solid Lines)

Pseudo-ops are requested by an acceptable linkage of the form

```
B, $MCP
, ($OP)
, parameter
, parameter
```

where $MCP is defined as 32.0 in protected storage, ($OP) is the pseudo-op code, and the parameters depend upon the pseudo-op for their meaning and number. The B, $MCP results in an IF interrupt which is taken by the IF analyzer. The action taken by the IF analyzer depends upon the pseudo-op requested and may involve setting up a tentacle table.

The Pseudo-Ops

Pseudo-ops are considered in two categories: the service pseudo-ops (Figure 14), and the major packages (Figure 15).

The service pseudo-ops are considered an extension of the requesting program; they operate disabled, and always return to the point of request via the service-op return routine (except $RET and $RAM; see description of return routine). The level bit (SL) is not changed when the service-ops are used.

The major packages are MCP programs which may use other MCP programs. They are enabled at some time, and may be primed by MCP programs. When a major package is used, it operates at MCP level (level bit SL = 1). The major packages use

Pseudo-Op	Code	Pseudo-Op	Code	Pseudo-Op	Code
$RD	1.00	$WEF	7.32	$EVEN	15.00
$RDS	1.01	$WEFS	7.33	$EVENS	15.01
$W	1.32	$REW	8.00	$ODD	15.32
$WS	1.33	$REWS	8.01	$ODDS	15.33
$CCW	2.00	$UNLD	8.32	$ECC	16.00
$REL	2.32	$UNLDS	8.33	$ECCS	16.01
$RELS	2.33	$RLF	9.00	$NOECC	16.32
$LOC	3.00	$RLFS	9.01	$NOECCS	16.33
$LOCS	3.01	$RLN	9.32	$ATID	17.00
$FC	3.32	$RLNS	9.33	$SIO	32.00
$FCS	3.33	$KLN	10.00	$RIO	32.32
$TIF	4.00	$KLNS	10.01	$RET	33.00
$TIFS	4.01	$FREE	10.32	$RAM	33.32
$ERG	5.00	$GONG	11.00	$STRG	34.00
$ERGS	5.01	$GONGS	11.01	$FECRG	34.32
$SP	5.32	$WAIT	11.32	$TIME	35.00
$SPS	5.33	$CHEX	12.00	$COMM	35.32
$BSP	6.00	$FREE	12.32*	$SIT	36.00
$BSPS	6.01	$IODEF	13.00	$FIXUP	36.32
$SPFL	6.32	$HD	14.00	$STLR	37.00
$SPFLS	6.33	$HDS	14.01	$FELR	37.32
$BSFL	7.00	$LD	14.32	$ABEX	38.00
$BSFLS	7.01	$LDS	14.33		

* Available only to MCP.

Figure 14. The Service Pesudo-Ops

Available to MCP and PP		Available to MCP Only		
Pseudo-Op	Code	Pseudo-Op	Name	Code
$DUMP	64.00	SJC1	Job Control 1	79.32
$EDUMP	64.32	SOUTPT	Output Command	80.00
$EOJ	65.00	SSPEOJ	Output EOJ	80.32
$HOLD	65.32	SDDT	Debugger	81.00
$RESLD	66.00	SJC4	IPL Entry to JC4	81.32
$FETCH	66.32	SSCR4	First Card Request	82.00
$SPU	67.00	SKOM	Input Commands	82.32
$SPR	67.32	SCOMD	System Commands	83.00
$ABEOJ	68.00	SLOG4	Logger 4	83.32
$SCR	68.32	SLOG2	Logger 2	84.00
		SLOG1	Logger 1	84.32

Figure 15. Major Package Pseudo-Ops

tentacle tables for linkage and return control. When a major package uses another major package, the latter must perform any necessary saving and restoring of registers

The Tentacle Table

Each major package program has an associated tentacle table located in its program area. The tentacle table provides operating control information for the major package program, and is used for linkage, return, and priming control.

The tentacle table (Figure 16) is composed of two and one-half words plus N half-word linkage parameters.

To determine the number of half word parameters (N) in the pseudo-op linkage, $OP is counted as the first half word, then a half word is reserved in the table for each parameter in the linkage. An additional two and one-half full words are in the tentacle table. Therefore, the size of each table is two and one-half words plus N half words.

Each linkage parameter will be processed as controlled by its parameter descriptor. For each parameter in the linkage following the half word for $OP, there will be a corresponding half word slot in the tentacle table.

The IF Analyzer

When an IF interrupt occurs, the IF analyzer receives control in the disabled mode from the interrupt table. The IF analyzer itself may also be entered by a disabled MCP program requesting a major package, or by the return routine entering a primed major package. The IF analyzer channels all requests for the service-ops to the identifier routine. Table 1 summarizes the entries to and exits from the IF analyzer.

The IF analyzer (Figure 17) saves low registers in the STLR buffer. The multiplier register ($MR; see Appendix C) is saved, and the linkage examined for a valid IF interrupt. If the instruction causing the interrupt was not a B,$MCP, control is given to dispatcher error control (SDISP) with error code 1 in $14. If the op code is less than 1.0, or has ones in any of bits 19 through 22, control is given to dispatcher error control with error code 2 in $14. If the op code is in the range of the service ops (less than 64.0), control is given to the identifier routine (KFRONT).

If PP is requesting a major package pseudo-op, the MCP mask and boundaries replace those of PP and the registers saved (STLR) are moved to the backup buffer, SLRBU. The op code is tested to determine whether PP may legally request it (must be less than SJC1). If PP is trying to use an MCP pseudo-op, control is given to dispatcher error control with error code 2 in $14.

The foregoing steps are skipped if the disabled entry was used (SIC,STIC; B,SIFAD). In this case low registers are saved in STLR and a full word is subtracted from the address saved upon entry. Then, for either method of entry, the entry in the IF analyzer operation table corresponding to the pseudo-op is located, and, in turn, gives the address of the tentacle table. If the busy bit in this table is 1, the return address is set as the location of the B,$MCP (or the SIC,STIC, if a disabled entry), low registers are restored, and control is returned to the linkage via the service op return routine.

If an MCP fixup routine tries to use a major package, an error code 80 is put in $14 and exit is made to error control (SDISP). A fixup routine may prime a major routine, but may not attempt to give it control directly, since the interrupt may have occurred while a major package was operating.

The required information is put in the tentacle table according to the parameter descriptors. The level bit (SL) is set to denote MCP ownership of the IC. The entry address is selected (MTC1), the entry modes are preset (MTC2A), and the routine is entered.

The IF analyzer operation table starts at MIFTTT and is used by the IF analyzer to determine the location of the tentacle table pertaining to the pseudo-op requested. The table is constructed so that each half word contains the address of the pertinent tentacle table. It is arranged in order of pseudo-op codes.

Identifier Routine

When a service-operation is requested, the identifier routine is entered disabled from the IF analyzer. The functions of the identifier routine are to:

1. Identify the specific unit assigned to the symbolic file referenced in the calling sequence,
2. Check the calling sequence for a legal request,
3. Transfer control to the proper service routine for the performance of the operation requested.

Table 1. Entries To and Exits From the IF Analzer

ENTRIES

Entry Symbol	Conditions	Registers Preserved	Remarks
SIFAE	Entered disabled from interrupt table due to IF interrupt. Location of interrupt in STIC.	All in STLR. If PP owns IC, also in SLRBU on other than service ops.	Enabled entry, with linkage: B, $MCP; , $OP , PARAMETER , PARAMETER
SIFAD	Entered disabled by user (MCP).	All in STLR.	Disabled entry. User must be MCP. Linkage: SIC, STIC: BD, SIFAD , $OP; , PARAMETER
MTC1	Entered disabled from return. Tentacle table parameters set up by return for primed routine.	None.	Sets up entry address for routine. Used for same purpose by return to enter primed routine.
MTC2A	Entered disabled by return routine returning from major package to major package. Branch address presumed pre-set in MTC3A.	None.	Establishes SIO or non-SIO mode, enabled or disabled branch, and branches to routine.

EXITS

Exit to	Conditions	Registers Restored	Remarks
SDISP	Error; code in $14;	None.	Errors possible: Type 1 (MNOMCP) illegal IF interrupt Type 2 (MOPCI) illegal op code Type 80 (MIFAS) MCP in Auto stack.
KFRONT	$OP is a service op $1.0≤OP≤64.0)	None.	Exit to identifier routine.
Address from tentacle table	Legal non-actuator pseudo-op.	None.	Tentacle linkage set up.
KSUPP	Tentacle table busy bit is set. Return is to the B, $MCP (or SIC, STIC).	None.	Exit to service-op return routine to wait in enabled loop.

The identifier is entered at KFRONT (Figure 18). If the pseudo-op is in the range 32.0 through 38.0 (Table 2), it does not use an IOD reference number, and the appropriate routine is entered. If the pseudo-op exceed 38.0, dispatcher error control is entered with error code 2 in $14.

For pseudo-ops less than 32.0, the IOD reference number (RN) is loaded in $1. For $RD, $RDS, $W, $WS, and $CCW, the control word address is loaded into $0. After turning off $USA and $AD, in the event they came on during one of the LVE instructions, the level bit (SL) is tested. If PP is requesting the pseudo-op, the IOD RN must be less than the maximum PP IOD RN and must be a non-zero integer, or control is given to dispatcher error control (SDISP) with error code 4 in $14.

Table 2. Identifier Routine Entries and Exits

ENTRIES

Entry Symbol	Conditions		Remarks
KFRONT	Entered disabled from the IF analyzer with pseudo-op $2VF points to calling sequence at pseudo-op. $3VF contains pseudo-op. Low registers are in STLR.	64.0	Control sent here by the IF analyzer on any actuator pseudo-op.

EXITS

Exit to	Conditions	Remarks
SDISP	$OP is illegal IOD reference number invalid. Channel or unit not available. Referenced unit has an interrupt stacked.	Type 2 error. Type 4 error. Type 3 error. Type 12 error.
KSUPP	IOD refers to a tape unit in mount or REW status. Channel is operating. Pseudo-op in SFREE for an unassigned unit.	Return address (STIC) set to FWA of linkage. Enabled loop until unit is ready. Return address (STIC) set to the instruction beyound linkage. Treated as NOP.

ENTER Routine

$32.0 \leq \$OP \leq 38.0$

The index registers are unchanged (same as at KFRONT).

Routines are:

Pseudo-Op		Entry Symbol	Pseudo-Op		Entry Symbol
$SIO	32.0	KSIO	$COMM	35.32	JCOMM
$RIO	32.32	KRIO	$SIT	36.0	JSITX
$RET	33.0	CRETN	$FIXUP	36.32	*SCBTT
$RAM	33.32	WRAMPP	$STLR	37.0	TSTLR
$STRG	34.0	KSTRG	$FELR	37.32	TFCHLR
$FECRG	34.32	KFECRG	$ABEX	38.0	YABEX
$TIME	35.0	ZTIME			

* For core storage conservation, $FIXUP has been setup as a major package in the 22 $DUMP leg.

ENTER Routine

$1.0 \leq \$OP \leq 17.0$

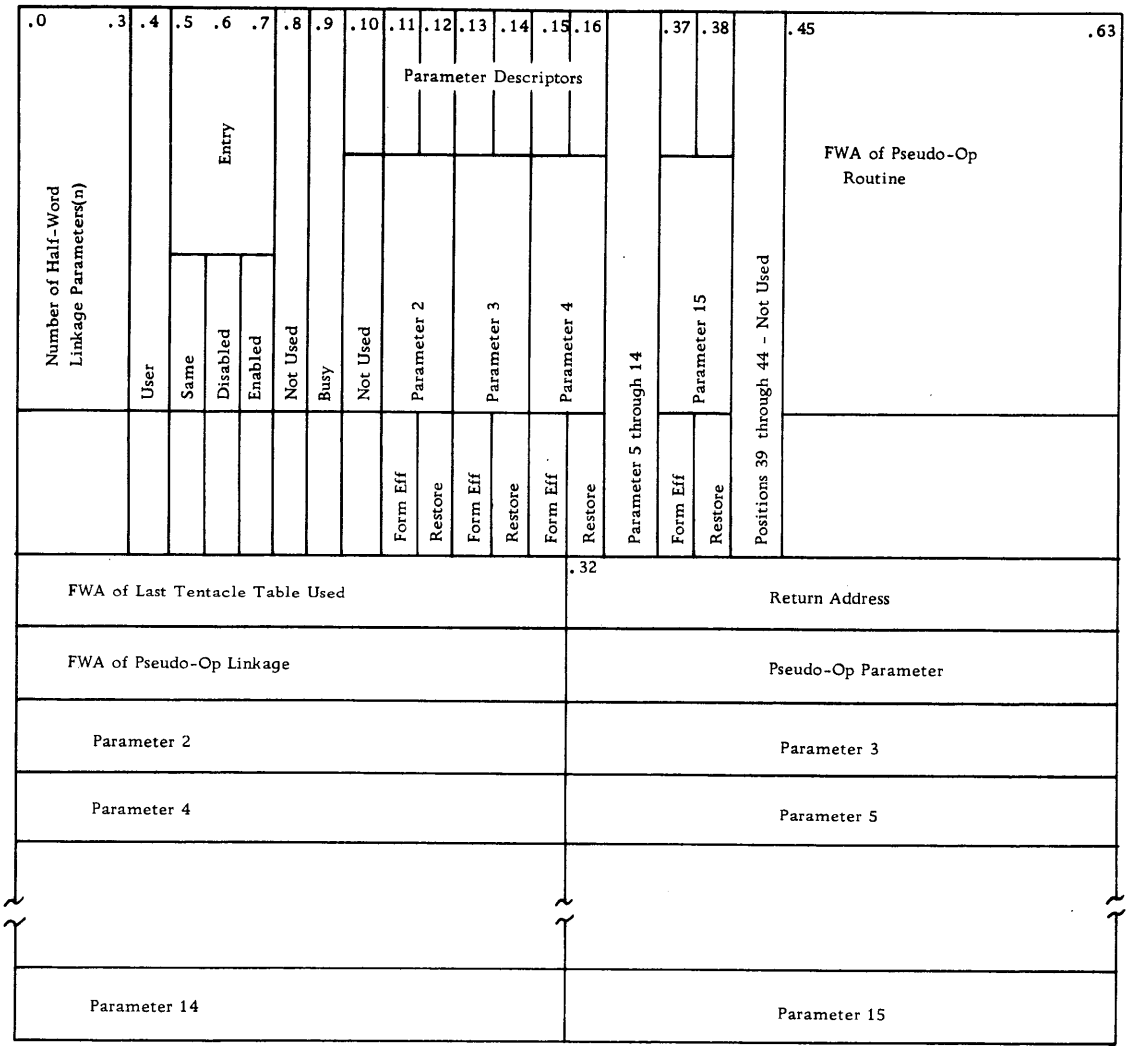
Return address set in STIC.

Actuated address and SEOP control set.

Routines are:

Pseudo-Op		Entry Symbol	Pseudo-Op		Entry Symbol
$RD or $RDS	1.0	ZSTART	$RLF	9.0	BRLFR
$W or $WS	1.32	EWR1	$RLN	9.32	BRLNR
$CCW	2.0	BCCWR	$KLN	10.0	BKLNR
$REL	2.32	WREL	$FREE	10.32	ZFREE
$LOC	3.0	KSLOC	$GONG	11.0	WGONG
$FC	3.32	WFC	$WAIT	11.32	KWAIT
$TIF	4.0	BTIFR	$CHEX	12.0	CCHEX
(Invalid)	4.32	SDISP (type 2)	$FREE	12.32	ZFREE
$ERG	5.0	EELG	$IODEF	13.0	JZIOR
$SP	5.32	R1	$HD	14.0	*RHD
$BSP	6.0	R1 + 1.0	$LD	14.32	*RLD
$SPFL	6.32	R1 + 2.0	$EVEN	15.0	*REVEN
$BSFL	7.0	R2 - .32	$ODD	15.32	*RODD
$WEF	7.32	EWT1	$ECC	16.0	*RECC
$REW	8.0	ZREWST	$NOECC	16.32	*RNOECC
$UNLD	8.32	ZUNLDB	$ATID	17.0	SCBALT

* KCOP gives control to the location as indicated; however, each routine resets $3 to 13.32 and control goes to RALLPS to do functions common to the mode and density operations.



Word 0

bit 0.4 - the user bit defines the identity of the instruction counter owner at the time the IF analyzer was entered. If 1, owner is MCP; if 0, owner is problem program.

bits 0.5 through 0.7 - entry mode bits determine whether the program is to be entered disabled, enabled-SIO'd, or enabled non-SIO'd:

.5	.6	.7	
0	0	0	Non-SIO, enabled entry mode
0	0	1	SIO enabled entry mode
0	1	X	Disabled entry mode
1	X	X	Same as user is entry mode (for PP user, this implies non-SIO, enabled).
(X = 1 or 0)			

bit 0.9 - busy bit - A 1 in bit position 0.9 indicates that the pseudo-op program is in use. The IF analyzer will continuously return to the beginning of the user's linkage via the service op return routine. The user must be in the non-SIO enabled mode, because the busy signal can only be turned off by an I-O routine receiving control upon I-O interruption.

bit 0.11, 0.13, . . . , 0.37 - parameter descriptor (form effective) - If a 1 is in bit position 0.11, the effective address of the first parameter will be formed and saved in the proper tentacle table position. With a 0 in bit position 0.11, the first parameter will be saved. Bit positions 0.11 through 0.38 are reserved for parameter descriptors, with two bit positions for each parameter. Except for the $OP parameter, there are two parameter descriptors (form effective or restore) for each parameter in the pseudo-op linkage, allowing for 14 parameters not including the $OP parameter.

bit 0.12, 0.14, . . . , 0.38 - parameter descriptor (restore) - If a 1 is in bit position 0.12, the first parameter will be returned to the user's linkage when control is transferred back to that package. With a 0 in bit position 0.12, the parameter will not be restored.

bits 0.45 through 0.63 - FWA of pseudo-op routine - Word zero, bit positions 0.45 through 0.63, contains the location for the first word address of the pseudo-op routine pertinent to the tentacle table. The IF analyzer utlimately transfers to this address in the specified entry mode.

Word 1, bits 0 through 23, Value field - FWA of last tentacle Table used - contains the first word address of the last tentacle table and locates the tentacle table pertinent to the user's routine. If the problem program is the user, this half word will be a zero.

bits 32 through 55, Value field - return address - contains the normal return as computed by the IF analyzer unless otherwise modified by the pseudo-operation routine. The return service pseudo-op ($RET) will transfer control to this address when requested by the pseudo-program owning the tentacle table.

Word 2, bits 0 through 23, Value Field - FWA of pseudo-op linkage - contains the address of the first half word in the user's pseudo-operation linkage (B, $MCP).

bits 32 through 55, Value Field - pseudo-op parameter - the identification of the pseudo-op as indicated in the second half of the linkage is saved in this slot.

Figure 16. Tentacle Table Structure

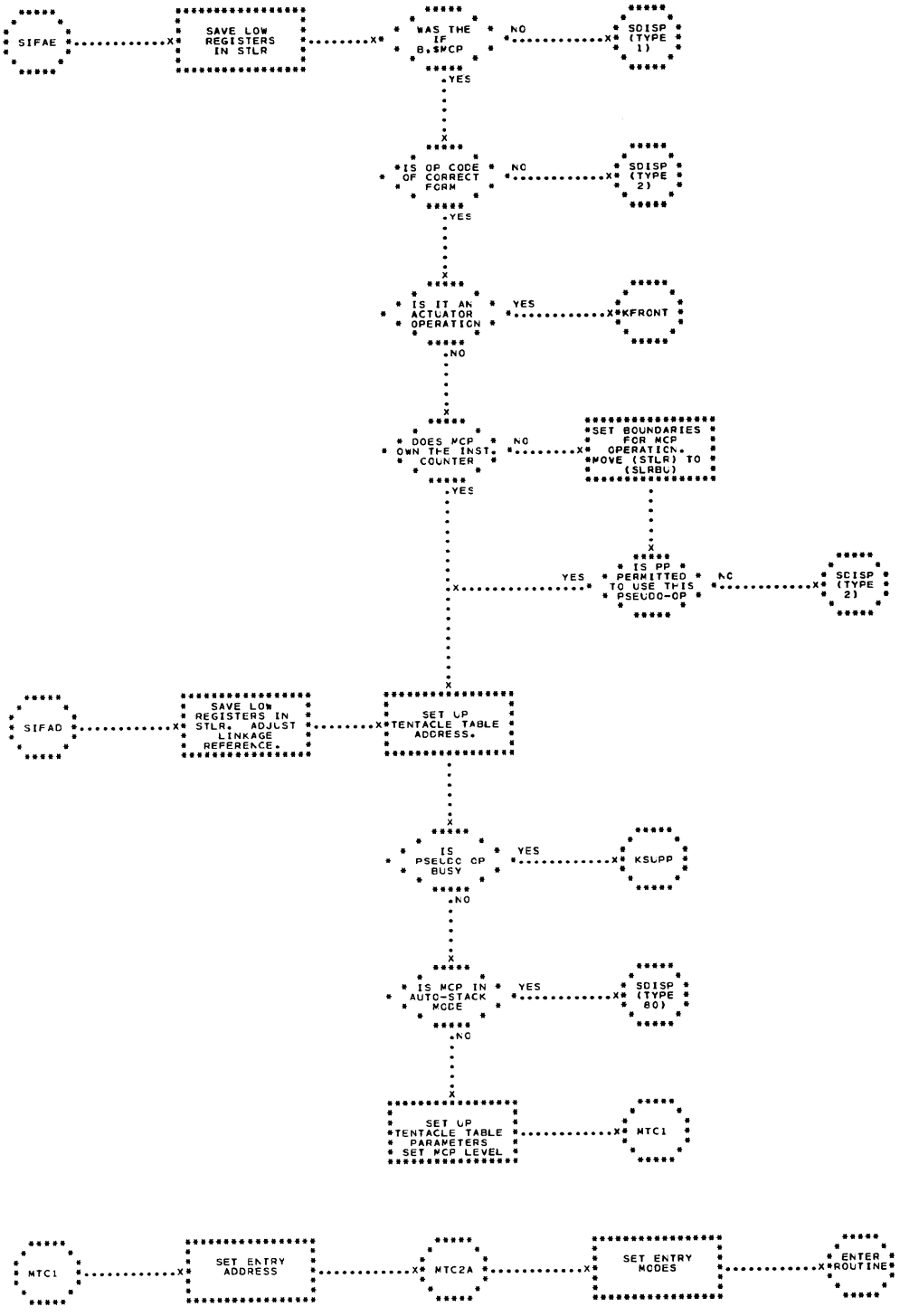


Figure 17. IF Analyzer

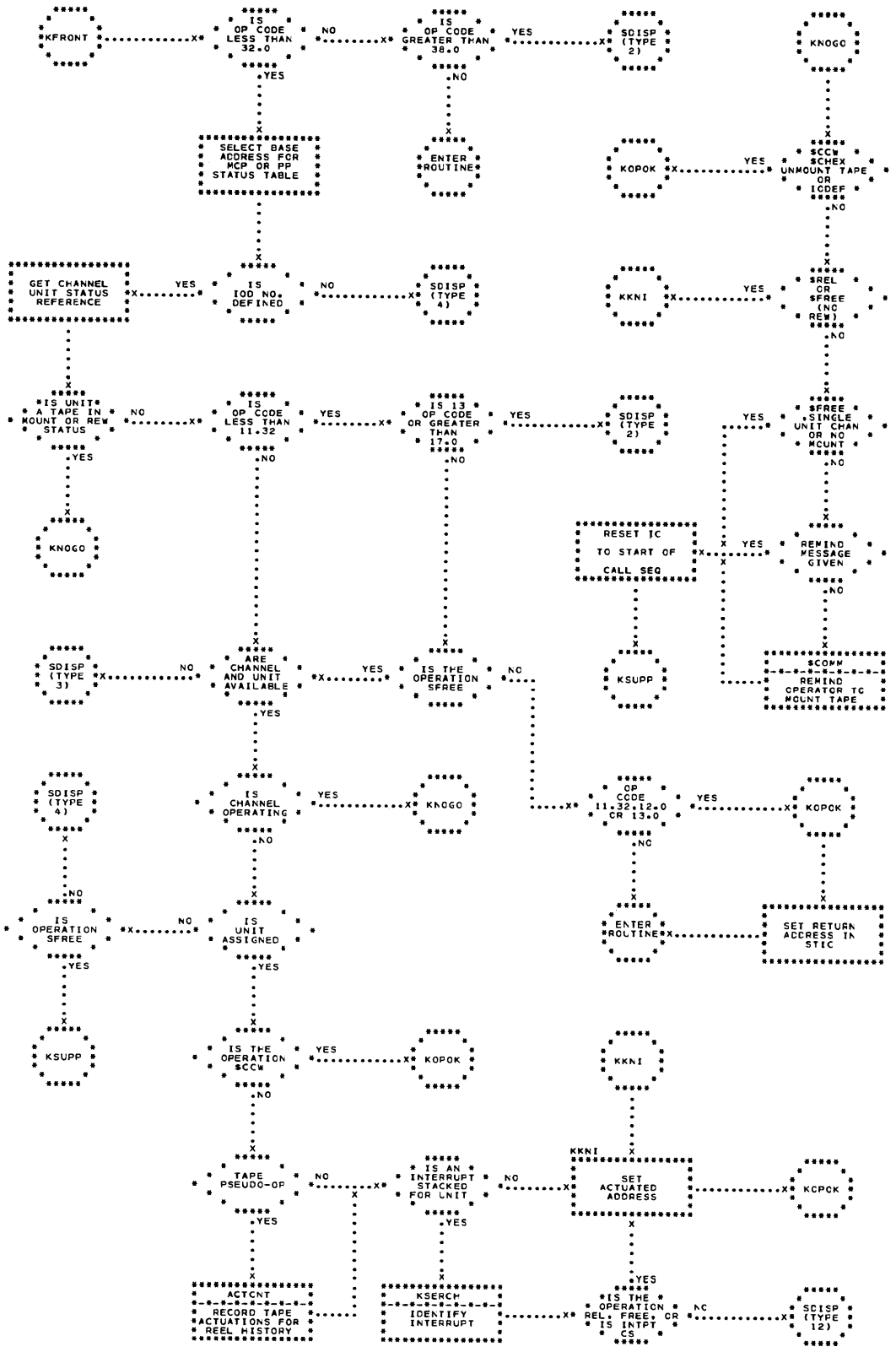


Figure 18. Identifier

The base address of the user's I-O table is added to $1 to locate the entry for the IOD RN. (If MCP is requesting SFREE, it is referring to a PP unit and the PP base address is added.) The unit area address for the IOD RN is loaded into $12 from the I-O table. A zero address denotes the absence of an IOD card for the IOD RN, in which case error control is entered with error code 4 in $14. The file area table is loaded into $13 from the I-O table. Index register 2 is adjusted to point to the first instruction of the linkage (B,$MCP), $14 is loaded with the channel number, and $10 with the reference address for the channel status word in the channel status table. The equipment code is loaded into $4, and $11 is loaded with the correct unit status word reference, depending on whether or not the channel is a multiple unit channel. The index registers except $2, are now ready for entry into any of the routines.

The status of index registers when identifier enters routine (pseudo-op less than 32.0) is as follows:

Index Register	Value Field Contains
$0	Control word address.
$1	IOD RN entry in I-O table.
$2	Return address if service op routine entered.
$3	The pseudo-op code.
$4	Equipment code.
$10	Channel status word reference.
$11	Unit status word reference.
$12	Unit area table address.
$13	File area table address.

If the unit is a tape in mount or rewind status, control is returned (KNOGO) to the linkage via the service op return routine to establish an enabled loop, unless the pseudo-op is one which will not require a hardware I-O instruction. If the pseudo-op is $CCW, $CHEX, or $IODEF, the appropriate routine may be entered, and control is given to KOPOK to accomplish the entry. If the pseudo-op is $REL or $FREE (in the non-rewind status), the actuated address and the SEOP bit must be stored before entering the routine, and control is given to KKNI.

For a single unit channel, or a tape unit not in mount or rewind status (KNI), the pseudo-op code is examined for $WAIT, $CHEX, or $IODEF, which may be entered immediately via KOPOK. A pseudo-op code greater than 17.0 is detected at this point, and control given to dispatcher error control with error code 2 in $14. If the op code is associated with density, mode or alter disposition pseudo-ops, these routines are given control immediately.

If the channel and unit selected are not physically available, as indicated the appropriate bit in the corresponding status word, dispatcher error control is entered with error code 3 in $14. If the channel is operating, the pseudo-op is treated in the same manner as with a tape unit in mount or rewind status (KNOGO).

If the unit has not been assigned, and the pseudo-op is not SFREE, dispatcher error control is entered with error code 4 in $14. Control is returned to the point after the linkage if MCP has requested SFREE for a unit already unassigned.

For pseudo-ops other than $CCW, the appropriate count ($RD, $W, control) is updated if the pseudo-op is for a tape unit, otherwise the unit suppressed bit is examined in the status table. If the requested unit has an interrupt stacked, and the pseudo-op is $REL, $FREE, or SFREE, the interrupt is unstacked and discarded and the pseudo-op entered. A stacked CS interrupt is always discarded. Otherwise, error control is entered with type 12 error selected.

The actuated address is stored in the file area (KKNI), the actuated file address is stored in the unit area, and the SEOP bit originally specified is stored in the SEOP bit for the unit. Index register 2 is adjusted for the return address (KOPOK) and stored in STIC and the routine is entered.

The Service-Op Return Routine

The disabled routines in MCP need to return to the point of the most recent interrupt, or, in the case of an IF interrupt, to the beginning or end of the calling sequence. In all instances, the low register buffer (STLR) contains the information necessary to return. The service-op return routine, KSUPP, is used for this purpose. It includes an unstacking mechanism for console EOP and for tape unit channel signals.

When the routine is entered, STIC must contain the address to which return is desired. If no unstacking is required, that is, the first two instructions are NOP's, the address in STIC is placed in a BE instruction, the multiplier register ($MR; see Appendix C) and the low registers restored from STLR, and the BE executed.

When a disabled routine decides to return via KSUPP, STIC is not always correct for the desired return, and must be adjusted before control goes to KSUPP. Occasionally, this is done in the routine. Other times, the disabled routine branches to a short routine to make the proper adjustment and branch to KSUPP. The most commonly used of these short routines are as follows:

Symbol	Quantity Placed in STIC
KCBUSY	$2VF - 2.0
KELOOP	$2VF - 1.32
KSUPP2	$2VF + .32
KCHOPN	$2VF
KSUPP2 + .32	$2VF

When it is necessary to stack a non-commentator EOP from the console, the first instruction of KSUPP, that is, NOP, KCOMMR, is changed to a branch by the commentator. The next entry to KSUPP results in the release of the EOP to the receptor. (See description of commentator.)

The second instruction of KSUPP, that is, NOP, KMTCSQ, is changed to a branch by the receptor when a CS arrives with a higher priority I-O interrupt. It is unstacked on a subsequent entry to KSUPP. (See description of receptor, channel signal, and console unstack control.)

The Return Routine

The identifier gives control to the return routine in the disabled mode when the pseudo-op $RET is requested. The problem program (PP) must use $RET to end its fixup routines in order to release stacked interrupts, and to permit MCP to restore the lower registers and instruction counter (IC). Only in this way can the PP be taken out of the auto-stack mode. MCP must use $RET to end MCP fixup routines, and to end any major pseudo-op routine.

The ultimate function of the return routine is to return control to the main stream of the problem program. A $RET request in PP main stream results in immediate return to the point of request via the service op return routine. If the $RET is given elsewhere (PP fixup, MCP fixup, MCP routine), the routine must insure that there are no stacked interrupts that can be unstacked, and no interrupted or primed MCP routines which should be given control.

MCP Features and the Return Routine

In order to follow the logic of the return routine, certain basic features of MCP must be understood. These features are described on the following pages.

1. The utilization of buffers to store low registers.
2. The priming of major pseudo-op routines.
3. The use of the program status table.

Low Register Buffers: There are four buffers used to store low registers: STLR, SLRBU, SLRPP, and SLRMCP. Each is thirty words long, and the last word of each is used to hold the IC associated with the low registers (STIC,, SICBU, SICPP, SICMCP).

The low register buffer, STLR, always contains the registers from the most recent interrupt. When the return routine receives control, STLR contains the registers at the B, $MCP which requested the return.

The back up buffer, SLRBU, is used to hold PP low registers as they were when the level changed from PP to MCP. It is filled from STLR by any MCP program which changes the level bit (SL) from zero to one. (Note that this excludes the service pseudo-ops, which operate at the level of the requestor, and which return via the service op return routine.)

The PP low register buffer (SLRPP) is used to hold PP main stream registers when a PP I-O Table of Exits is to be entered. It is filled from STLR when a PP I-O interrupt occurs in non-SIO'd PP main stream. (Note that a PP I-O interrupt occurring at any other time is always stacked, and unstacked when SIO'd PP main stream requests $RIO.) It may be filled by the return routine from SLRBU when MCP has requested $RET and PP interrupts must be unstacked.

The MCP low register buffer (SLRMCP) is used to hold MCP main stream low registers when MCP interrupts MCP. It is filled from STLR by the receptor when the interrupt occurs, or by the return routine when an MCP program requests $RET and an MCP interrupt is unstacked.

Word 9 of each buffer holds $MASK when the buffer is in use. With respect to buffers SLRMCP and SLRBU, bit 0 of word 9 is used to denote that the buffer is in use. The corresponding position of $MASK is permanently 1, so the bit is set whenever the buffer is filled. The return routine resets bit 9.0 of the buffer whenever it restores registers from SLRMCP or SLRBU.

Primed Routines: A pseudo-op is said to be primed when delayed entry has been designated for it. (See description of prime routine.) The need to prime a major pseudo-op arises in two situations:

1. An MCP auto-stacked routine must use the pseudo-op, but major pseudo-ops may not be used in the auto-stack mode.
2. An MCP major package requires that another major package (or itself) be entered at some time subsequent to its $RET.

In these cases, the prime routine is used to enter the request in a revolving queue which the return routine must empty before returning to PP. When a major pseudo-op routine is given control by the return routine because it was primed, its tentacle table has the following unique characteristics which remain when the pseudo-op routine requests $RET, and are used by the return routine to determine subsequent control.

1. The user is MCP.
2. The FWA of the last tentacle table used is zero.
3. The return address is zero.

The Program Status Table: The program status table is a two word table used to record the various modes

of PP and MCP. The first word denotes PP status, the second MCP status. Both words have the same structure:

Symbol	Bits	Quantity
0.0	.0-.17	Address of the file area table corresponding to the IOD reference number for which $WAIT was requested, if the wait is still in effect. If the wait is not in effect, it is 0.
SAS	.25	Auto-stacked bit. 1 denotes the program is auto-stacked, 0 that it is not.
SQK	.32-.49	Interrupt queue count. 0 if no interrupts are stacked for program.
SSIO	.61	SIO bit. If 1, the program is in SIO mode.

The Logic of the Return Routine

IC at MCP Level (SL=1): The return routine (Figures 19,20) is entered at CRETN from the identifier. If the IC is at MCP level (SL is one) $RET was requested by one of the following:

1. An MCP auto-stacked routine (I-O fixup).
2. A major package which was primed.
3. A major package being used by PP.
4. A major package being used by another major package.

Considering each in turn (Figure 19), if MCP is auto-stacked, the auto-stack bit is reset in the program status table (SPROGS). MCP may either be in SIO mode or not in SIO mode. If in SIO mode, then MCP interrupted MCP (MCP is non-SIO at PP level) and return is made restoring low registers from SLRMCP. If not in SIO mode, there may or may not be MCP interrupts stacked. Any stacked MCP interrupts must be taken, taking a waited interrupt first if it has occurred. If MCP is not in SIO mode and has no interrupts stacked, then SLRMCP is examined. If it is in use, MCP interrupted MCP and return is made restoring low registers from SLRMCP. Otherwise, control is given to MUP1 to attempt return to PP. (MUP1 logic is described later in this section.)

If $RET is requested by a major package (not auto-stacked), MCP is put in non-SIO mode. If the requesting routine was primed, (MCP user, FWA of last tentacle table zero), control is given to MUP1 to attempt return to PP (case 2).

If $RET is requested by a major package being used by PP, the tentacle table parameters are restored as specified, the return address is moved from the tentacle table to SICBU, and control is given to MUP1 to attempt return to PP (case 3).

If $RET is requested by a major package being used by another major package, the tentacle table parameters are restored and the return address placed in the low register buffer (STIC). Conditions are set up for return to the routine via the IF analyzer (MTC2A), and if no MCP interrupts are stacked, the IF analyzer is entered. If MCP interrupts are stacked, the unstack routine (KUNSTC) is entered (case 4). NOTE: MCP interrupts are unstacked even if a SIO'd major package is returning to a SIO'd major package.

MUP1 Program: Control is given to MUP1 under the following circumstances:

1. MCP interrupted PP and is returning from auto-stack with no MCP interrupts stacked.
2. A primed routine issued the $RET. (Primed routines originally get control from MUP1).
3. The routine which issued the $RET was being used by PP.

At MUP1, control is given to the unstack routine (KUNSTC) if any MCP interrupts are stacked. If the prime queue is not empty, the next major package in the queue must be unprimed before returning to PP. Its busy bit is tested. If the routine is busy, implying that it released control while waiting for I-O activity to be completed and was subsequently primed, control is returned to the $RET request via the service op return routine. This establishes an enabled loop which will continually renew the request to unprime the routine while allowing the I-O interrupt to occur. If the routine is not busy, the parameters are moved from the prime queue to the tentacle table, the prime queue controls are adjusted, and the tentacle table is set up to indicate that the routine was primed. The IF analyzer is then entered at MTC1 to complete entry to the routine.

If no routines are primed, the address of the current tentacle table (MCR) is cleared (MPP1), and $9 is adjusted to refer to the PP program status table (SPROGS). The subroutine JMPP1 is entered to determine if either $OP, $AD, $USA, or $DS had come on simultaneously with an I-O interrupt when PP was interrupted. If so, dispatcher error control (SDISP) is entered at PP level with the type 14 error code set in $14. Otherwise, the subroutine returns, and PP is checked for auto-stack, non-wait SIO or wait-interrupt not queued modes. If in either, or if it has no interrupts stacked, control may be given to PP restoring registers from the backup buffer

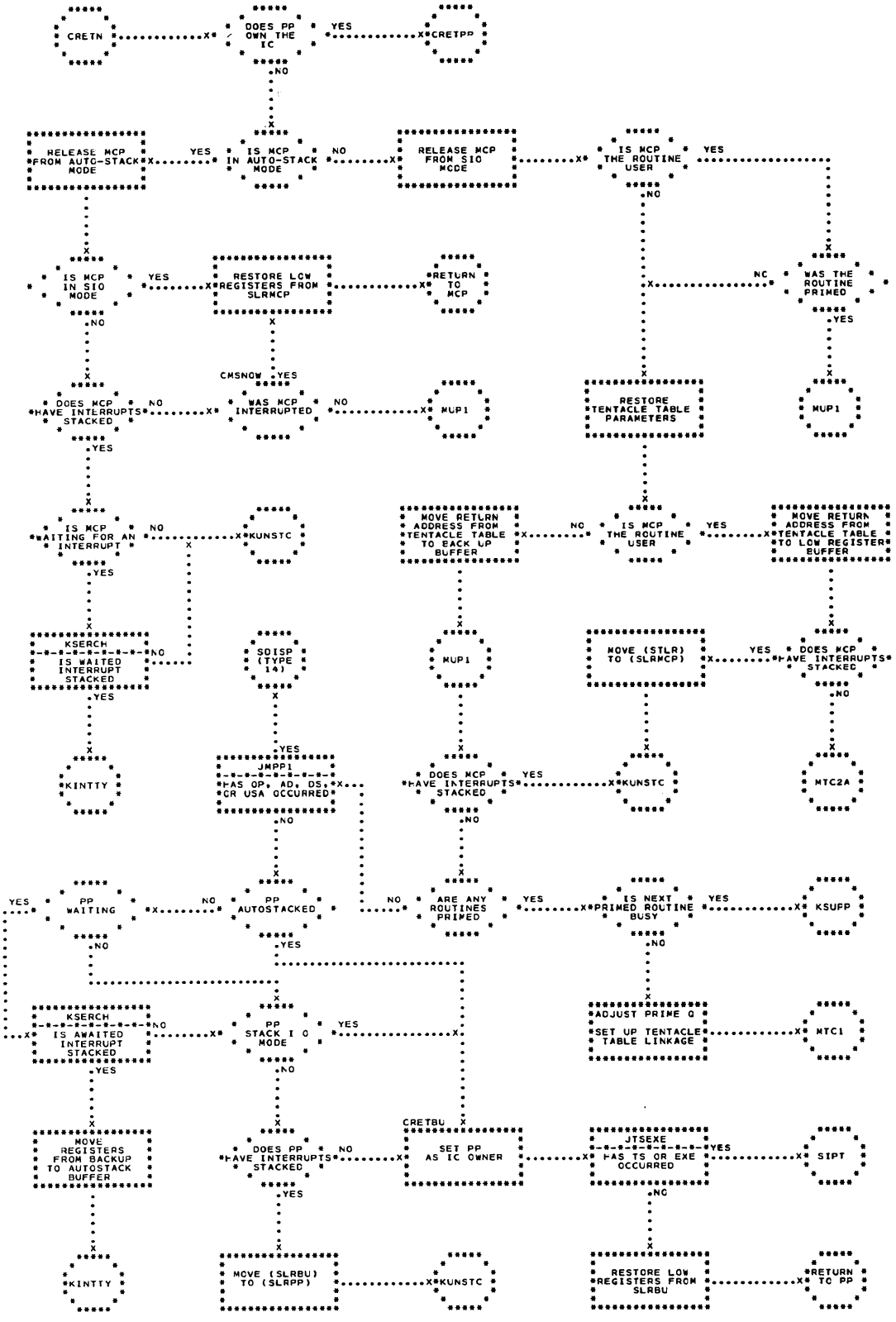


Figure 19. Return Routine - Chart 1

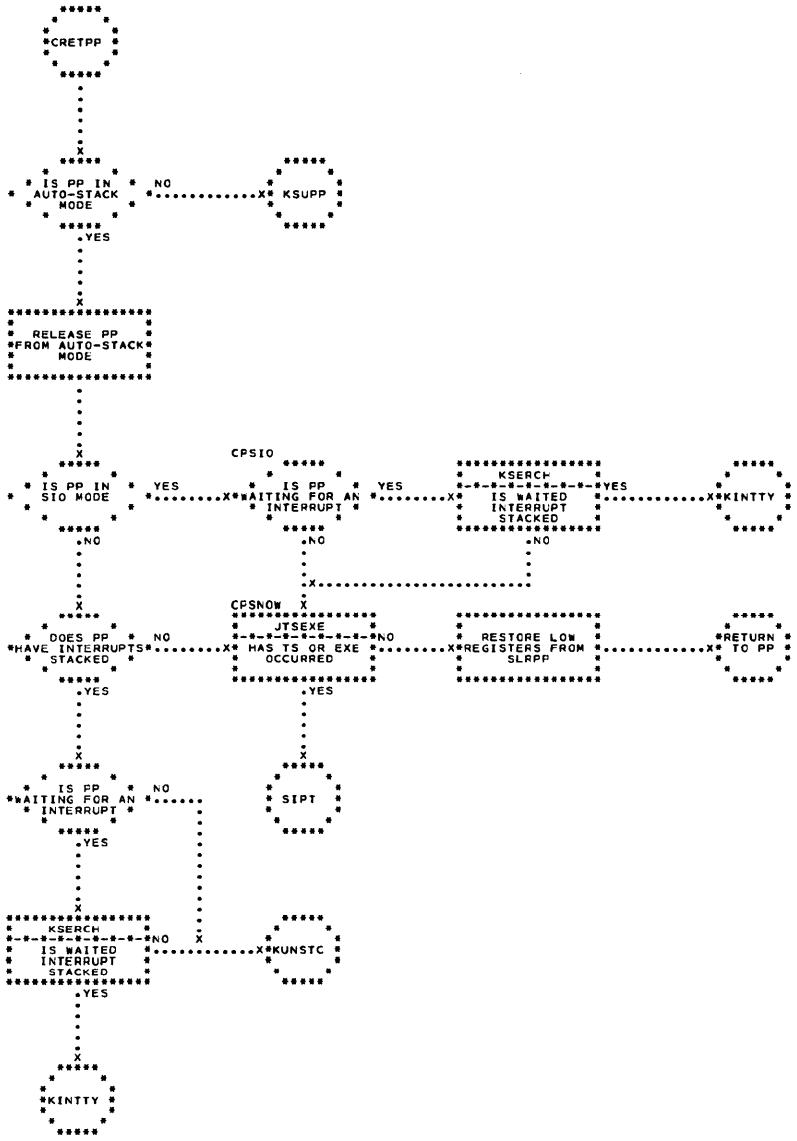


Figure 20. Return Routine - Chart 2

(SLRBU). If PP interrupts are stacked, the contents of SLRBU are moved to SLRPP, and an interrupt is unstacked.

If return is to be from the backup buffer, it must be examined (CRETBU) to determine that TS and EXE are not set. (See the maskable interrupts.) If either is on, the corresponding interrupt is faked by entry to the parallel interrupt table (SIPT) at the appropriate point. If both are off, control is returned to PP with registers restored from the backup buffer.

IC at PP Level (SL=0): If $RET is requested by PP (SL=0), the return routine (CRETPP, Figure 20) examines the PP auto-stack bit in the program status table (SPROGS). If PP requested $RET in main stream, STIC is adjusted to the half word after the $RET pseudo-op and the service op return routine (KSUPP) is used to return immediately. When $RET is requested by PP I-O fixup, PP is taken out of stack mode. Then, if PP is in SIO mode and waiting for an interrupt, the search subroutine (KSERCH) is entered to determine if the waited interrupt has been stacked. If it has, it is unstacked and control given to the receptor at KINTTY. If PP is not in SIO mode and has interrupts stacked, one is unstacked: either the one waited (if PP is waiting and the waited interrupt has occurred) or the next one to be unstacked.

If PP is in SIO mode and not waiting, or not in SIO mode and has no interrupt stacked, the routine (CPSNOW) uses the JTSEXE subroutine to determine if the TS indicator had been set in SLRPP since the I-O interrupt, or if the EXE indicator had been set with the I-O interrupt. In either case, the interrupt is faked by entering the parallel interrupt table (SIPT) at the appropriate point. (See maskable interrupts.) If neither is on in SLRPP, low registers are restored from SLRPP and control returned to the location specified in SICPP.

The exits from the return routine are shown in Table 3.

Error Control

The dispatcher error control is entered when errors generated by program execution are detected by MCP. The routine determines the program responsible for the error. If PP generated the error, ABEOJ is primed, and a message written on the output tape. If MCP generated the error, a message is written on the typewriter and a BD,$ executed.

The error control routine is entered disabled at SDISP (Figure 21) with the A8 error code in $14VF. If the error occurred at MCP level, or if the error is an MCP error (denoted by the magnitude of the error code), the console is released and the error code is converted from A8 to IQS. It is placed in an IQS message which is typed (TYPE XX ERROR), and a BD, $ executed.

If SDISP is entered at PP level with a PP error code in $14VF, the MCP mask and boundaries are selected, SLRBU is filled from STLR, and MCP level is set. ABEOJ is primed, the error code placed in the message skeleton, the ABEX error code is set up, a check is made via RYPRBR for installation optional console typing of the error message, and control given to the short message routine (ZSPLPR).

The error control routine may be entered at SDSPDS if PP accomplishes a branch to $40.0_{(8)}$ or $40.40_{(8)}$. In this case, low registers are saved in STLR, error code 16 put in $14VF, and control given to SDISP.

When the error routine branches to the short message routine, the flow of control is predetermined all the way to the ABEOJ package. The short message routine enters the system print program, a major package. The tentacle table for the system print program will have the user bit set to MCP, and the FWA of the last tentacle table used will be zero, since none was used since PP last had control. When $RET is requested by the system print routine, the return routine will decide that system print had been primed, and examine the prime queue for other primed routines. ABEOJ was primed when the level was changed, and thus will be unprimed immediately, and control never returned to the short message routine.

Error Codes - Problem Program Errors:

Error Type	$14VF	Meaning
1	10001.0	Illegal IF Interrupt
2	10002.0	Pseudo-Op code invalid
3	10003.0	Channel not available Unit not available
4	10004.0	IOD invalid, or Unit not assigned
5	10005.0	I-O TOE address on IOD card invalid
6	10006.0	Request for too many scratch tapes
7	10007.0	Control word invalid
8	10010.0	Control word address invalid
9	10011.0	Communication with protected area
10	412.0	$ATID non-tape IOD
11	401.0	$Chex linkage specifies an illegal address
12	402.0	Unit suppressed
13	403.0	50 maskable interrupts
14	404.0	OP, AD, USA or DS interrupts

Error Type	$14VF	Meaning
15	405.0	Bad label
16	406.0	SuccessfulB,$MCP
17	407.0	Bad PTOE address
18	410.0	Illegal $COMM first word address
19	411.0	Illegal $FIXUP word
20	1012.0	Non-autostacked $STLR
21	1001.0	Repeated UKs ended job
22	1002.0	Incorrect tape mounted twice
23	1003.0	$SCR, $SPR or $SPU calling sequence

Error Codes - MCP Errors:

Error Type	$14VF	Meaning
75	3405.0	MCP setup I-O error
76	3406.0	I-O operation rejected
77	3407.0	File not stacked in interrupt queue
78	3410.0	Interrupt queue too small
79	3411.0	Prime queue too small
80	4012.0	MCP error while auto-stacked
81	4001.0	MCP EPGK
82	4002.0	Special assignment error
83	4003.0	Assign error
84	4004.0	Job control error
85	4005.0	Repeated UKs on disk

The Prime Routine

The prime routine is used to enter the parameters in a revolving queue, and to update the prime queue count. The return routine empties this queue, honoring the prime request.

The prime routine (Figure 22) is entered with the calling sequence:

```
SIC, SPRIMR
BD, SPRIME
, $OP
(Parameter specification as
for entry via the IF analyzer.)
```

After saving index registers 0 through 4, the routine adds the pseudo-op code to the base address of the IF analyzer operation table to fetch the address of the tentacle table. The routine uses the tentacle table to determine the number of parameters to be stored in the queue, the return address, and the form effective control. The tentacle table is not changed.

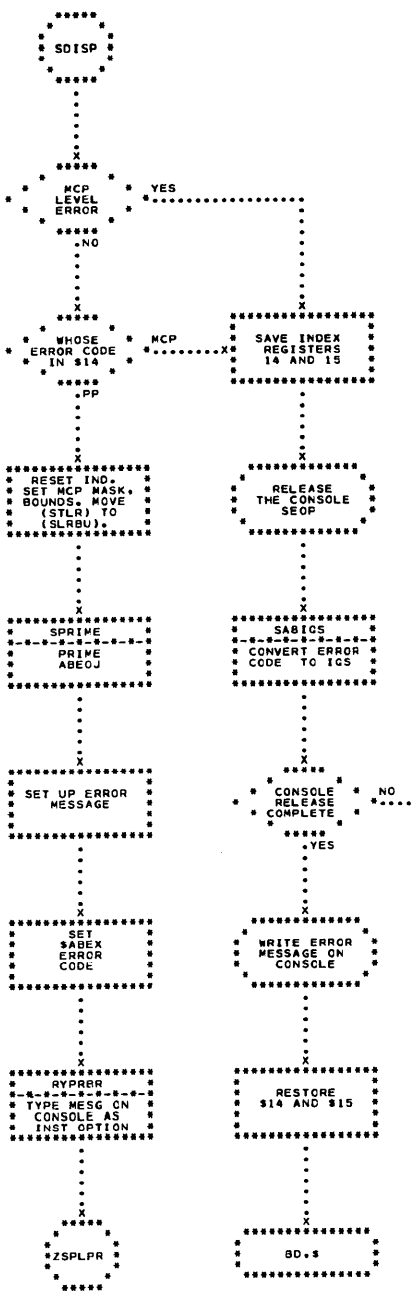
The current queue count (MPRMQK) is examined. If zero, the prime and unprime control words (MR4, MR5) are initialized to insure synchronism.

The parameter count specified by the tentacle table is added to the queue count and to the linkage address, and the resulting return address is stored. If the updated queue count exceeds 30 half words, dispatcher error control is entered at MCP level with error code 79 in $14.

The new queue count is stored (MPRMQK), and the current prime queue control word (MR4) is used to control storage of the parameters in the queue. When the parameters have all been entered in the queue, the updated control word is stored, index registers 0 through 4 are restored, and control is returned (still disabled) to the linkage return address.

Table 3. Exits from the Return Routine

Exit to	Conditions	Registers Restored	Remarks
KUNSTC	An MCP or PP interrupt must be unstacked.	(1) If return to a major package from a major package is being postponed to take an MCP interrupt, (STLR) are moved to (SLRMCP). (2) If return to PP from MCP is being postponed to take a PP interrupt, (SLRBU) are moved to SLRPP).	An I-O table of exits will get control in the auto-stack mode.
KINTTY	A waited interrupt must be taken.	None.	MCP or PP is waiting, and the waited interrupt has been found stacked.
MTC1	There are no MCP interrupts to be unstacked, but a primed routine must be entered.	Tentacle table linkage set up.	The routine is entered via the IF analyzer which sets up entry address and modes.
MTC2A	Returning to the MCP routine which was using the routine that gave $RET.	Return address set in MTC3A.	When an MCP routine uses another MCP routine the latter must perform any necessary saving and restoring of registers.
KSUPP	Return to location is STIC.	All by KSUPP from STLR.	Return to linkage at B, $MCP is made if the next primed routine is busy. Return after the linkage is made if $RET is given in PP main stream. See the service op return routine.
SDISP	An OP, AD, DS, or USA interrupt occurred simultaneously with the interrupt which caused the change to MCP level.	Error code in $14.	No return will be made. Control given to error control.
Return to MCP at location of interrupt	$RET given by an MCP fixup. MCP has no interrupts to be unstacked.	All from SLRMCP.	Normal return when MCP is interrupted.
Return to PP at location of interrupt	No interrupts to be unstacked, no MCP routine to be unprimed, no unfinished MCP routines, no error interrupts.	All from SLRBU or SLRPP.	Ultimate return, Registers restored from SLRBU if MCP gave $RET, and from SLRPP if PP gave $RET.
SIPT (Parallel interrupt table)	TS occurred while in PP fixup or at MCP level or EXE occurred simultaneously with the I-O interrupt.	All from SLRBU.	These are treated as maskable interrupts. They require special handling since they are permanently masked on. Entry to the parallel interrupt table releases the interrupt.



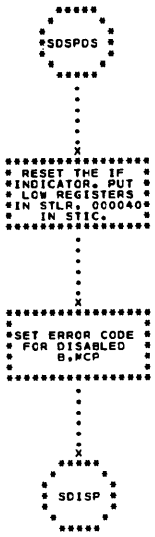


Figure 21. Dispatcher Error Control

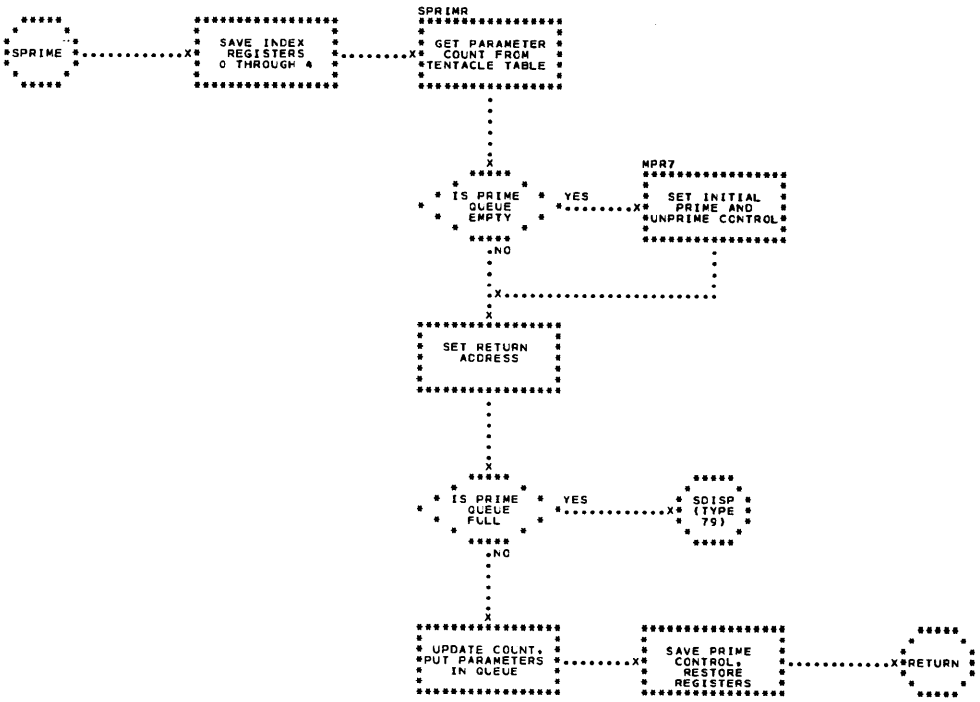


Figure 22. Prime Routine

In order to accomplish its function as system supervisor and automatic operator, a substantial portion of MCP is devoted to programs concerned with system operation. These programs will be considered in three general areas:

1. Initial program load - programs concerned with getting the system started.
2. Job control - programs concerned with guiding jobs through the system, making absolute assignments for symbolic I-O, providing tape mounting instructions for the operator, etc. .
3. System commands - programs concerned with requests to change some facet of system operation.

The initializing program, by its very nature, is a one-shot program which must give consideration to all aspects of system operation. Two other programs are major packages or subroutines of major packages. They communicate in the manner previously discussed, using the tentacle tables, the priming mechanism, and the return routine. (See the Dispatcher.)

SYSTEM INPUT MODES

The first step in the study of system operation programs is to discuss the characteristics of the system input modes and the usage of the system input program. From the point of view of program logic, system input may be classified as either being in overlapped mode or not overlapped (bypass) mode. (Refer to the portion of the MCP Reference Manual concerned with system modes.) Although there is a distinction between on-line and offline overlapped modes, its influence on program logic is not as great.

Overlapped Modes

In the overlapped modes, the input program has two input sources operating at the same time. One source is the read tape, which provides jobs to phase 4 of job control (JC4) to run, and the other is either the scan tape or the card reader, which provides cards to Phase 1 of job control (JC1). JC1 makes preliminary decisions concerning which jobs can be executed, and sets up I-O assignment tables to allow tape mounting instructions to be given before the job comes up for execution.

The system command package must be sensitive to both the source of a command and the mode of the system. Commands encountered by JC1 may require immediate action, or may have to be deferred until phase 4. For example, a COMD, OUTPUT will be performed immediately if the source is JC4 or the console, but deferred if the source is JC1.

Bypass Mode

In the bypass mode, JC1 has no function. I-O assignments are made as the jobs are encountered by phase 4 of job control.

Use of the Input Program

The system input program provides one system pseudo-op, $SCR, which is available to any program. This pseudo-op provides cards from the phase 4 source (the read tape, or the card reader in the bypass mode), provided a job boundary is not encountered. The first card of a job is defined as the first JOB or COMD card not preceded by a B card (excluding T cards).

The input program provides a special pseudo-op, SSCR4, for the benefit of JC4. The SSCR4 pseudo-op is a request for the next card. However, it differs from $SCR in calling sequence and the meaning of the end return. The calling sequence for SSCR4 is:

```
B, $MCP
 , SSCR4
 , FWA(I)
   (end return)
   (normal return)
```

The SSCR4 pseudo-op is a request for one card; thus, no card count is provided, and a partial transmission is not possible. The end return is given by the input program when no cards are immediately available from the phase 4 source. This differs from the end return for $SCR, which is given when no more cards are available for that job. JC4 uses SSCR4 when it is looking for a job card, and only at the beginning of a job.

The input program provides one other pseudo-op with an eight valued parameter. This pseudo-op, SKOM, is most easily discussed by considering the eight sub-types as separate pseudo-ops, as follows:

Pseudo-Op	Calling Sequence	Function
SCR1 (3.0)	B, $MCP , SKOM , SCR1 A VF, 0.0 (End return) (Normal return)	To be used by JC1 to request the address of the next card in the phase 1 buffer. That address will be placed in A. End return will be used if no cards are available.
SCAN (3.32)	B, $MCP , SKOM , SCAN (Normal return)	To be used by JC1 to request the input program to respond to the next SCR1 request with the first card from the next job. (SCAN to the next job boundary in the phase 1 buffer.)
SEJSCN (1.32)	B, $MCP , SKOM , SEJSCN (Normal return)	To be used by JC4 to request the input program to respond to the next SSCR4 request with the first card of the next job. (SCAN to the next job boundary in the phase 4 buffer.)
SONL (0.0) SOFFL (.32) SBYP (2.0)	B, $MCP , SKOM , (Parameter) (Normal return)	To be used by the system command program to inform the input program of a mode change command.
SEOF (1.0) SREW (2.32)	B, $MCP , SKOM , (Parameter) (Normal return)	To be used by the system command program to inform the input program of an input command.

Being major packages, any of the operations previously mentioned may be primed instead of entered directly.

JOB CONTROL

Job control consists of four major packages and their subroutines (decode, unassign, assign, move, etc.) servicing eight pseudo-ops:

Major Package	Pseudo-Op
JC1	SJC1
JC4	$EOJ
	$ABEOJ
	SJC4
Resume Load	$RESLD
The accounting program (logger)	SLOG1
	SLOG2
	SLOG4

Job control will be discussed considering only the overlapped mode of operation. It will be shown later that operation in the bypass mode actually is a special case of overlapped operation. Remember that the basic purpose of overlapped operation is to accomplish tape mounting for one job while a preceding job is running. The following discussion presents the basic concepts of preassignment of tape units as done by job control in overlapped operation. The actual implementation may differ in detail due to various environmental constraints.

Overlapped operation is controlled by phase 1 of job control (JC1), phase 4 of job control (JC4) and the subroutines, decode, unassign, assign, and move. These routines are all tied together by a set of tables called the I-O assignment tables. A detailed description of these tables appears elsewhere in this section. For the moment, they will be defined as having one entry for each job which has passed through phase 1 (JC1 scanning) but not though phase 4 (JC4 EOJ). An entry will be defined as something to identify the job and its I-O requirements. Then, the functions of the six routines may be defined as follows in terms of operations on or for the jobs in the tables.

Job Control 1: Attempts to keep the I-O assignment tables full by taking input (jobs) from the scan source.

Job Control 4: Attempts to empty the I-O assignment tables by taking jobs from the read source and running them.

Decode: On request from JC1, decodes IOD cards into an entry in the tables.

Unassign: On request from JC4 at EOJ, makes the I-O devices used by the PP being terminated available to assignment.

Assign: On request from JC4 at the beginning of the next job in the PP reference table, insures that all its I-O requests have been assigned; in addition, it proceeds down the table assigning leftover tape units until either the tape units are all assigned or the tape requirements of all jobs in the table have been met.

Move: On request from JC4, constructs the I-O control tables needed to run the next job in the tables.

Thus, overlapped operation could be pictured as JC4 pursuing JC1 around the I-O assignment tables, one trying to keep them full, one trying to empty them, with both making demands on the input program for jobs from different sources.

Bypass operation can then be considered as overlapped operation with a set of tables whose capacity is one job.

Job Control, Phase 1 (JC1)

The job control phase 1 major package performs two tasks in MCP. First, it enters symbolic information into the three I-O assignment tables: PP reference, I-O request, and first reel, for pre-assignment of I-O. Second, it makes preliminary decisions about the executability of jobs. A subsidiary function is to dispatch COMD cards to the system command program. JC1 may be considered an interface between the input program and the decode routine, and between the input program and system command package. It operates only in the overlapped mode, and scans jobs on the phase 1 system input tape. (See Table 4.)

JC1 is entered when first used, at YC11ST, and at YC1A on subsequent occasions. It operates enabled, and is always entered via the unpriming mechanism in the return routine, having been primed by JC4 or the input program. JC1 uses three other major packages and two subroutines, as tabulated:

Job Control 1 (Figure 23) normally gets control from the input program, via the unprimed mechanism, when a new job is sensed in the input Phase 1 buffer. JC1 then examines the B cards in that job until either the buffer is empty or the job is fully entered in the I-O assignment tables. Normally, then, JC1 consists of a loop through the calling sequences to the input program and the uncode and decode routines. The remaining code exists essentially to handle special returns from these three routines.

Special Returns for JC1

One such return is end return from the input program, meaning that the input buffer is temporarily exhausted. JC1 gives up control via $RET, with the assurance that it will be given control when the supply of cards is replenished.

Another return is error return from uncode, meaning one of four things has been detected: (case 1) a COMD card, (case 2) a T card, (case 3) the first non-B card, and (case 4) a real error in the B-card in question. Case 1, the appearance of a COMD card, is handled by going to the system command routine. Error return from this routine, in turn, means that the COMD card said REJECT. JC1 effects the reject by setting the TRJECT (see I-O Assignment) bit to one in the PP reference table for the last job. Case 2, a T card, is ignored by JC1. Case 3, the first non-B card, signals that the current job is now complete in the I-O assignment tables. A final entry is made to decode, the input program is instructed to scan to the next job, and control is given up via $RET. Finally, Case 4, a real error in a B-card, causes JC1 to enter decode with a reject disposition set, and from that point it proceeds as in Case 3.

A third exception occurs when the decode routine comes back with error return. If one of the I-O assignment tables is full, SJ1FUL is set to one, and control is given up via $RET (see JC4). Other decode rejects reduce to Cases 3 and 4, as discussed. Whenever JC1 rejects a job, it gives JC4 a reason for it by placing the address of a 4-word diagnosing error message in the second word of the PP reference table for that job. This message will then be printed by JC4 in lieu of running the job.

Miscellaneous JC1 Functions

JC1 has other miscellaneous functions:

1. JOB cards, in addition to furnishing an ID for the PP reference table, cause JC1 to enter the accounting program, via SLOG1 (see description of accounting procedures).
2. Any TYPE, COMPILE card will terminate analysis of the current job, since the following cards are only symbolic input.
3. To accommodate the Fortran IV compiler, IOD cards are ignored by JC1 if the type field begins with $. Such IOD's are treated as T cards.
4. Before each new job is scanned by JC1, a bit (REJJOB. 61) is checked. If this bit is on, an uncorrectible data error has been associated with the previous job (see Input Program). JC1 will reject the previous job just as if a COMD, REJECT has been encountered.
5. The first time JC1 is entered (at YC11ST) it saves the IPL reject count, which is a count of the number of jobs to be skipped by JC1 and JC4 before execution is begun. This count is tested and decre-

Table 4. Use of Major Packages by JC1

Name	Symbol	Linkage	Function
Input Program	SKOM	B, $MCP , SKOM , 3.32	Instruct Input Program to scan to next job.
Input Program	SKOM	B, $MCP , SKOM , 3.0 YC1CA , 0 End Return Normal Return	Request the location of next card in the Input Program Phase 1 buffer. (Location is placed in YC1CA.)
Uncode	YUNCOD	LVI, $14, Y1UCXW B, YUNCOD XW, 0 VF, 1.0 @DISP , 0($10) , YC1DB Error Return Normal Return	Verify card, convert, and break out into YC1DB.
Commands	SCOMD	B, $MCP , SCOMD , 1.0 , YC1DB , 0 End Return Normal Return	To route COMD cards to the command major package.
Logger	SLOG1	B, $MCP , SLOG1 NOP, ($10)	To give a logger card during Phase I.
Decode	LDECOD	LVI, $15, $+1. B, LDECOD VF, YC1DB-1. CF, 0 VF, 0 Error Return Normal Return	To enter JOB names and I-O requests into the I-O assignment tables and make a preliminary check on the IOD cards.

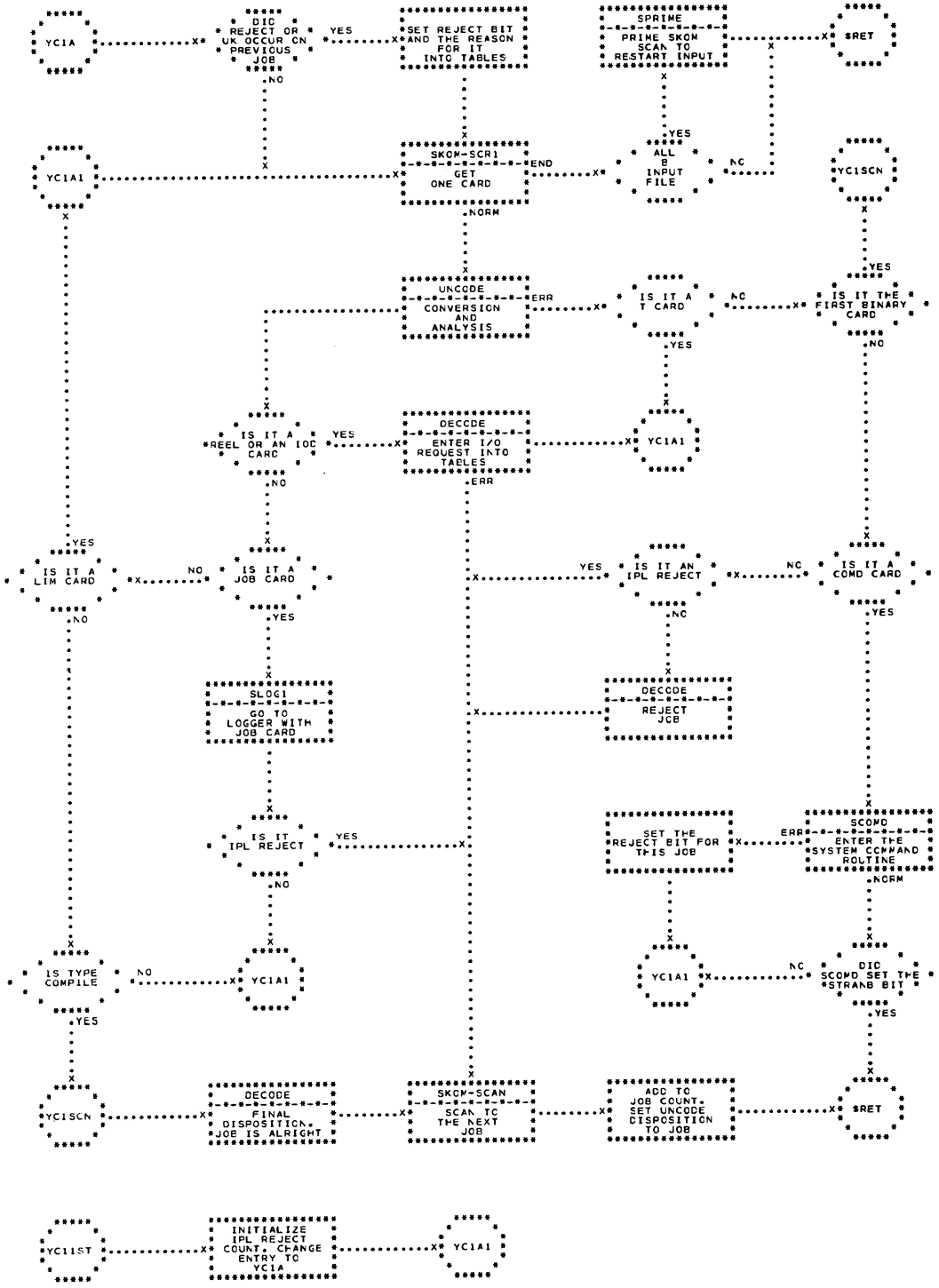


Figure 23. Job Control 1

mented at the beginning of each job, and jobs are skipped until the count reaches zero. They are never entered into the I-O assignment tables, and they will be ignored by JC4.

Job Control, Phase 4 (JC4)

Phase 4 of job control (JC4) has several functions. One of its functions is to be the instrument for the removal of a job from the computer, involving such things as unloading of tapes used by the PP, dumping in response to $EDUMP requests, closing of the job's system output file, and several other items. Other functions of JC4 all involve the initiation of execution of problem programs in response to the B-cards which precede every MCP job. JC4 initializes PP memory and MCP so that the new job is ready to be loaded and executed. It sets up the boundary control; in response to IOD cards, it builds the I-O tables for the PP; it instructs the operator to mount and dismount tape reels according to the IOD cards of the current job or jobs to be run; from the TYPE card it determines the compiling chain (if any) to be used, and keeps track of its links; it recognizes and dispatches COMD cards to the system command routine; and finally, it scans and takes appropriate action on the I-O assignment tables built by JC1. Thus, JC4 is partly an end-of-job, mostly a beginning-of-job, and only for compilers a control-of-job program.

JC4 has one entry point, YC4NO, and three pseudo-ops associated with it:

1. $EOJ -- the current job (or compiler) has finished normally.
2. $ABEOJ -- the current job has finished abnormally.
3. SJC4 -- the current job does not exist; JC4 has been trying unsuccessfully to get a JOB card from the input program.

Both $EOJ and $ABEOJ may be given either by the PP or from an MCP major package via SPRIME. However, SJC4, occurs only when JC4 primes itself. JC4 runs enabled, in the non-SIO mode, and saves and restores no index registers. It runs mostly in PP memory.

The only exit from JC4 is a $RET with, however, one or more of five pseudo-ops primed previously: $RESLD, SJC4, SCOMD, $EOJ, and SJC1. Of these, $RESLD is primed normally at the end of JC4 when a new job or compiler is ready to be loaded. At this point the SLRBU buffer will contain the proper boundary control, the remainder of the buffer containing all zeros with the exception of the MK and IF mask bits. The next pseudo-op, SJC4, is primed when JC4 is unable to read a job card. If a COMD card were read instead, SCOMD would be primed as well. $EOJ is primed whenever JC4 detects and error in the job prohibiting its execution. Finally, SJC1 is primed (along with $RESLD) whenever JC4 sees that the I-O assignment tables have been filled (SJ1FUL = 1), and some space has been made available again.

Major Package Pseudo-Ops Used by JC4

The major package pseudo-ops used by JC4 are shown in Table 5. The service ops and subroutines used by JC4 are shown in Tables 6 and 7. The routines shown in Table 6 are outside Job Control; the routines shown in Table 7 are all basically a part of JC4.

Table 5. JC4 Major Packages

Major Package	Pseudo-Op	Linkage	Function
Input Program	SKOM	B, $MCP , SKOM , 1.32	To "scan" to the 1st card of the next job.
Input Program	SSCR4	B, $MCP , SSCR4 , YBCBU End Return	To ask for the 1st card of a job.
Input Program	$SCR	B, $MCP , $SCR , YBCBU , 1 , 0 End Return	To ask for other B-cards in a job.
Output Program	SSPEOJ	B, $MCP , SSPEOJ	To empty the buffers and write a tape mark on the system output tape.
Output Program	$SPU	B, $MCP , $SPU , YBCBU , 1.	To punch a job card via the output tape.
Dump	$DUMP	B, $MCP , $DUMP , YEDLL	To dump according to $EDUMP formats in case of $ABEOJ.
Disk Fetch	$FETCH	B, $MCP , $FETCH (AX)DD(BU,48,6),...X , 0 , YBCBU , 15. , 0 B, WRESTR NOP	To get LIM and IOD cards for compilers and compiled PP's.
Logger	SLOG2	B, $MCP , SLOG2	To signal end-of-job to a logger.
Logger	SLOG4	B, $MCP , SLOG4 NOP, YBCBU NOP, YJCDBU	To give a logger CC and A8 JOB cards at beginning of job.

Table 6. JC4 Service Ops and Subroutines

Name	Symbol	Linkage	Function
Commentator	$COMM	B, $MCP , $COMM , YC4CM CF, 4.0	To print on-line a beginning of job message.
$TIME	$TIME	B, $MCP , $TIME VF, YC4CM2	To put the time of day into the aforementioned message.
Short Message	ZSPLPR	SIC, ZSPLP9 BD, ZSPLPR , "buffer" , 9.0	To print the JOB, TYPE, IOD, REEL and TITLE cards.
Prime Routine	SPRIME	SIC, SPRIMR BD, SPRIME , ($OP)	To prime one or more other packages, such as SCOMD.
Return	$RET	B, $MCP , $RET	To return.
Major Package Fetcher	YMPFCH	LX, 1, YXWEJM B, YMPFCH LX, 1, RIOABEX-1.0 B, YMPFCH	To get itself into memory from disk. To call the kill PP I/O and attempt $ABEX program into the MCP overlay area.
A6 to IQS Conversion	SA6IQS	LVI, 15, $+1. BD, SA6IQS VF, YJCDB1+1. CF, 8 VF, YC4CM1	To get the JOB name into IQS.
Breakdown Routine	SBRK8	LVI, 15, $+1. BD, SBRK8 VF, YUCBF1+.6 CF, 71. VF, YUCBF2 YPCCT CF, 0 (error return)	To get the compiler names in a chain broken out, and ready to put into KSILO.

Table 7. JC4 Routines

Name	Symbol		Linkage	Function
Decode	LDECOD		LVI, $15, $+1. B, LDECOD	To enter I-O requests when in the bypass mode or for compiler IOD cards.
		YDECVF	VF, 0 CF, 7 VF, 0 B, YICER2	
Assign	TASIGN		LVI, $15, A B, TASIGN	To assign I-O requests, in both bypass and overlapped modes.
		A	XW,	
Move	TMOVE		LVI, $15, B B, TMOVE	To construct I-O tables for the next job to be run. Secondly, to clear out a slot in the PPREF table.
		B	XW, C, D, E, F (error return)	
Uncode	YUNCOD		LVI, $14, A B, YUNCOD	To verify, convert, and break out card designated by disp, and put information in A and F(J).
		A	XW, B, C, D VF, DISP , F(J) (error return)	
Card Code to A6 Conversion	SCA6		LVI, $15, $+1. BD, SCA6 LVE, , SJCTWS CF, 80 VF, YUCBF1	To convert compiler names to BCD.
Unassign	TJUNAS		LVI, 15, A B, TJUNAS	To clear the I-O tables and get ready to assign new job or compiler.
JC4 Print	YPR		LVI, 1, A SIC, YPRRET B, YPR	To print error diagnostics on or off line on a selective basis for each installation.
Read From Source Routine	YRDFSO		SIC, $15 B, YRDFSO , YBCBU	To get an IOD card from $SCR, disk, or core.

JC4 Operations

JC4 will be described in 6 general areas: end-of-job, beginning-of-job, bypass I-O assignment, overlapped I-O assignment, compiler control, and error control. (See Figures 24-31.)

End-of-Job

At YC4NO, if the op is SJC4 there is no PP to be terminated so that the JC4 program is called into PP core and given control, otherwise a section of code is called into the overlay area of MCP. This code assures that all PP I-O is done. If the PP is writing, a release is issued when the write exceeds an installation determined time constant. If the PP is reading, a release is issued immediately. If the PP is doing a control function, MCP simply waits for completion. For the $EOJ case, the JC4 program is called into PP area and given control. For $ABEOJ, a $DUMP is taken using the proper limits. MCP then checks for $ABEX usage. If there is no successful use of $ABEX, the JC4 program is called into PP area and given control. For successful use of $ABEX, MCP puts the PP in non-suppressed mainstream mode and discards all I-O interrupts before returning to the PP.

After calling itself in from disk, JC4 checks the op code for SJC4 to skip the end-of-job procedure if it has already been effected, i.e., JC4 is looping waiting for a new job card.

The next step is to reset SSYRFT, the temporary arc assignment pointer, to SDKMCP, the permanent reset arc (see description of move routine later in this section). Now the compiler routine, YC4INT is entered if a compiler was running, otherwise, the input program is instructed to "scan" to the next JOB or COMD card. The PP's I-O is then "unassigned, and the output program instructed to empty its buffers (SSPEOJ), provided there is something in them (YSSPBT = zero). In addition, several items are reset, among them the commentator buffer (PONOUT) and the maskable interrupt counter (YMISCO). At this point termination of the current job is complete. and JC4 is ready to begin a new job.

Beginning-of-Job

At YC4B2 (Figure 24), the LFINB bit is checked to see if decode is currently processing B cards for JC1. If so, steps are taken to allow either JC1 to finish processing the phase 1 job or to force a reject of the phase 1 job to prevent a system hang. In any case control goes to RLFINF where the SSCR4 pseudo-op is then used to ask for a job card. If the input program gives end return, JC4 again waits via a prime SJC4-$RET sequence, but first it checks SJ1FUL to make sure phase 1 is not stopped for lack of table space. If the input program gives normal return, the card is sent to uncode for verification. If it is a JOB card, one more hurdle must be passed: the WREJJB routine which checks for phase 1 rejects (see JC4 error control). Then the job card is converted to A8, given to the logger, converted to IQS, and printed and punched on the output tape. A message is printed, using the commentator, with the job name, the time, and the day. With this message, the job is fully annotated, and JC4 turns to the succeeding B-cards. Two separate logic flows exist, one (for the bypass mode) beginning at YTCSR, the other (for the overlapped modes) at WJ4PA. The flow converges again at WJC4G.

Bypass I-O Assignment

In bypass I-O assignment (Figure 27), the TYPE card is read and uncoded using the Uncode routine. If it contained COMPILE or COMPILGO, the compiler set-up routine is entered at YCOJB1. If not, the LIM card is read at YC4B5. The read source routine is used because the flow is rejoined at this point for compiler LIM and IOD cards. The limits are saved in all the necessary places: YEDLL, the $EDUMP buffer; SLRBU, the eventual boundary control register; and SMARK, to establish the upper extent of the PP's I-O tables. Before the upper limit is used to establish the PP I-O location table base address (SBAPP), JC4 makes sure it is greater than YMAX, to prevent any possible memory conflict. The next step by JC4 is to string out all the IOD cards in memory in the format required by the decode and move subroutines. This is done in a loop beginning at YIODSO (Figure 27) which is broken normally when the first binary card is encountered. The code from YC4C9 to WJC4G effects successively the decoding of the IOD and REEL cards, the actual I-O assignment, and the construction of tables by Move. This whole sequence is skipped for jobs with no IOD cards. At the end SCORG is checked so that special action may be taken for compiler at YC4INT.

Overlapped I-O Assignment

This logic starts at WJ4PA (Figure 30) by checking the TYPE card, as in the bypass mode. It proceeds to YCOJB if compiler setup is necessary. At WJ4PF the read source is set to $SCR, because WJ4PFF is the entry point for go phases of overlapped COMPILGO jobs. Now the logic differs from that of the bypass mode. Since the IOD cards were decoded by JC1, the PP is ready to be assigned. If all goes well, the LIM card will be read (at WJ4PG) and uncoded. If assignment comes back with a reject, a diagnosing message will be printed. After uncoding the LIM

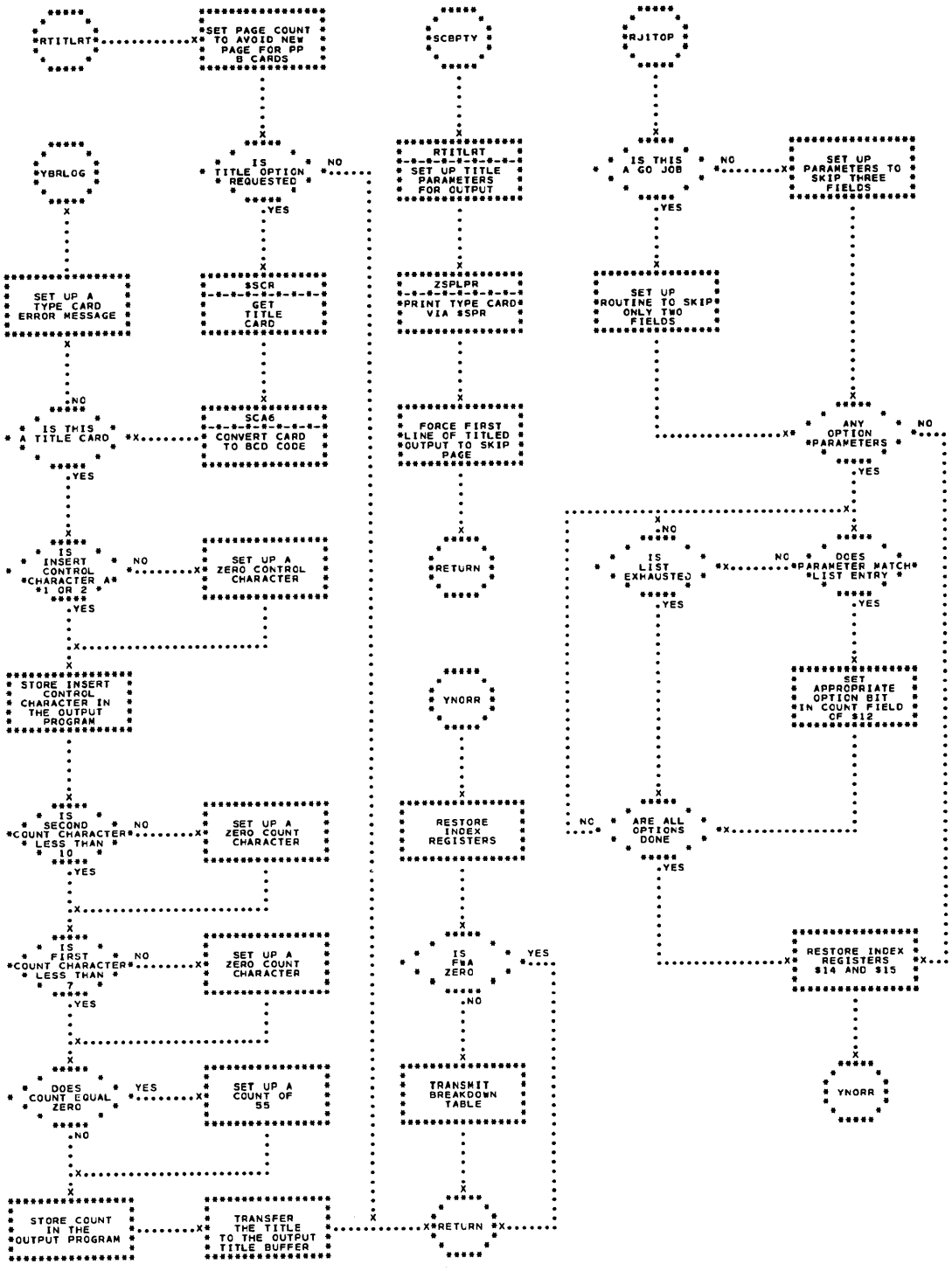


Figure 24. JC4 - Chart 1 - Entry

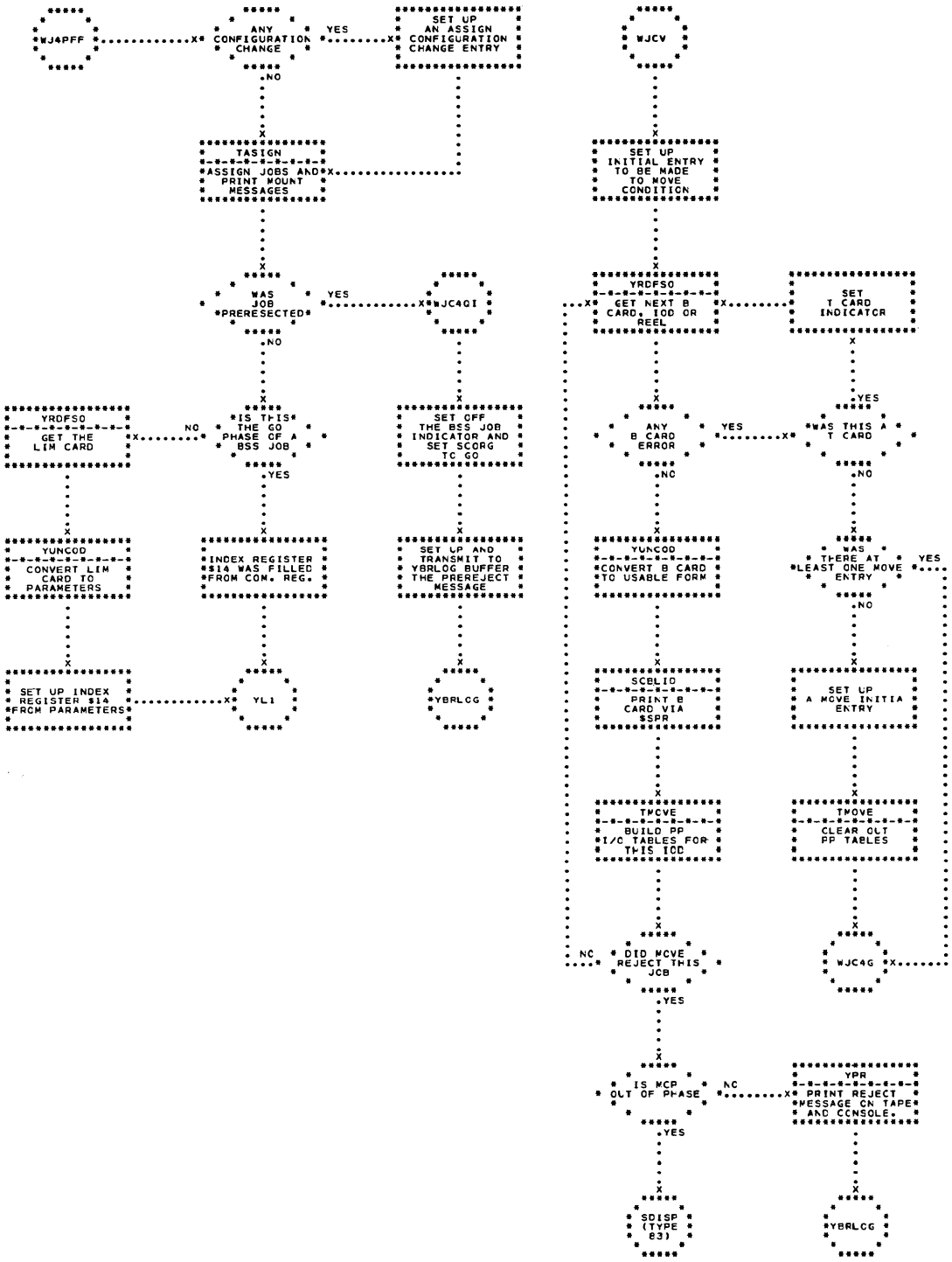


Figure 25. JC4 - Chart 2 - Job and Type Card Handling

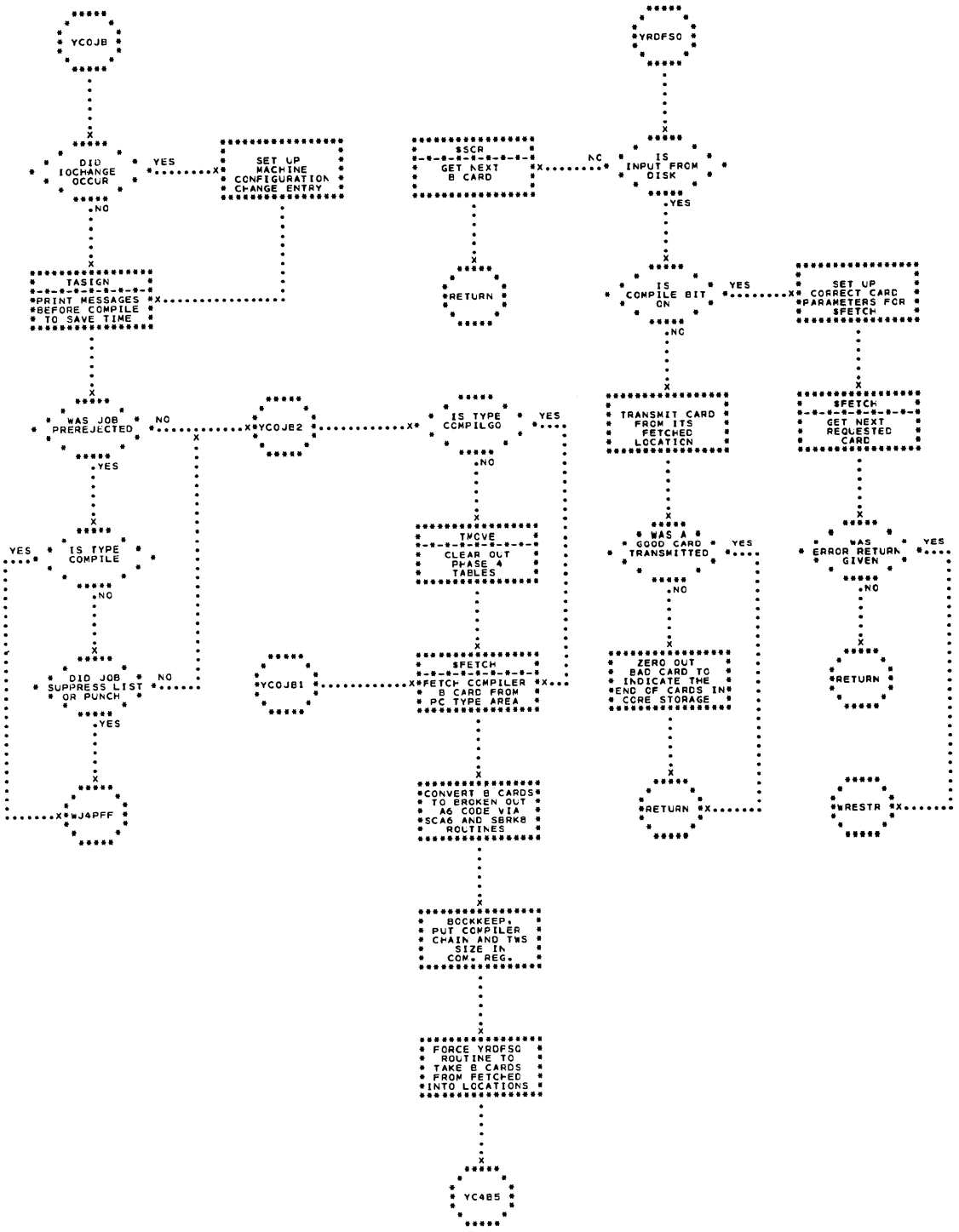


Figure 26. JC4 - Chart 3 - Compiler Control

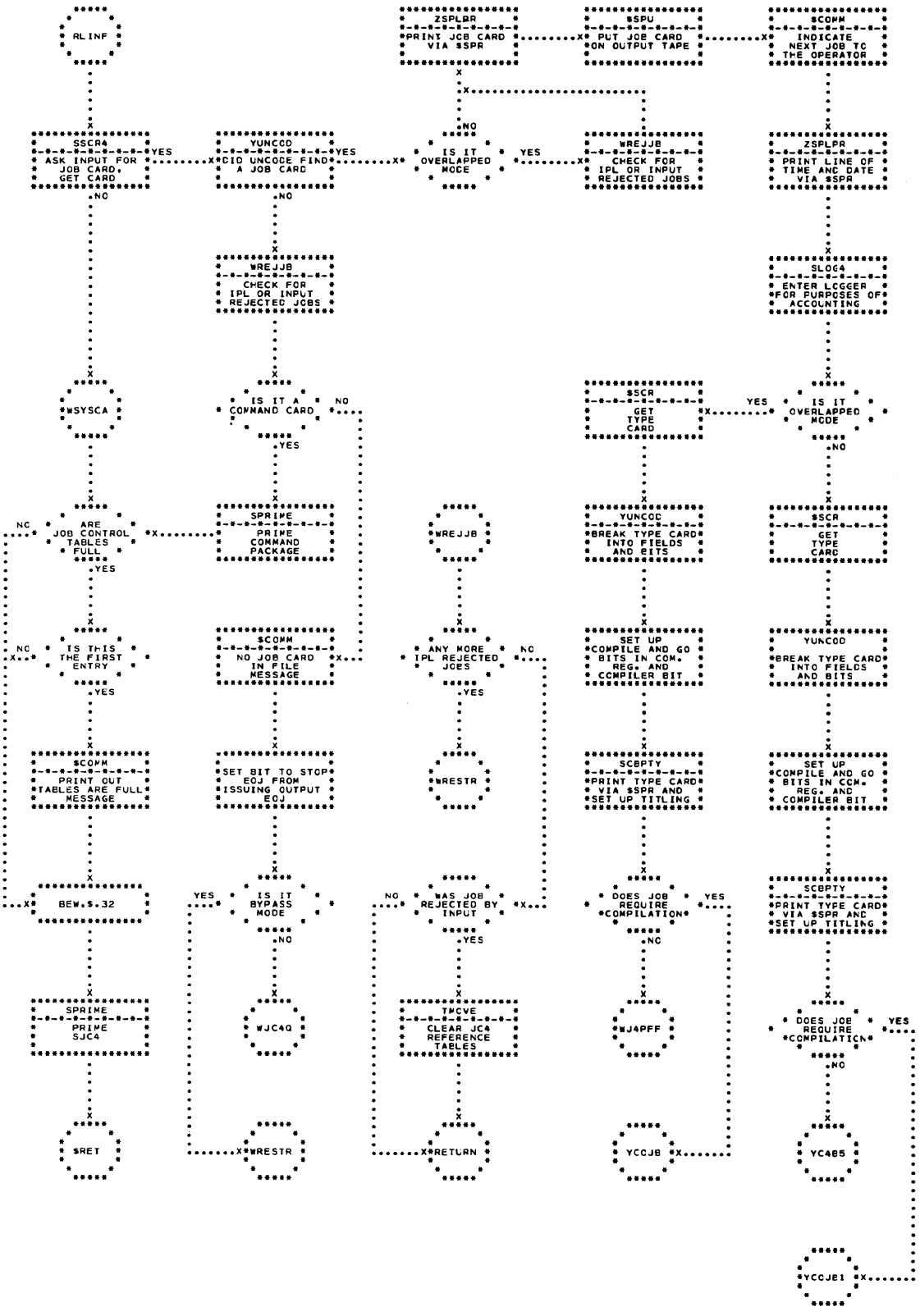


Figure 27. JC4 - Chart 4 - Bypass and Compiler I-O1

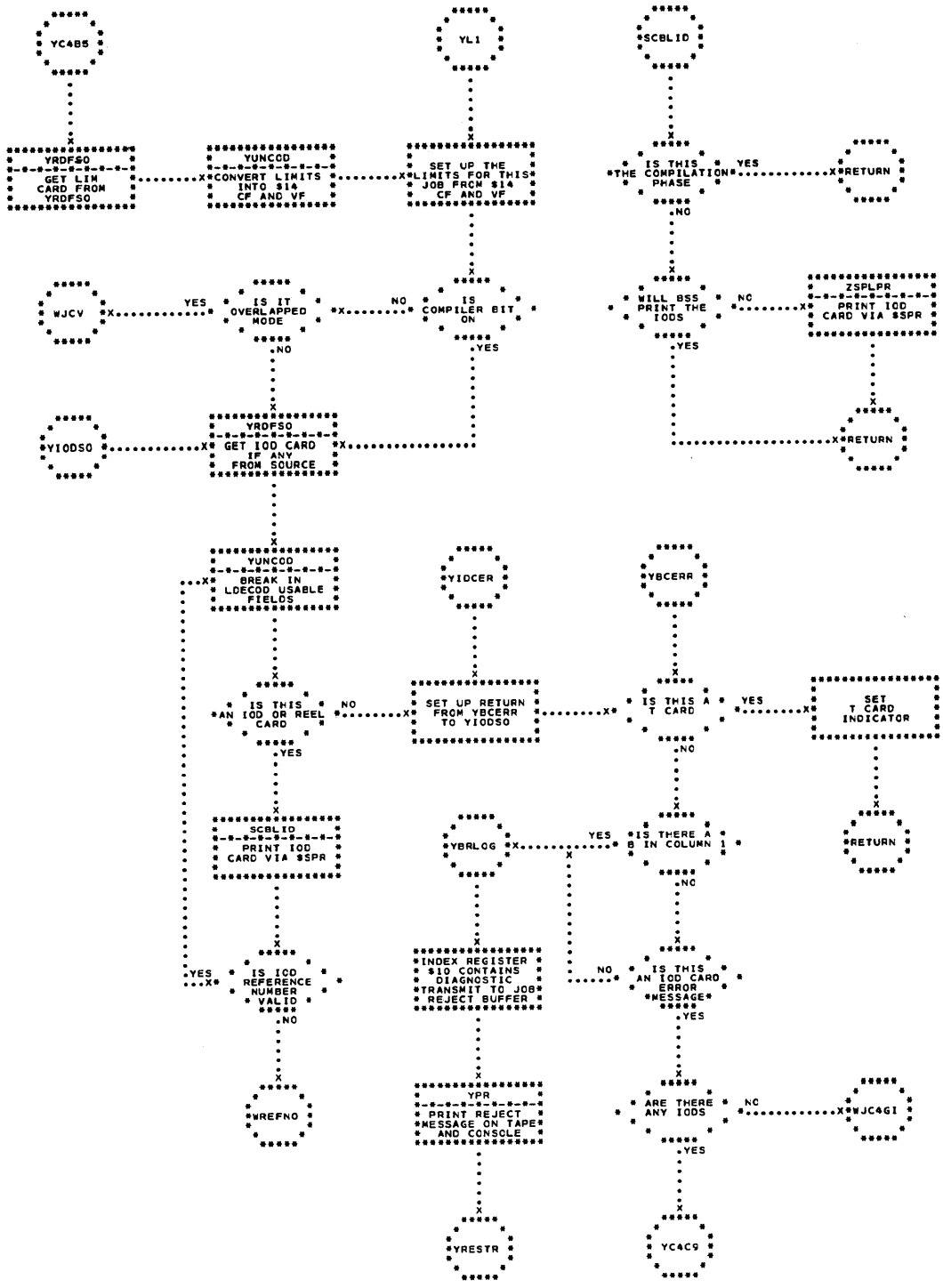


Figure 28. JC4 - Chart 5 - Bypass and Compiler I-O2

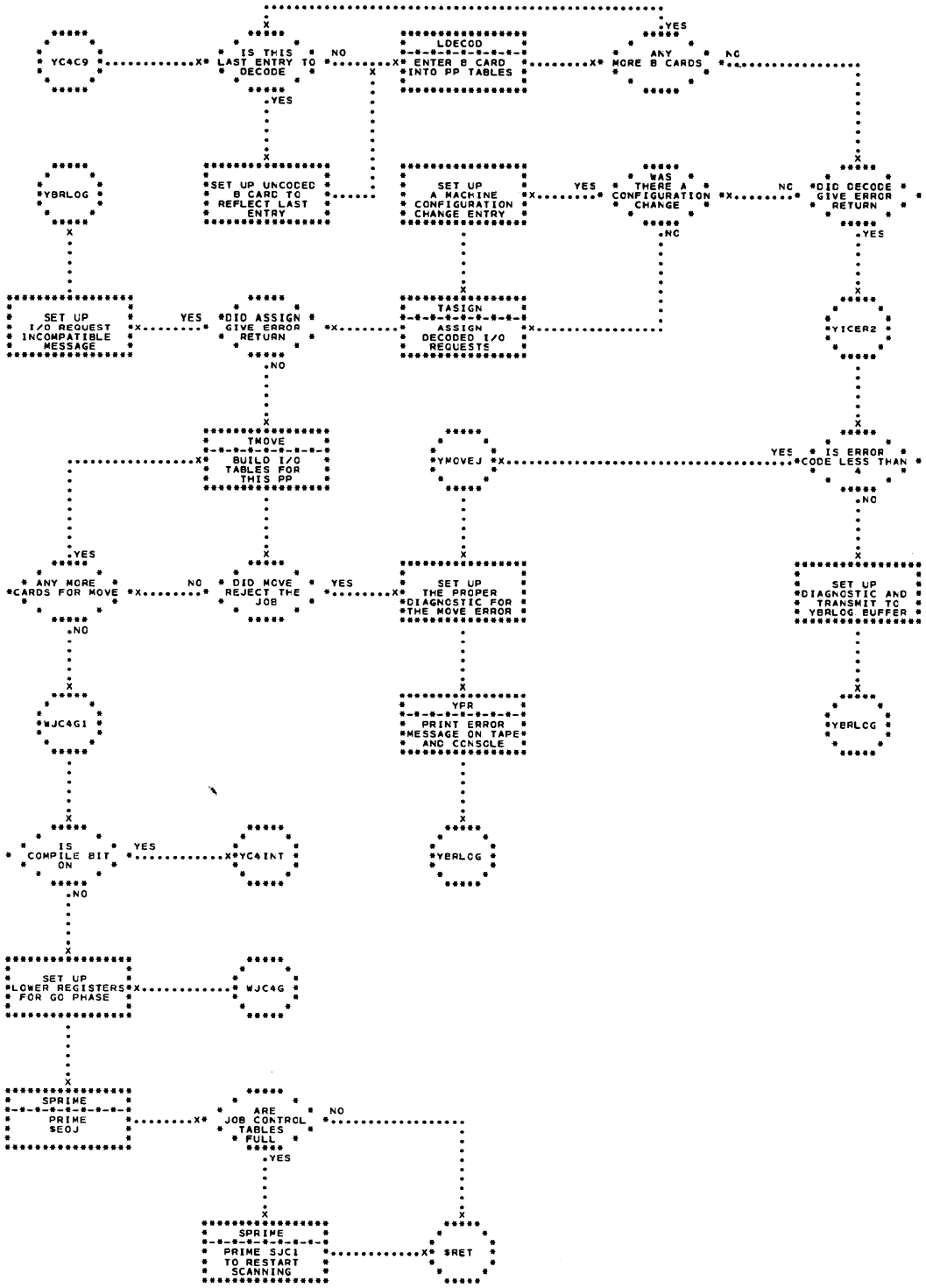


Figure 29. JC4 - Chart 6 - Compiler EOJ

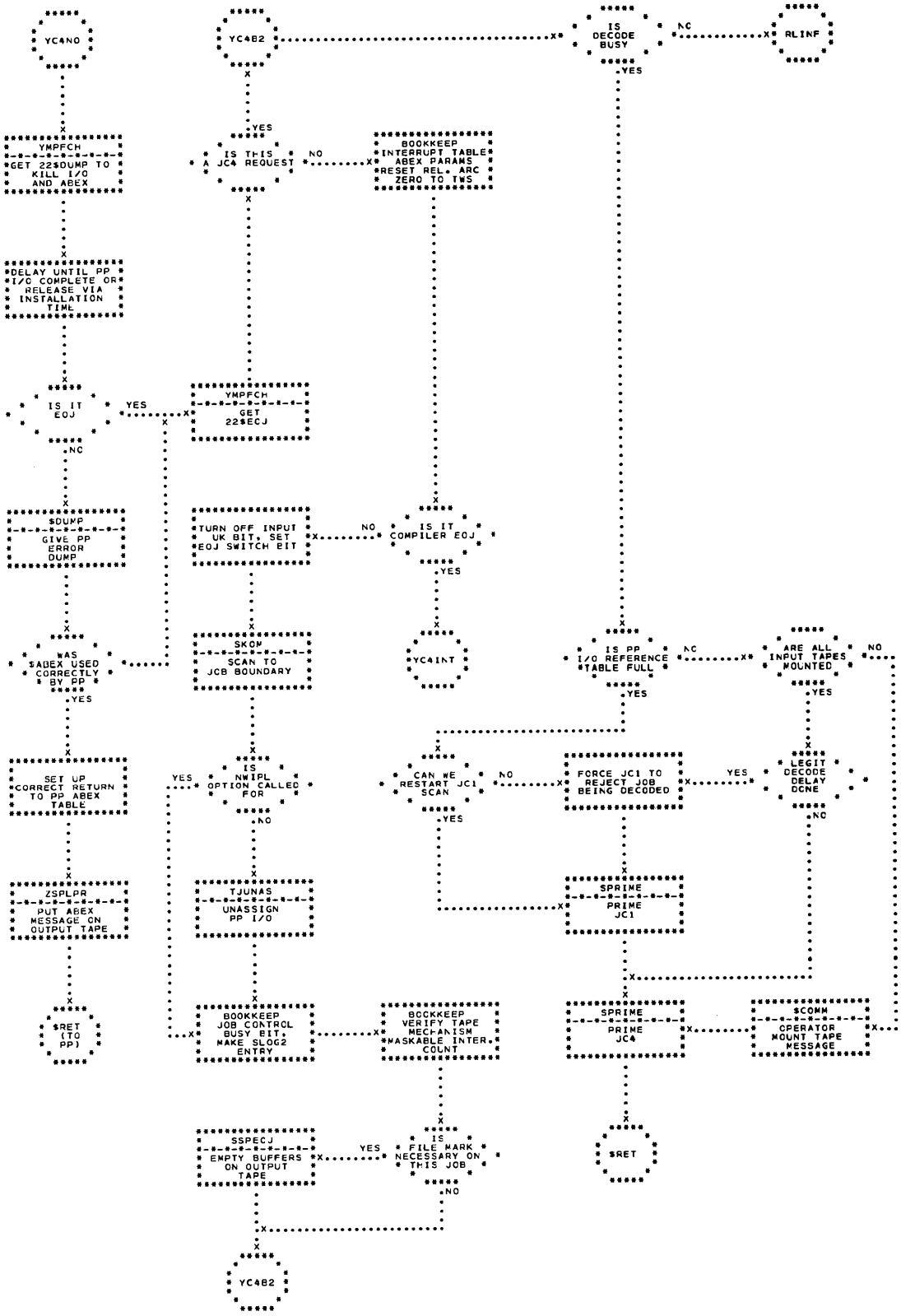


Figure 30. JC4 - Chart 7 - Overlap Mode I-O

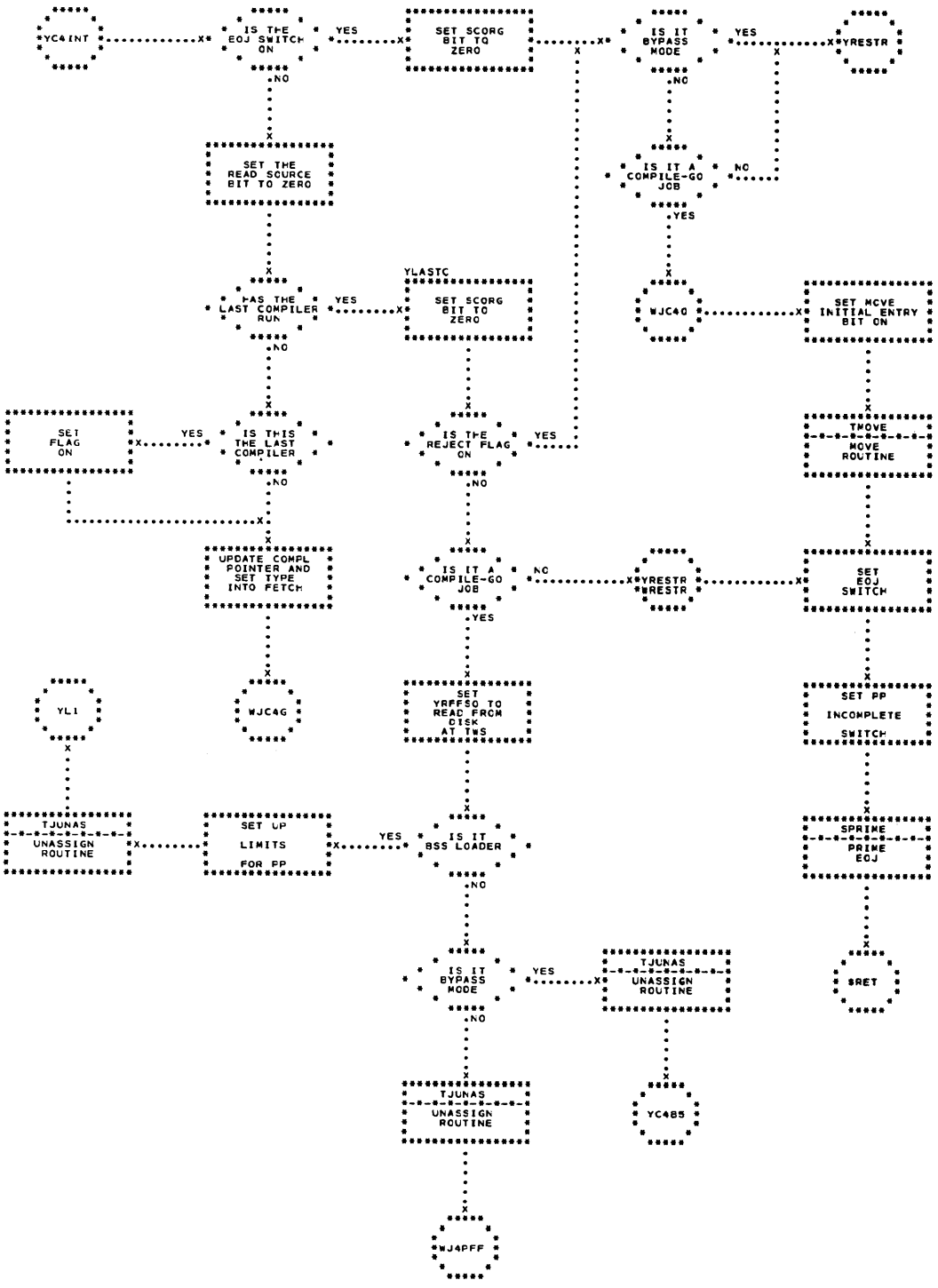


Figure 31. JC4 - Chart 8 - Title Card Analysis

card, the overlapped mode logic borrows some bypass code, at YL1, to process the limits. The bypass code returns to WJCV-1.0 at which point JC4 goes into a loop from WJCV to WMOVIO+1.0 to "move" the IOD and REEL cards. In addition to constructing I-O tables for the PP, Move checks the job name against the name in the PP reference table, and steps the TPPRUN pointer to the next table slot (see I-O assignment). It is therefore necessary for JC4 to enter Move even if there are no IOD cards. This is done at WTEST, the normal exit from the loop, if the initial entry to Move is still on. When the IOD cards are exhausted, the overlapped mode logic rejoins the bypass logic at WJC4G.

Compiler Control

During their execution, compilers are treated by MCP exactly as any other problem program. However, JC4 treats them in a special way for three reasons: (1) their binary decks, LIM and IOD cards are in PROSA on the disk, (2) they may be chained together, (3) their output may be executable programs (COMPILGO). Because of reason 1 above, JC1 does not scan the LIM and IOD cards in the overlapped mode, meaning that IOD cards must be decoded and assigned from scratch by JC4. Therefore, all compiler IOD's are assigned by JC4 in the bypass mode. When a COMPILE or COMPILGO TYPE card is uncoded by JC4, the compiler setup routine is entered at YCOJB (overlapped mode) or at YCOJB1 (bypass mode). At YCOJB (Figure 26), JC4 must first determine whether the job was pre-rejected by JC1 (TRJECT = 1). A COMPILGO job will be run in the compile phase in this case only if the pre-reject was due to decode, and both list and punch options called for, since the compiled deck may then be easily corrected. Otherwise, the pre-reject will be diagnosed at WJ4PFF (I-O assignment will return with reject disposition set). If the job is a COMPILE job and was not pre-rejected by JC1, the TPPRUN pointer is stepped by entering Move at YCPREJ. This must be done to keep the pointer in phase with the jobs being executed, since a slot for this job was created by JC1. The compiler setup begins in earnest at YCOJB1. After fetching the processor chain type-area for the compiler chain named on the TYPE card, JC4 converts the individual compiler names to BCD and "breaks out" the names in order to set up the communication region. When KSILO is initialized, JC4 sets up $13 to cause the read source routine (YRDFSO) to transmit LIM and IOD cards from lower memory. The bypass logic for I-O assignment is then entered at YC4B5.

The code beginning at YC4INT (Figure 29) performs the loading of the next compiler in the compiling chain. It is entered whenever a compiler gives $EOJ or $ABEOJ, and when the I-O has been assigned initially for the chain by JC4. It will terminate compilation immediately if an abnormal EOJ is given (YEOJS = 1), proceeding, if the job is an overlapped COMPILGO, to Move at WJC4Q to advance TPPRUN over the GO phase. Otherwise, KSILO, YDFCS and $13 (read source) are set appropriately, and exit is made to WJC4G. The flow continues to YLASTC when the last link in a chain has run, at which point it might be necessary to run the GO phase of a COMPILGO job. This is accomplished (after setting the necessary parameters) by branching to the appropriate logic (overlapped or bypass) by way of Unassign, going directly to YL1 for BSS jobs.

Error Control

Much of the JC4 code exists to handle error returns from the various subroutines of JC4. These error returns may be categorized as follows:

1. Bypass and compiler errors
2. Overlapped errors
   a. Before I-O assignment
   b. During and after I-O assignment
3. Special (WREJJB)

In Case 1, a message diagnosing the error is printed via YPR, followed by an exit to YRESTR (SYN, WRESTR), which sets the YEOJS and SPINCL appropriately and exits by priming $EOJ and giving a $RET. Case 2a is handled similarly, except that here the TPPRUN pointer must be stepped past the current job. This is accomplished by exiting to WJC4Q, which goes to Move and then to YRESTR. Error returns from Assign and Move make up Case 2b. Here the TPPRUN index will already have been stepped (although Move normally steps TPPRUN, Assign does it when the job is pre-rejected or rejected in Assign). JC4 diagnoses Assign rejects by picking up the address of a message from the TPREFT slot, the address of which in turn is found in TNEXT (see JC1).

Finally, the WREJJB routine is entered each time the first non-COMD card of a job is read. First, the IPL reject count (SREJJB) is checked. If it is non-zero, it is decremented, and the job skipped over by exiting to YRESTR. TPPRUN is not stepped, because no entry was made by JC1. If it is zero, the current job in the PP reference table is checked for YTRB = 1. If it is, it means that although the job was entered in the tables by JC1, the input program was unable to bring it to phase four (repeated tape

failure, tape breakage, etc.). The TPPRUN index must therefore be stepped over the current slot.

Job Control Subroutines

This section describes job control subroutines, including those that actually perform a job control function such as Decode, Assign, Move, etc., as well as those used for such basic tasks as code conversion, even though the latter are not functionally a part of job control.

The Uncode Routine

At many points in job control 1 and 4 it is necessary to verify a B card, convert it to BCD, and get information from it. The uncode subroutine was written to perform this function. The uncode routine is used with the following linkage:

```
        LVI, $14, A
        B, YUNCOD
A       XW, B, C, D
        VF, Disp
        , FWA(I)
        , RA(J)
        (Error Return)
        (Normal Return)
```

where:

Disp denotes the type of card expected:

- 0.0 - TYPE
- 1.0 - JOB
- 2.0 - LIM
- 4.0 - IOD or REEL

FWA(I) is the location of the card.

RA(J) is the location where the converted card is to be put. If RA(J) = 0, no transmission is desired.

Error return signifies that the card designated by Disp was unidentifiable or contained in uncorrectible error.

Normal return signifies that the card designated by Disp has been recognized, converted, and processed.

For a normal return, the XW at A will contain information according to Disp:

Disp	B	C	D
TYPE	---	Bits 1.18-1.35 in communication region	---
JOB	---	Number of fields	---
LIM	Lower limit	Upper limit	---
IOD/REEL	Absolute exit	Number of fields	I-O reference number

Uncode (Figure 32) uses two subroutines, SCA6 and SBRK8, respectively to convert the card to BCD, and break out the fields into consecutive full word locations. (A field is defined as a collection of characters, none of which is a comma, bounded on the left by a comma or column 9 of a card, and on the right by a comma or column 63 of a card.) The first field is compared against its mask in the table at YTYPMK. If the comparison is successful, further information must be extracted from the card. If not, error return is given, and no information is transmitted, unless a JOB card was expected.

The remainder of Uncode consists of three routines to extract information from TYPE, LIM, and IOD or REEL cards. The TYPE card routine has an exit to JC4 if an UPDATE was encountered in the bypass mode. A TYPE, UPDATE card causes uncode to set loader limits to their maximum, the read source ($13) to $SCR, and SCORG to 1 (compile). When the $RESLD package gets control, it will be set up to load C and P cards directly into MCP. Otherwise uncode sets up communication region (KSILO) bits 1.18-1.35 in the calling sequence. JC4 gets the actual compiler name from YUCBF2, the breakout buffer. The LIM card routine converts the limits from BCD to binary, and verifies that they do not violate MCP memory. Note that the checking allows for the possibility that MCP might occupy lower rather than upper memory. The IOD card routine sets up the IOD information in the format required by the decode and move subroutines.

Data and storage used by Uncode are: YIDX12 and YUCX12 are used by the IOD and LIM card routines, respectively, to convert numbers from BCD to binary. YUTOE is the symbolic location of the reference number on a BCD IOD card. Cards are converted to BCD into YUCBF1 and broken out into YUCBF2. YGOMK through YFORTR is the uncode symbol table. Uncode also uses ABSSID, a data entry in the loader, and YNLST and YNOPUN in the JC4 compile and go subroutines.

Decode, Assign, Move

These three subroutines perform key functions in moving jobs through MCP. Decode sets up I-O assignment tables in phase 1 (or phase 4 bypass). Assign associates physical I-O units with symbolic I-O requests, assigning tape units as far ahead as possible. Move sets up the I-O controls to run the immediate job.

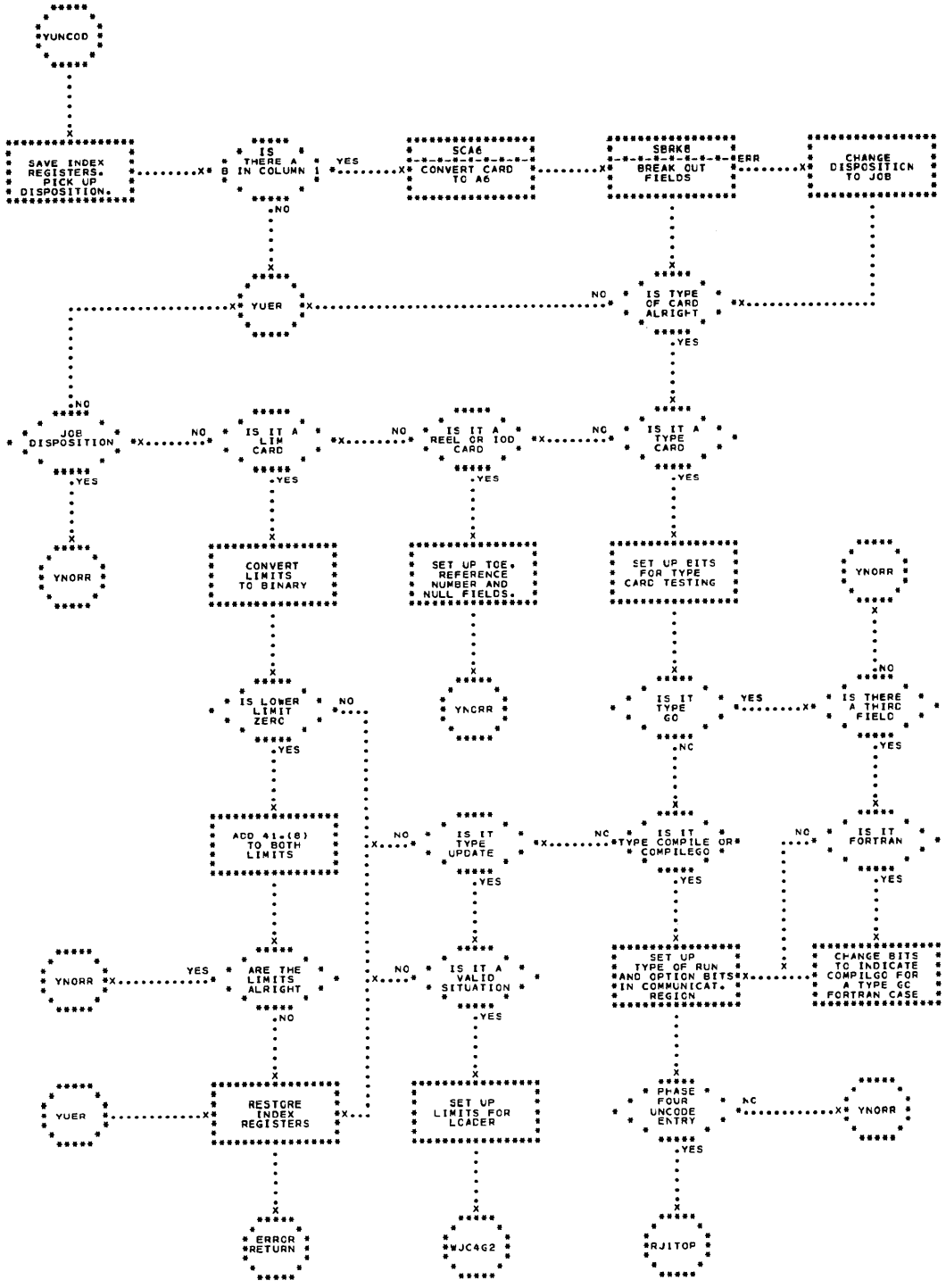


Figure 32. Uncode Routine

I-O Assignment Tables: The I-O assignment tables are the PP reference table, the I-O request table, and the first reel number table. Decode, Assign, and Move are concerned with setting up or responding to these tables.

THE PROBLEM PROGRAM REFERENCE TABLE (TPREFT): This table has one two-word entry for each PP which has arrived at phase 1 but has not been run. The length of the table is controlled by the parameter TPPC, the number of jobs allowed in the table plus one. Presently this parameter is set at 21, to handle 20 problem programs. (e.g., one in phase 4, and ten on each of two input tapes.)

The table control words are as follows:

TPPREF -- a constant index word for the TPREFT table, used for refill.

TPPRUN -- an index word pointing to the next PP to be run.

The format of the first of the two 64-bit words for each PP is as follows:

TCRREF (0.0) -- 18 bits, an address of the first word used within the I-O request table for the PP.

TUNCT (.28) -- 8 bits, is the number of words used within the I-O request table for the PP.

TJBPRO (.36) -- 1 bit, is zero if the PP has not been run or processed. It is one if the PP has been run.

TRJECT (.37) -- 1 bit, is zero if the PP has not been rejected, a one if the PP has been rejected in phase 1.

TASGNP (.38) -- 1 bit, is zero if any one or more of the I-O requests are not pre-assigned for the PP, is one if all requests have been pre-assigned for the PP.

TLPPEN (.39) -- 1 bit, is zero if this word is not the last plus one entry within the TPREFT table, is one if this is the last plus one entry made by decode within the TPREFT.

TLREFN (.40) -- 12 bits, contains the largest reference number found on any one of the IOD's submitted for this PP.

TIODCT (.52) -- 12 bits, the total number of IOD's submitted to decode for this PP.

The second of the two words, TPNAME, is the name of the PP taken from the first 8 characters of the JOB card in A6 format.

THE I-O REQUEST TABLE (TIOREQ): This table contains one word for each unique I-O unit requested by the PP. The length of the table is controlled by the parameter TIOC, which is the maximum number of problem programs multiplied by an estimated average number of units used by each problem program. Presently the parameter is 80. The format of each word is as follows:

TASGNI (0.0) -- 1 bit, is zero if this request word has not been pre-assigned, and will be set to one when pre-assignment has been made.

TUNOBT (.1) -- 1 bit, is zero for normal pre-assignment case. It will be set to one if this request was once pre-assigned and the unit to which it was pre-assigned was taken from the machine configuration.

TPRINT (.2) -- 1 bit, set to one when (if tape request) the tape mounting message has been sent to the operator via the commentator.

TYPE (.3) -- 4 bits, indicating the equipment types being requested.

- 0001 - Disk request
- 0010 - Console request
- 0011 - Card reader request
- 0100 - Card punch request
- 0101 - Printer request
- 1000 - Tape request

TABSUN (.7) -- 3 bits, indicating the absolute unit number to which the request has been assigned. (Multi-unit channels only.)

TRLSYM (.11) -- 7 bits, is a number relative to a position within the symbolic channel table. Hence, this number is relative to the symbolic channel field of the IOD card which is used for channel separation.

TLAST (.18) -- 1 bit, indicates the last plus one word of this I-O request table, used only to note when to refill the value field of the TIOREQ tables control index.

TIODSQ (.19) -- 9 bits, denotes the sequence in which the IOD for this request was received.

TFREEL (.28) -- 18 bits, is the cross-reference address of the reel label within the TFSTRE table. (Tape IOD's only)

TABSCH (.46) -- 18 bits, is the absolute channel address (to locate the Channel Status Table) to which this request has been assigned.

THE FIRST REEL NUMBER TABLE (TFSTRE): This table contains one word for each reel label requested by a REEL card. The reel label is 8 characters long and is in A6 format, using only 48 of the 64 bits. The size of the table is determined by the parameter TFRC, presently 50.

The table entry, TREELN, is the name of the tape reel to be mounted on a tape unit. The first three characters specify whether the tape is labelled or unlabelled and whether the tape is protected or not protected. The remaining five characters are the label.

The preceding three tables are duplicated to handle problem programs in the bypass mode. They are only for one job, however, and the table sizes are chosen accordingly. The formats of the tables are the same as for the overlapped mode.

In the overlapped first reel table, bit .50 (RMULTIRL) is used to indicate that multiple reels have been requested for the IOD. The job-to-job tape passing section of Assign will not match a later job's similar reel request to this multiple reel IOD because the final reel on the drive will not be the reel that we wish to pass to the later job.

Bits .48 and .58-.63 are used internally by the Assign program to keep track of pass tapes that did not arrive at the actual running state.

Table Usage: The usage of these tables is summarized as follows:

PROBLEM PROGRAM REFERENCE TABLE (TPREFT):

	Decode	Assign	Move
TCRREF	S	US	US
TUNCT	S	US	US
TJBPRO	RE	CS	CS
TRJECT	RE	CS	CS
TASGNP	RE	CS	US
TLPPEN	S	US	US
TLREFN	S	NU	US
TIODCT	S	NU	US
TPNAME	S	US	US

S = Set (most cases set to not zero).
RE = Reset (most cases set to zero).
CS = Conditional set.
US = Used but not modified (conditional).
CR = Conditional reset.
NU = Not used, not modified.
NR = Not used and reset.
UR = Used and then reset (conditional).

I-O REQUEST TABLE (TIOREQ):

	Decode	Assign	Move
TASGNI	RE	CS	NR
TUNOBI	RE	CS	NR
TPRINT	RE	CS	NR
TTYPE	S	US	UR
TABSUN	RE	CS	UR
TRLSYM	CS	U	UR
TLAST	US	US	UR
TIODSQ	S	NU	UR
TFREEL	CS	US	UR
TABSCH	RE	CS	UR

THE FIRST REEL TABLE (FSTRE):

	Decode	Assign	Move
TREELN	CS	US	UR
RMULTIRL	S	US	NU

THE CURRENT REEL ON DRIVE TABLES (SBCRODT): This table contains one full word for each of the possible twenty-four tape units (6 units on each of 4 channels) with a corresponding job name table of 24 one-word entries. The format of the one-word entries for each of the two tables is as follows:

Bits	Content
0.00-0.29	The name of the reel currently on this tape unit
0.30	The reel protection indicator (0 - not protected) (1 - protected)
0.31	The reel label indicator (0 - not labeled) (1 - labeled)
0.32	The Psave indicator (0 - not pass save reel) (1 - pass save reel)
0.54	The final disposition indicator (0 - not save) (1 - saved)
0.55-0.60	The Channel number (32-33-34 or 35)
0.61-0.63	The Unit number (0 through 5)
24.00-24.63	The IQS name of the job which originally requested the tape for this unit.

A summarization of the use of these tables follows:

Bits	Assign	Move	Unassign(or SFREE-SUNLD)
0.00-0.29	US	CS	CR
0.30	US	CS	CR
0.31	US	CS	CR
0.32	CS	CS	CR
0.54	NU	NU	UR
0.55-0.60	NU	NU	NU

Bits	Assign	Move	Unassign (or SFREE-SUNLD)
0.61-0.63	NU	NU	NU
24.0-24.63	NU	NU	NU

The Decode Routine: The decode routine is used by job control to enter information from IOD and REEL cards into the I-O assignment tables. It is entered by JC1 in the overlapped modes, and by JC4 in the bypass mode, with the following linkage:

```
      LVI, 15, X
      B, LDECOD
X     VF, ALPHA
      CF, B
      VF, C
      error return
      normal return
```

where ALPHA locates the information (see following), B identifies the system mode (zero is overlapped, non-zero is bypass), and C is used to contain the error code in the event the error return is used.

The table at ALPHA contains the information for the IOD or REEL card as follows:

Location	Content
ALPHA-1.0	Job name in A6 code.
ALPHA (.28-.46)	Number of broken out fields following.
(.52-.63)	IOD reference number.
ALPHA+1.0	IOD or REEL.
+2.0	Type or first label.
+3.0	PTOE or second label etc.

When all the requests for a job have been processed, decode must be notified in order to separate I-O requests from successive jobs. This final entry is accomplished by setting the word at ALPHA+1.0 to zero. Also, after several entries have been made in the tables, it may be necessary to reject the job and clear these entries. This is accomplished by setting bit .26 of X in the linkage and using the final entry procedure. This bit must be zero for a normal entry.

The decode routine will use the error return for one of two reasons:

1. The IOD or REEL card is invalid. The corresponding error codes are:
   - 5.0 A REEL card following a non-tape IOD.
   - 6.0 Invalid type field on IOD.
   - 7.0 Two IOD's requesting infinity disk on the same channel.
   - 8.0 The first card submitted was not an IOD card.
2. One of the I-O assignment tables is full. The error codes used are:
   - 1.0 The PP reference table is full.
   - 2.0 The I-O request table is full.
   - 3.0 The first reel table is full.
   - 4.0 The internal tape channel and unit tables are full.

The I-O request and first reel tables will be cleared for the job in question prior to the error return when the error is due to the first reason. When error return is given to JC1 for the second reason, JC1 sets SJ1FUL and issues $RET.

The decode routine (Figure 33) checks the system mode to set up the program for the correct table entry. A mode change is allowed only between jobs. If the entry is an initial entry, the routine clears and sets up certain internal tables. Then, if overlap, a check is made to see whether or not a slot is available within the PP reference table (PPREF). If not, a type 1 condition is set up in the linkage, indicies are restored, and the error return is made. If a slot is available, the program proceeds to check if the entry is a final entry (IOD/REEL zero in ALPHA+1). If so, the PP to be entered is a PP without IOD cards, or possibly a PP that was rejected by job control.

If the entry is not a final entry, the flow follows one of two paths depending on whether the card is IOD or REEL. If it is IOD (LIODRN), the I-O request table is checked for room, and LIODTP entered if the type is tape. A tape IOD has both channel and unit fields for separation, while the remaining I-O types have only a channel field for separation. The I-O request table (TIOREQ) is then set up with proper entries. For each non-tape IOD card with a unique type or channel field, there will be an entry made in the TIOREQ table. For a tape IOD an entry will be made in the TIOREQ table for each IOD card with a null channel and unit field, and each IOD card with unique channel and unit fields. In the case where channel fields are alike and unit fields differ a TIOREQ table entry is made requesting the same channel as the previous IOD card with that channel. Making a table entry consists of setting up the following fields in the TIOREQ word:

- TYPE identifying the equipment type.
- TRLSYM for channel separation (tape only).
- TIODSQ the sequence number in which the IOD was received.

If the IOD card has a TYPE field requesting a disk, a check is made to see if it is an illegal request for infinity disk. If so, the error return is made. Otherwise, a normal return is made.

If the card is a REEL card, it will be processed only if it is the first REEL card following a tape IOD. If the last IOD was not a tape IOD, the error return procedure is followed. If the card is not the first reel card for the IOD, it is ignored. The multi-reel bit is set to one in the first reel table, and the normal return made. (On a single unit, mounting instructions for tape reels other than the first are of no value

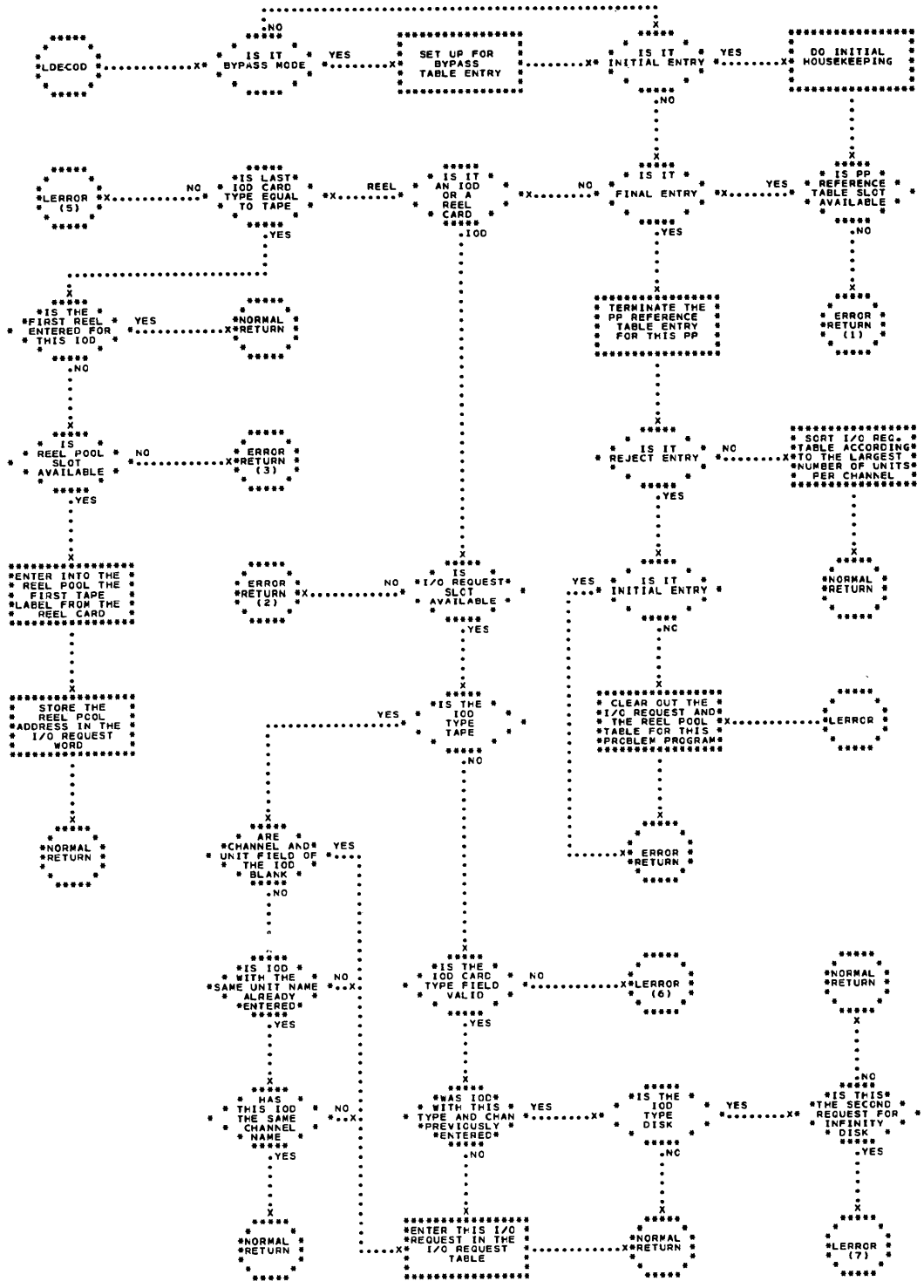


Figure 33. Decode Routine

until PP has finished with the first. Since this will not happen until phase 4, pre-assignment is complete when the first reel is called for.)

If the REEL card is acceptable, the reel label is entered in the first reel table, and its address is entered in the I-O request table. A normal return is made. If a final entry is made and the reject bit is off, the PP reference table entry is made for this PP. These entries are defined in the preceding section on I-O assignment tables.

When an invalid card is detected or a reject request is received, the program (LERROR) clears any entries it had made in the I-O request and first reel tables for this job, and sets the reject bit in the PP reference table.

The Assign Routine: The assign routine (Figures 34-39) pre-assigns the problem program requests which were submitted through IOD cards. This enables the operator to set up in advance the tape units to be used by the problem program. The set up of tapes as far in advance as possible optimizes set-up time. The basic rule used for assignment is to utilize all tape units when they become unassigned. At assign time, all requests are checked for adherence to the machine I-O configuration. If there are any discrepancies, the problem program will be rejected. The assign routine is entered from JC4 with the linkage:

```
        LVI, $15, Y
        B, TASIGN
    Y   XW, A, B, C
        (Return)
```

where

A is the status of the machine:
- 0 - no configuration change has taken place.
- ≠0 - configuration change has taken place.

B is the exit disposition:
- 0 - next PP has been assigned successfully and is ready for Move.
- ≠0 - next PP has been rejected and cannot be given to Move.

C is the Assign entry mode:
- 0 - overlapped mode.
- ≠0 - unoverlapped mode.

Assign uses SA8IQS to convert and $COMM to output messages to the operator.

Upon entry to Assign (TASIGN, Figure 34), the first check made is whether or not a machine configuration change took place since the last time the assign routine was entered. Conditions for stating a configuration change must be set by job control for the initial entry to Assign in order that the necessary tables for I-O validity checking be set up. The configuration change section performs the following functions:

1. All requests reserving units that have become unavailable due to a machine configuration change, must be relieved of these units. At the same time these relieved requests must be marked so that they have priority in re-assignment (pseudo-unoverlapped).
2. The equipment count is generated for the validity check table. This table contains the equipment code and count and a zero field for the requested count for a PP.
3. The multi-unit count table is generated. This gives the total number of tapes on a channel. The format of the table is half-word: .0-.17 the channel number and .18-.21 the unit count. Along with this, the largest count per channel is stored for PP request checking.

Upon returning from the configuration change section (or if there was no change), a mode check and set up is made. If the mode is overlapped, the index pointing to the PP reference table is set to the next slot in the table. This contains the information about requests which have not been completely assigned. If the mode is unoverlapped, this index will be set to the unoverlapped table.

If the mode is overlapped, it is necessary to cycle through all preassigned tape requests. As each preassigned tape request is encountered, the channel and unit numbers in the TIOREQ table (TABSCH and TABSUN) are used to turn on the overlapped reserved bit in the Unit Status Table (SOVRES). This is necessary because of tape passing from job to job. Consider the following situation: Job A and Job C both use the same tape. Both Job A and C turn on the SOVRES bit (Job C does so redundantly). Later when Job A goes into the running phase, the Move program sets the SOVRES bit to zero. Since Job C has already turned on the SOVRES bit earlier, it cannot do so now; therefore the tape unit appears to be available and is to be reserved for a later Job E.

The Assign routine refers to the PP reference table to get the information about the PP: rejected, completely assigned or already processed. If the PP is accepted, the table contains the cross reference address to the I-O request table and, if any I-O was requested by the PP, the count of the number of individual units requested. This count is used to set up the minor index (TIXA) which is used for scanning the I-O request table. With the exception of the print bit, the information was computed and stored in the I-O request table by the decode routine. Information pertaining to the assignment of a unit, such as the absolute channel address, the absolute unit number and the assigned bit, are computed and stored by the assign routine. The unoverlapped bit that appears in the I-O request table is a special indicator denoting the fact that this request was once assigned and was dis-assigned due to a machine configuration change.

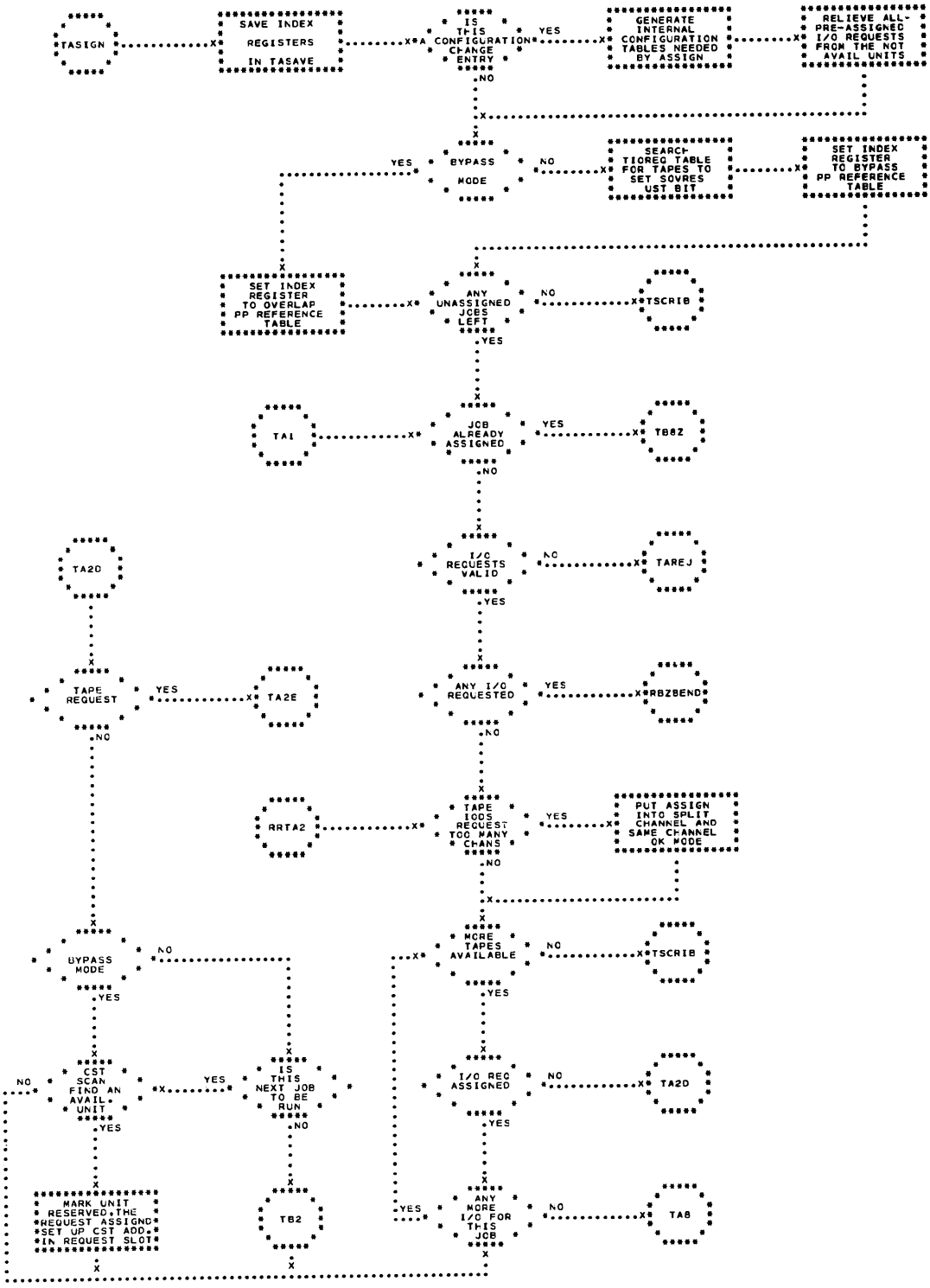


Figure 34. Assign Routine - Chart 1

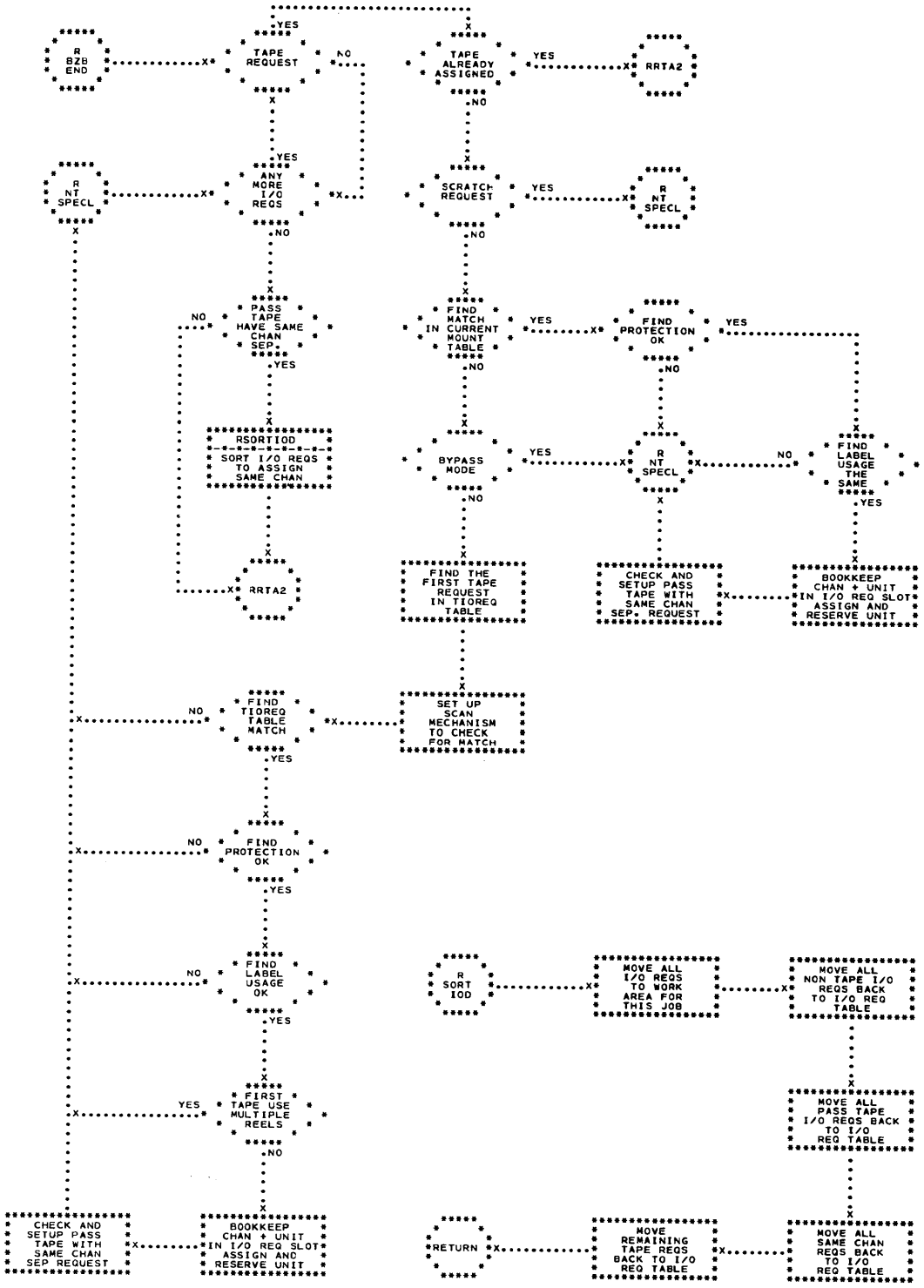


Figure 35. Assign Routine - Chart 2

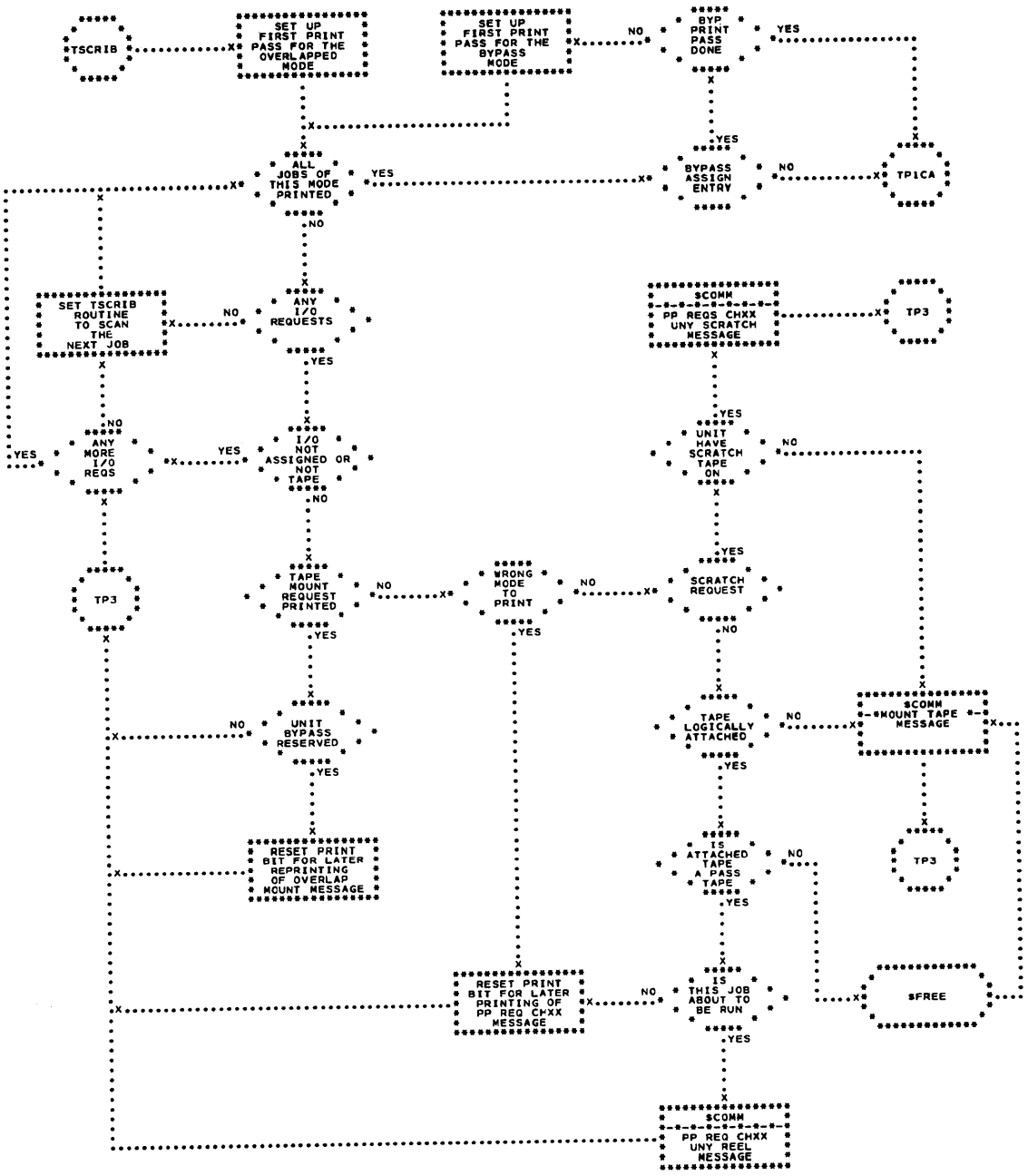


Figure 36. Assign Routine - Chart 3

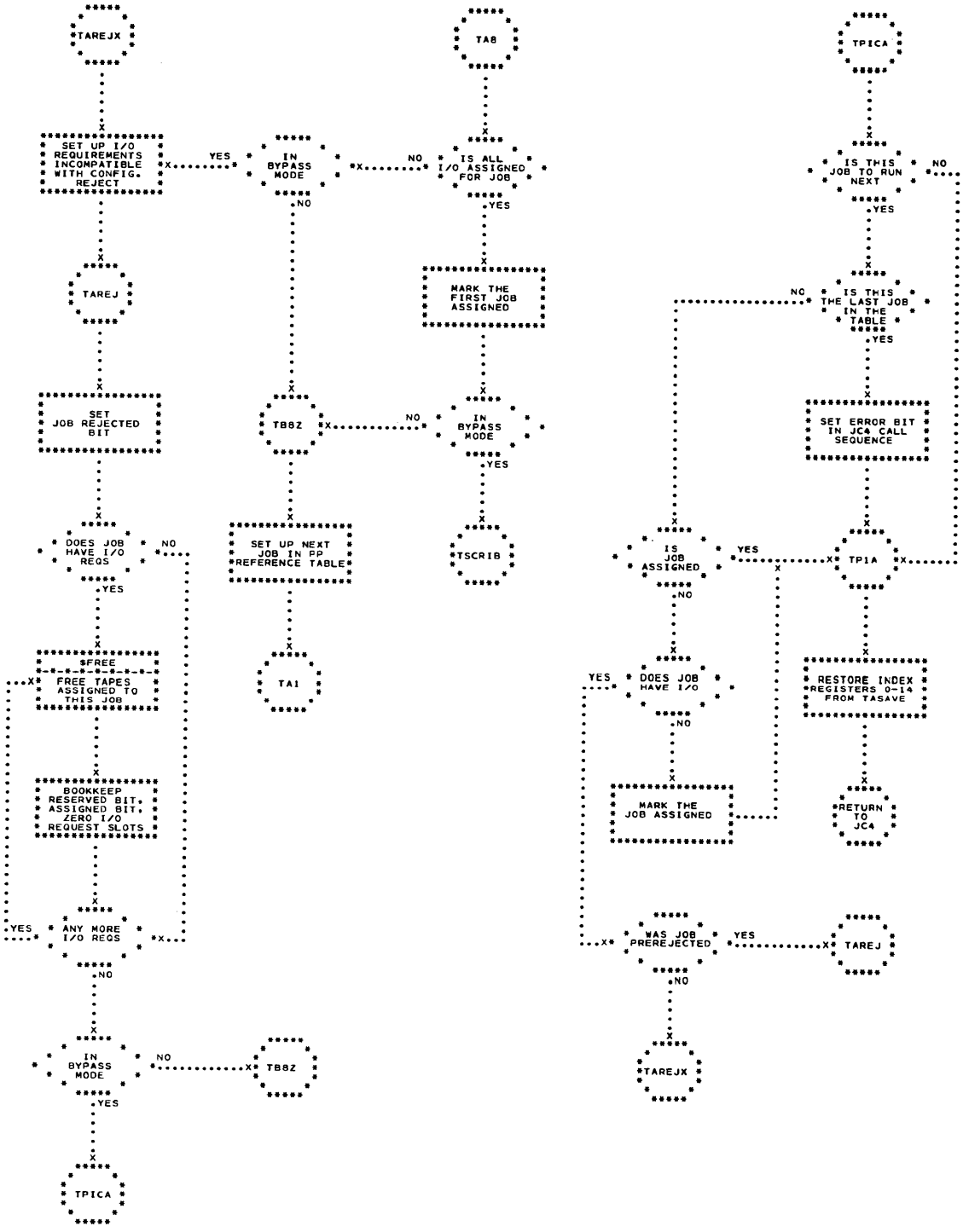


Figure 37. Assign Routine - Chart 4

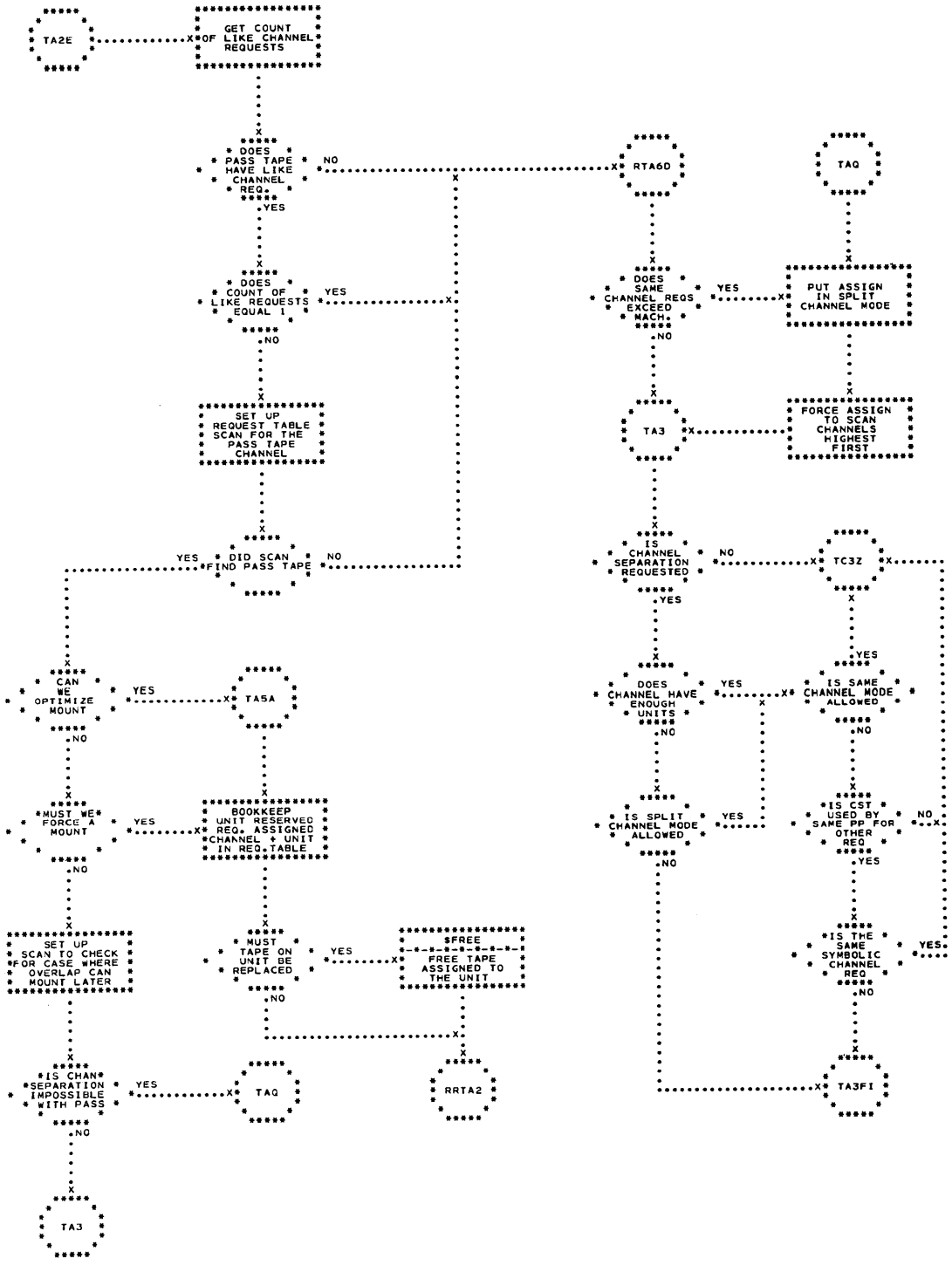


Figure 38. Assign Routine - Chart 5

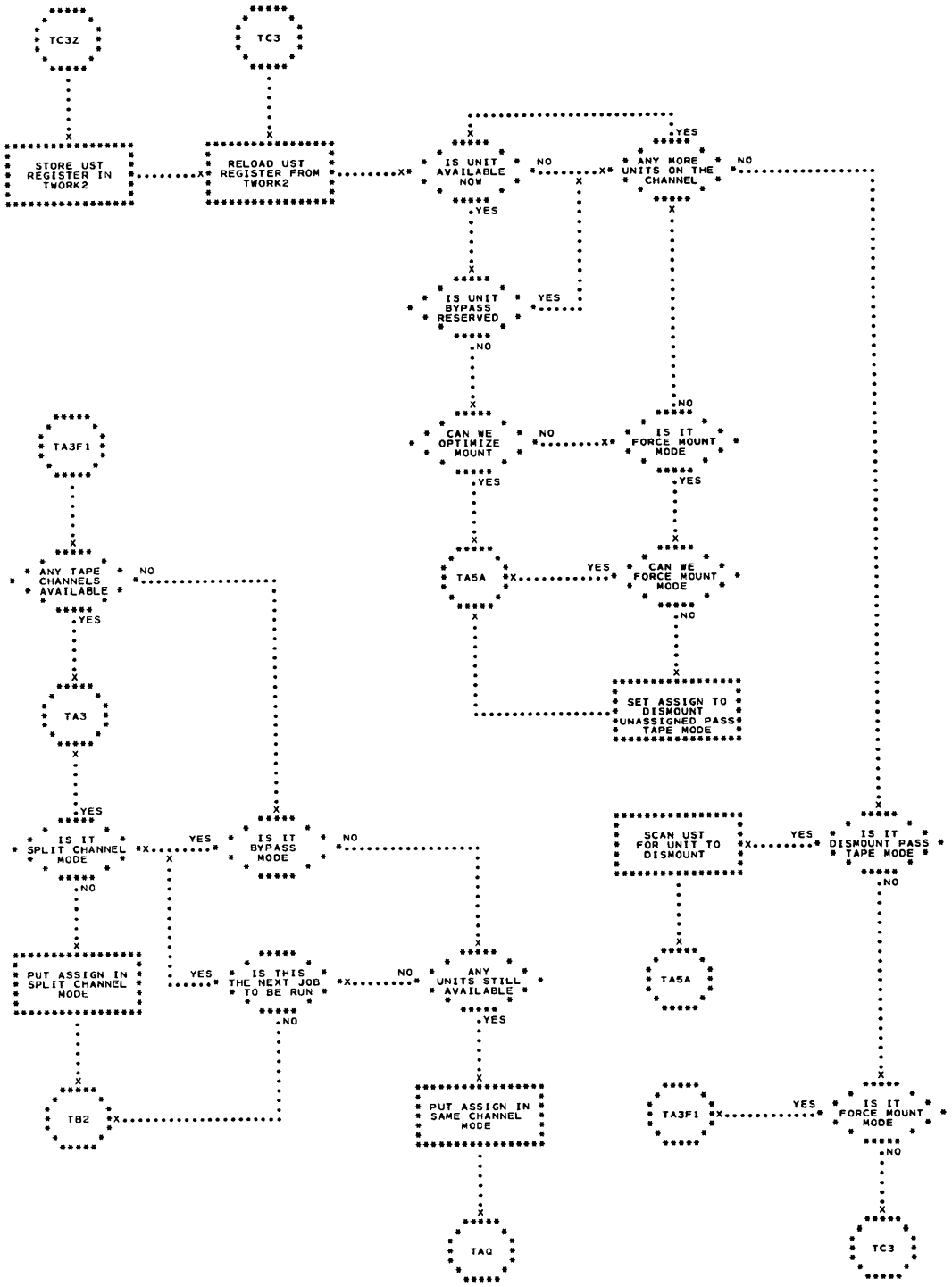


Figure 39. Assign Routine - Chart 6

When the unoverlapped indicator is on, this request has top assignment priority and if there is no unit available to satisfy this request, the assign routine will reject the PP. If there is a unit available assigned to a request of a PP that is to run at a later time, then the assign routine will relieve that PP of the unit and give it to the top priority request. The PP that lost the unit to the priority request will be assigned a unit as soon as one becomes available.

The first reel table is used to provide the first label or reel name for the mount message. This message is printed as soon as there is a unit available for the IOD request. If there was no REEL card after the tape IOD, the request is then considered to be for a scratch tape. In this case, the first reel entry in the I-O request table will be zero.

All three of the I-O assignment tables are used in a circular fashion.

The I-O assignment procedure is as follows:

1. Each PP is checked to see if the requested IODs are valid. If not, the PP is rejected.

2. If the PP to be assigned has no tape requests, the tape passing mechanism is skipped.

3. Requests for specific tapes are checked against current tape unit mountings (as per the SCRODT table). The I-O request is assigned if the requested reel is mounted.

4. If the system is in an overlapped mode, unassigned tape requests are checked against the I-O request table (TIOREQ) for the same reel request. If the reel is matched and protection and label usage is correct, the request is marked assigned.

5. If a tape request has been preassigned because of steps 3 or 4, a check is made for another tape I-O for the same channel within the PP. If such a request is found, the PP I-O requests are reordered in an attempt to maintain the channel separation mechanism. The following illustrates a typical reordering.

Before	After
TAPE CH1	DISK
DISK	CONSOLE
TAPE CH4 UNA	TAPE CH4 UNB
CONSOLE	TAPE CH3
TAPE CH2 UNZ	TAPE CH4 UNA
TAPE CH1 UN2	TAPE CH1 UN1
*TAPE CH4 UNB	TAPE CH1 UN2
*TAPE CH3	TAPE CH2 UNZ

* Indicates that this tape has been matched to either the SCRODT or TIOREQ tables.

6. If any tape requests are found that have the same separation as a matched tape, they are preassigned now.

7. Any other tape requests are preassigned now. All non-tape requests are assigned when the job is about to be run.

Working on one PP at a time, each individual I-O request is compared with each unit in the I-O configuration status tables to see whether or not the unit satisfies the request. If the request is satisfied as far as the type and channel grouping is concerned, further checks are made to see if the unit is available, unassigned, and not reserved. In the case of tape, a check is made to determine whether or not optimum reel mounting (the program tries to place a scratch tape on a unit that has a scratch tape already mounted) is to take place. If the unit is assigned or not available, it will be ignored. If the unit is reserved, a test is made to see if the requesting PP is being entered in the bypass mode. If it is not, the unit will be ignored as this unit is reserved for a PP in queue ahead of this PP. If the mode is bypass and the I-O request is otherwise satisfied, then this request will override the previous assignment for that unit.

The pre-assignment will be carried out, processing one program at a time, either until there are no more units available for reservation or there are no more requests to process. After processing a program, a check is made to see whether or not the program is completely pre-assigned. If it is, the assigned bit is set to one in the PP reference table, signifying that all units have been pre-assigned so that continual scanning of the PP's I-O request table is not necessary upon entry to the assign routine. At the same time as the above check, there is another check to find out whether or not the mode is unoverlapped. If it is, then a quick exit is made from assign disregarding the remaining PP's that are in the queue.

Before any exit is made, all tape mountings necessary will be printed for the operator via the commentator. The disposition field of the linkage will also be adjusted according to the status of the next PP to be run. If the status is reject, the pointer (TPPRUN) will be advanced to the next PP. This pointer is used by the move routine and should always be synchronized with the incoming problem programs via the input program.

All the information necessary for assignment is stored away in the I-O assignment tables. This information is later transformed and merged with the IOD card to form the Actuator tables. This transformation is carried on by the move routine.

The Move Routine: The move routine (Figures 40, 41) obtains and merges IOD or REEL cards with information generated by the assign routine to form the four actuator tables: symbolic I-O location table, file area tables, unit area tables, and the reel pool table. Certain system parameters will also be set up by the move routine: the maximum reference number used by the problem program, the next available full word

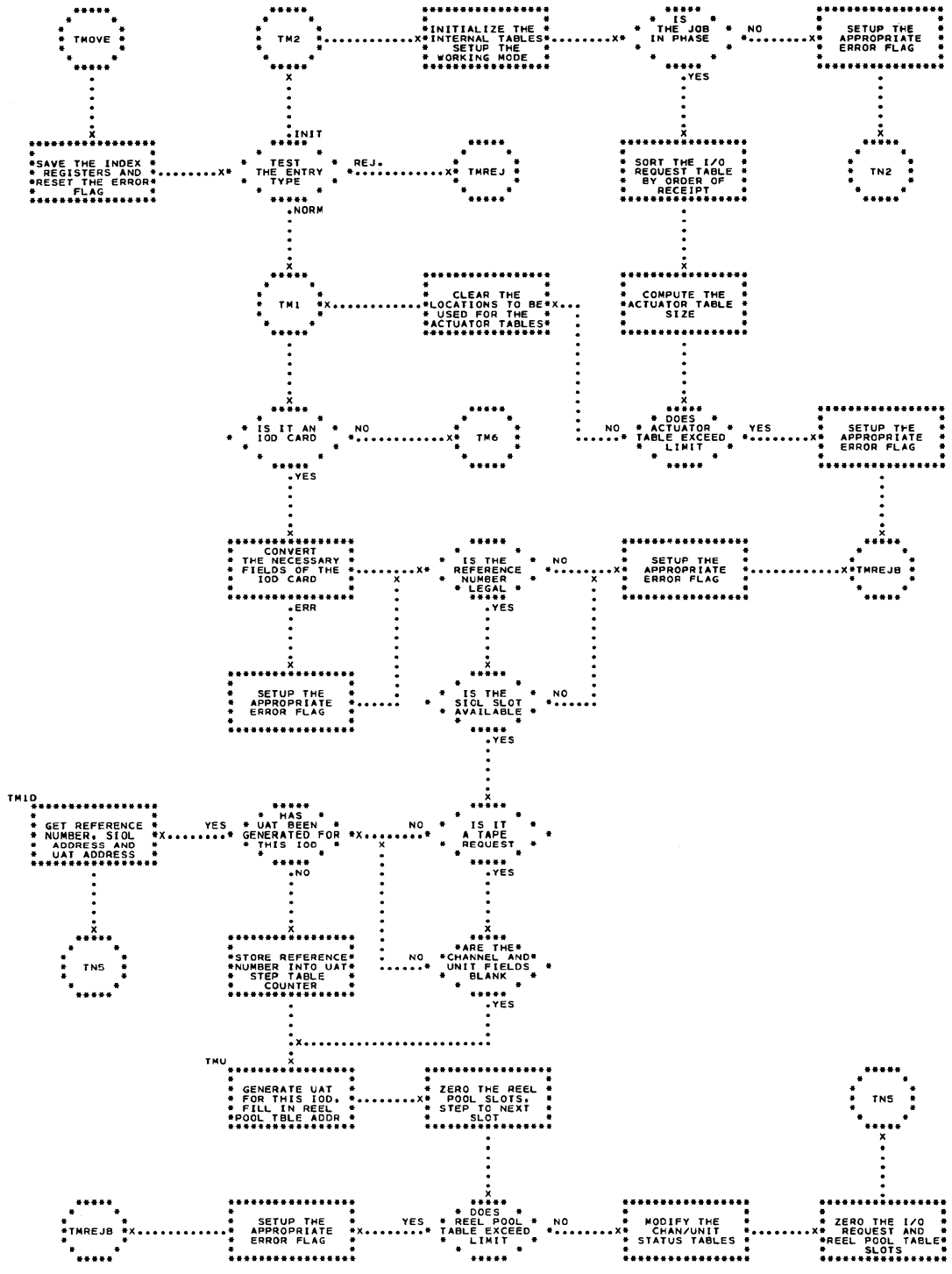


Figure 40. Move Routine - Chart 1

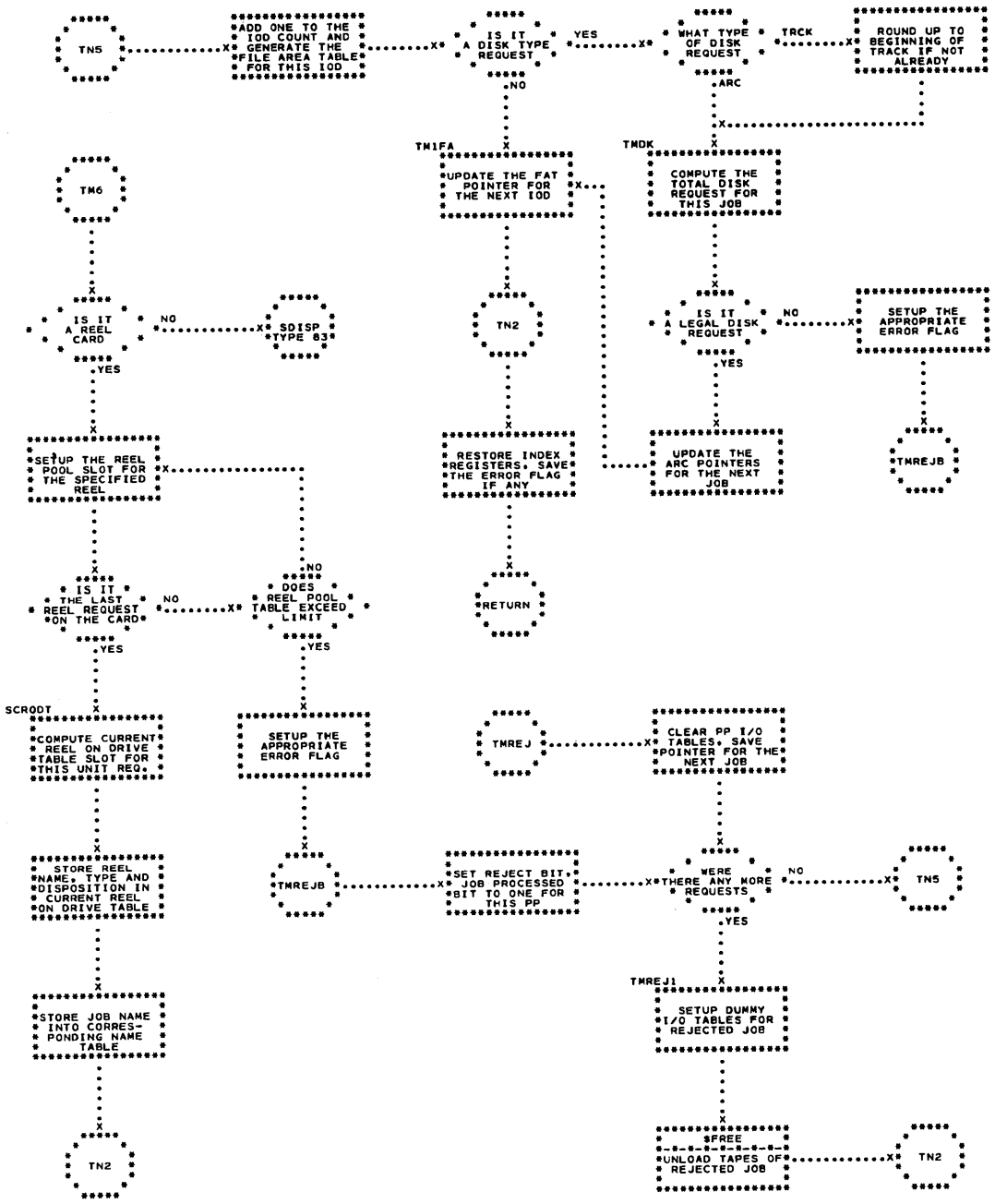


Figure 41. Move Routine - Chart 2

following the reel pool table, and the arc address of the next available arc that is unused by the problem program. The move routine is entered with the linkage:

```
        LVI, $15, Z
        B, TMOVE
Z       XW, A, B, C, D
        (return)
```

where:

A is the first word address of the IOD or REEL card breakout:
- + for normal entry
- - for initial entry

B is the return disposition:

Bits 28-30 indicate IOD errors:
- 0 - none
- 1 - in disposition field
- 2 - in density field
- 4 - in mode field

Bits 44-47 indicate other errors, if any:
- 0 - none
- 1 - illegal disk request
- 2 - out of phase
- 4 - exceeding reel pool table
- 8 - zero I-O reference number

C is job control reject indicator:
- 0 - means accept next PP
- 1 - means reject next PP

D is mode and owner indicator:

Bit 25 is mode indicator:
- 0 - means overlapped
- 1 - means unoverlapped

Bit 26 is ownership bit:
- 0 - PP owns IOD
- 1 - MCP owns IOD

The move routine (TMOVE, Figure 40) is used by JC4. The initial entry for each new problem program is noted to the move routine by a negative index found in the linkage. This initial entry allows the move routine to initialize itself for the next PP by resetting certain move parameters, computing the size and resetting the memory area used by the actuator tables, and checking the validity of the PP about to be run. After the initial entry procedure has been performed, control is sent to the beginning of the move routine (TM1) as in the normal entry.

At TM1, the routine checks for the type of card. If IOD, the card is decoded into a format more usable by the remaining routine. A check is then made to see whether the request for this unit via another IOD card has been submitted prior to this IOD card. If not, a new Unit Area Table is constructed and the channel unit status table slot for the unit assigned to this request is modified. The information about the absolute assignment is taken from the I-O request table. If the unit has been requested by another IOD card prior to this IOD card, the construction of the unit area table and the modification of the channel unit status table are bypassed. All IOD cards cause the move routine to construct a file area table. If the IOD submitted is for tape, a two-word slot is automatically reserved within the reel pool table whether or not a REEL card follows.

When a REEL card is submitted to the move routine, control is passed to the section that constructs the reel pool table (TM6). This section looks back at the last IOD card submitted and finds the unit area table constructed for that IOD card. It then takes the address of the reserved slot within the reel pool table to see if this slot is available for a reel name entry. If it is, the reel pool table continues to be constructed until the reel names are exhausted from the REEL card. If it is not, the move routine scans the reel pool table until it finds a vacant slot for the reel name. When the reel pool table is set up for each reel, the Current Reel on Drive Table is set up with the reel name for the specified tape unit, and the PP name which requested the unit.

If a disk IOD card is submitted and the type is DISK, the next available arc is assigned to it. If type is TRACK, the next available arc address is rounded forward to the next track address, then this IOD request is assigned to that address. Hence, for optimum disk assignment or allocation, a PP should submit all disk requesting IOD's in a grouped fashion, all IOD's within the type field of DISK together, then all IOD's with the type field of TRACK.

After performing the above functions for each IOD/REEL card submitted, control is returned to job control via the TN2 return.

In the case of a reject entry to Move from job control there are no checks made for proper phasing. The next PP appearing in queue to be run is rejected by clearing out the I-O request table entries and reel pool table entries. Then all the non-scratch tapes assigned for this PP that were previously mounted by the operator must be removed from the respective tape drives. The tape unloading procedure is carried out by using the $FREE service routine. The reject routine tests the entry mode. If the mode is bypass, the rejection is made in the not-overlapped PP reference (TPURFT), I-O request (TUIORQ) and reel pool (TUFRE) tables. If the entry mode is overlapped, the rejection is made in the overlapped tables for this particular PP. In the overlapped mode, the TPPRUN and TSAV index words are stepped to point to the next PP to be run. After the rejection is complete, the exit from move is from normal exit TN2 with no error flag.

The TN2 return sets up SMARK (SYN, TMARK) with the last memory cell used by move in setting up the actuator tables. The preset error flag index is then stored in the return linkage to job control, the index

registers restored, and control returned to job control.

THE ACTUATOR TABLES ALLOCATION ALGORITHM:

Let B equal the B limit (higher) from the LIM card.
Let S equal the first word of the Symbolic I-O Location Table.
Let R equal the largest Reference Number per problem program.
Let F equal the first location of the File Area Table.
Let I equal the total number of IOD cards for problem program.
Let U equal the first location of the Unit Area Table.
Let R' equal the total number of unique units requested by problem program.
Let P equal the first address of the Reel Pool Table.
Let N equal the number of individual REEL requests, plus one for every IOD card with the type of TAPE that was submitted without a REEL card.
Let TMARK equal the next slot available after the Reel Pool Table.

then:
S = MAX (B, 30000_8)
S + R = F
F + (I * 7) = U
U + (R' * 9) = P
P + N = Contents of TMARK

The Unassign Routine: The unassign routine (Figure 42) makes the I-O units used by the previous problem program available again for assignment, unloads and rewinds the tapes used and tells the operator what to do with the unloaded reels. This routine is used only by JC4, and is entered by the following calling sequence:

LVI, 15, (return address)
B, TJUNAS

It saves and restores the used index registers. Two subroutines are entered: SFREE (an actuator op) and the reel history routine (JHISTE). SFREE effects the unassignment of tape units, and JHISTE prints via the output program a record of the tape reels used by the problem program.

After saving index registers, unassign initializes the reel history routine and tests TMAXRF, or SMAXRN, to see if the previous job had any IOD cards. Starting at TJNEXT, the program loops through successive entries in the PP symbolic I-O location table. Unassignment proceeds on a unit basis. As a unit is checked out, the SUUNAS bit is set to one in the unit area table, preventing repeated unassignment of a unit with several IOD cards attached to it. For tape units, the mount bits are set to zero (to guarantee entry to the actuator), the SFREE routine is used to unassign the unit, and then the reel history routine is entered. For non-tape units, unassign first waits for the unit to become not busy, after which the unit is released. Any repeated EKJ's result in a type 76 error exit to SDISP. For both kinds of units, the appropriate channel and unit status table bits are reset at the end of the loop (at TJTAP and TJNTPS), and the disabled modes. After all units are unassigned, any stacked PP interrupts are discarded from SQUE, and the SAS and SSIO bits in the program status table are cleared.

Aside from the index saving area, unassign uses only one word of working storage, TJCW, to copy control word into. It always returns to the address in $15.

JC4 Print Program

This program prints out JC4 and $RESLD error diagnostics on the system output tape, and allows each installation to modify the routine easily so that these messages may be put out via the commentator. The unmodified routine (YPR) uses no index registers, and employs the short message routine (ZSPLPR) to effect the printout. All messages must be 8 words long and in A8 code.

The modified routine uses $2 and does not restore it, uses SA8IQS to convert the message, and $COMM to print the message online.

The routine may be modified by putting in a NOP at location YPR with any of three effective addresses:

1. NOP, 0.0 produces messages on-line only in the BYPASS mode.
2. NOP, 1.0 produces messages on-line in both the BYPASS and ONLINE modes.
3. NOP, 2.0 produces messages on-line for all modes (BYPASS, ONLINE, OFFLINE).

Major Package Fetcher

This routine calls in segments of MCP from the disk as required, and keeps track of the current segment in core. It is entered at YMPFCH, and on the basis of the content of $1 and the current value of YMPSAV, decides whether or not a fetch is necessary. If not, it returns immediately at YFHBR. If necessary, the fetch is made via $FETCH, and the fetch parameter CBUFF, and CFOT are reset to indicate an empty fetch buffer. Return is then made to the fetched segment at FHBR.

The Reel History Routine

The reel history routine (JHISTE) prints the job name, the Channel and unit number, the IOD number which

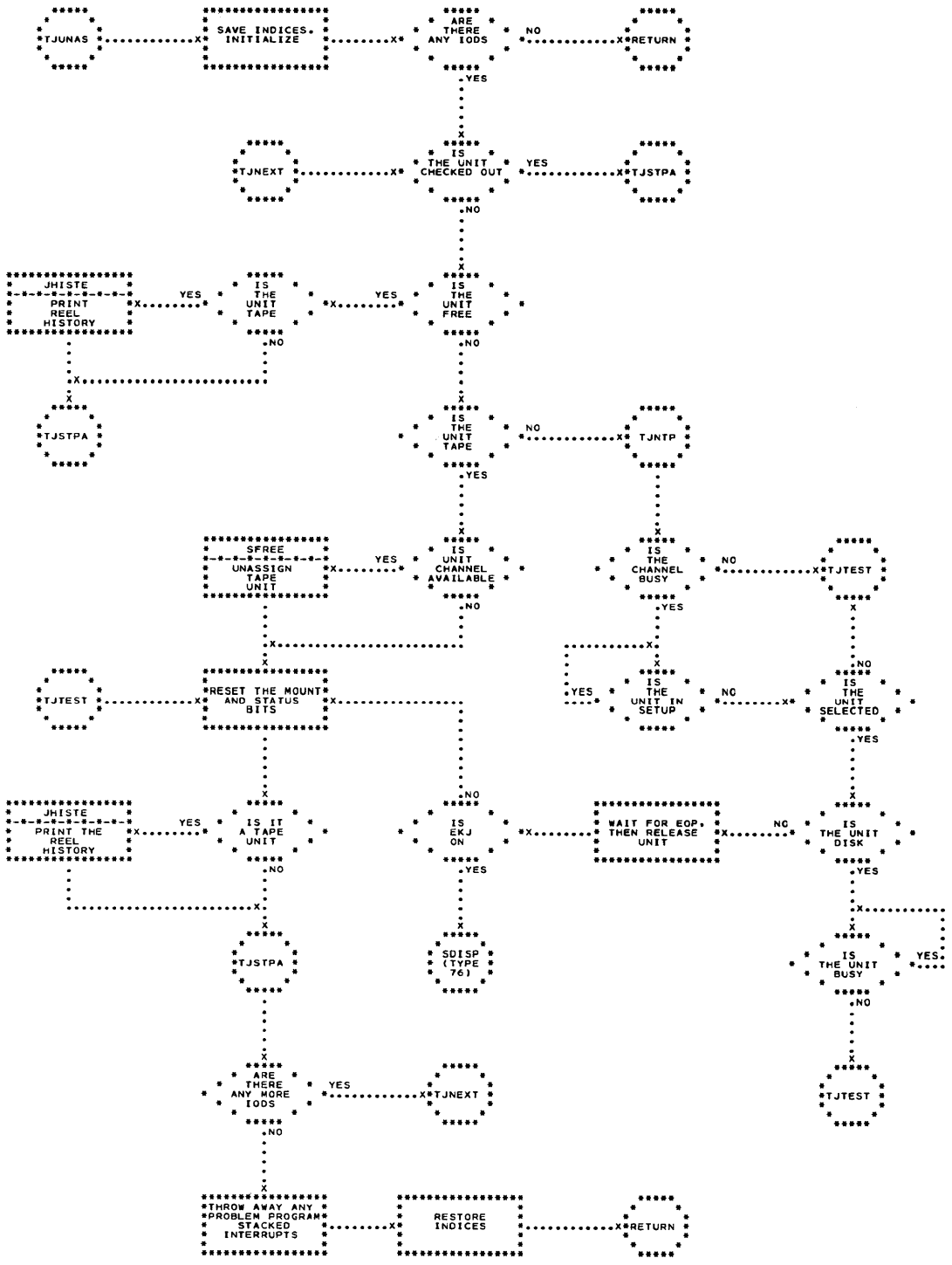


Figure 42. Unassign Routine

last activated the unit, a list of all the reels used by a unit with a count of its actuations and unit check errors, and indicates the reels final disposition. It is used by the unassign routine at EOJ time, and is entered once for each physical tape unit assigned to the PP. The routine uses the short message routine to perform the output via $SPR.

Conversion Routines

Four subroutines are used to convert data from one format to another. They operate disabled and are as follows:

SIQSA8 SIQSA6	IQS to BCD
SA6IQS SA8IQS	BCD to IQS
SCA6	Card image to BCD
SBRK8	Field break down

The IQS to BCD Conversion Routine: This routine is entered at SIQSA6 or SIQSA8 according to the BCD byte size required, with the following linkage:

```
LVI, 15, $+1.0
BD, SIQSAx
VF, A
CF, N
VF, B
Return
```

where

- A - is the FWA of the IQS string,
- N - is the number of characters to be converted,
- B - is the FWA where the BCD is to be stored.

The routine selects a 6 or 8 bit store instruction and performs the conversion by table lookup. The routine saves, uses, and restores index registers 13, 14, and 15.

The BCD to IQS Conversion Routine: This routine is entered at SA6IQS or SA8IQS according to the byte size to be converted, with the following linkage:

```
LVI, 15, $+1.0
BD, SAxIQS
VF, A
CF, N
VF, B
Return
```

where

- A - is the address of the BCD string,
- N - is the number of characters,
- B - is the location where the IQS string is to be stored.

The routine adjusts for the input byte size and performs the conversion by table lookup. It saves, uses, and restores index registers 13, 14, and 15.

The Card Image to BCD Routine: This routine is entered at SCA6 or SCA8 according to the BCD byte size desired, with the following linkage:

```
LVI, 15, $+1.0
BD, SCAx
VF, FWA(I)
CF, N
VF, A
```

where

- FWA(I) - is the location of the first column in the card image to be converted,
- N - is the number of columns to be converted,
- A - is the address at which the BCD string is to be stored.

The routine performs the conversion by table lookup, inserting a right parenthesis if an illegal punch combination is detected. Index registers 1 through 6 are used and not restored.

The Breakdown Routine: The function of the breakdown routine is to place each field of a string of BCD characters into a separate word. A field is defined as bounded on the left by a comma or the beginning of the string and on the right by a comma or the end of the string. The routine is written so that it can be assembled to take six bit characters or eight bit characters, according to the parameters QKSIZ (character size) and QSKIP (number of bits to skip at the end of a field word). These parameters are presently set at 6 and .16 respectively. A field is limited to eight characters.

The routine is entered at SBRK8 with the linkage:

```
LVI, 15, $+1.0
BD, SBRK8
VF, A
CF, N
VF, B
VF, M
Error return
Normal return
```

where

- A - is the starting location of the string,
- N - is the number of characters to be broken into fields,
- B - is the starting location where the fields are to be stored,
- M - is the number of fields found by the routine.

The error return is used if a field is found containing more than eight non-blank characters. Blanks are

omitted in setting up a field, as are commas. The routine saves, uses, and restores index registers 1, 2, and 3.

The Short Message Routine

The short message routine provides MCP with a common 17 word buffer for output via $SPR. It operates disabled, and is entered with the linkage:

```
SIC, ZSPLP9
BD, ZSPLPR
, FWA(I)
, N.
Return
```

where

FWA(I) - is the starting location of the message intended for $SPR, and

N - is the number of words.

The short message routine moves the message to its buffer and uses $SPR via the disabled entry to the IF analyzer. The routine saves, uses, and restores index registers 0 and 1. Note that control is not returned when the short message routine is used by a program which is not a major package. $SPR will appear to the return routine to have been primed, and primed routines do not return to the requestor. (See dispatcher error control.)

Resume Load

The resume load major package controls the loading of binary decks prior to execution of programs under MCP. Its main function is to supply cards to the MCP loader, which it uses as a subroutine, and respond to exceptional returns from it. It also serves as the last phase of the Fortran BSS Loader. It is primed by phase 4 of job control, and may also be used by the PP.

The Resume Load Package

Resume load has one entry point, YRESLD (Figure 43). Two pseudo-ops are associated with this entry point, $RESLD and 74.32. The latter pseudo-op forces resume load to use $SCR as a source of cards. Otherwise, the SCORG bit, KSILO (see JC4) and SREADS determine the source of cards and the manner of loading. Other preset registers hold limits for the loader (YLLSAV), and the disk type-area, if any (YDFCS). Resume load runs enabled, in the RIO mode, and saves no index registers.

Exit is always made via $RET. If the current deck was successfully loaded, its return or branch address is placed both into SICBU and location 1.32 of the tentacle table (see $RET). If an error was encountered during loading, $EOJ or $ABEOJ is primed so that the program is not executed.

Resume load uses the following routines:

1. The JC4 print program - YPR - to print out diagnosing messages for loading errors.
2. The major package fetcher - YMPFCH - to call itself into memory (if necessary).
3. The MCP Loader - MLOADR - to load cards.
4. The Input program - $SCR - to get cards from the system input source.
5. The Disk fetch - $FETCH - to get card images from the disk.
6. The Prime and return - SPRIME, $RET - as noted above.

Resume load immediately enters YMPFCH to get itself into memory from disk.

Its first task is to determine whether it is to act as a postprocessor to the BSS loader as is the case if KSILO+2.0 says BSSbbbbb in BCD, and if the current job is in the GO phase (SCORG=0) of a COMPILGO (KSILO+1.19=1). If it is a BSS postprocessor, resume load obtains the loader limits from KSILO and sets them up to extend from the lower limit to the base address of blank common, to inhibit loading into blank common. If the BSS job is segmented, exit is made to MRNRM5. Otherwise, PP memory is set to plus (+) infinity, starting at location (8)41. to the base of the relocation tables. It then exits to MRNRM5 in the loader, in such a way as to persuade the loader that it should load the next (i.e., the first) Fortran subprogram.

If resume load is not a BSS postprocessor, it sets up loader limits from YLLSAV, the fetch type-area from YDFCS, and $13 with the read source. At this point the op code is checked for 74.32, or unconditional $SCR. If this is the initial entry for the current deck (YLLSAV.25=1) PP memory is cleared from location 41.(8) TO SBAPP, or the base address of the PP I-O location table. The 1st card is already in YBCBU, if this is a GO phase, otherwise, it must be read from the source indicated by $13.

At this point we have completed all of the initialization in resume load. The routine basically consists of a loop between YRLCR and YLBRA.32: get a card, load a card, etc.. If the loader gives branch return, the loop is broken, and return is made to the PP provided no input program UK has occurred (REJJOB.60=0).

The remaining code exists to handle exceptional returns; $SCR end return, $FETCH end or error return, and loader error return. All are errors and cause a diagnostic message to be printed via YPR, with the exception of end return from $FETCH for a BSS job, which is handled specially. After printing the message, resume load primes $ABEOJ or $EOJ, depending on the initial entry bit. YEOJS and SPINCL

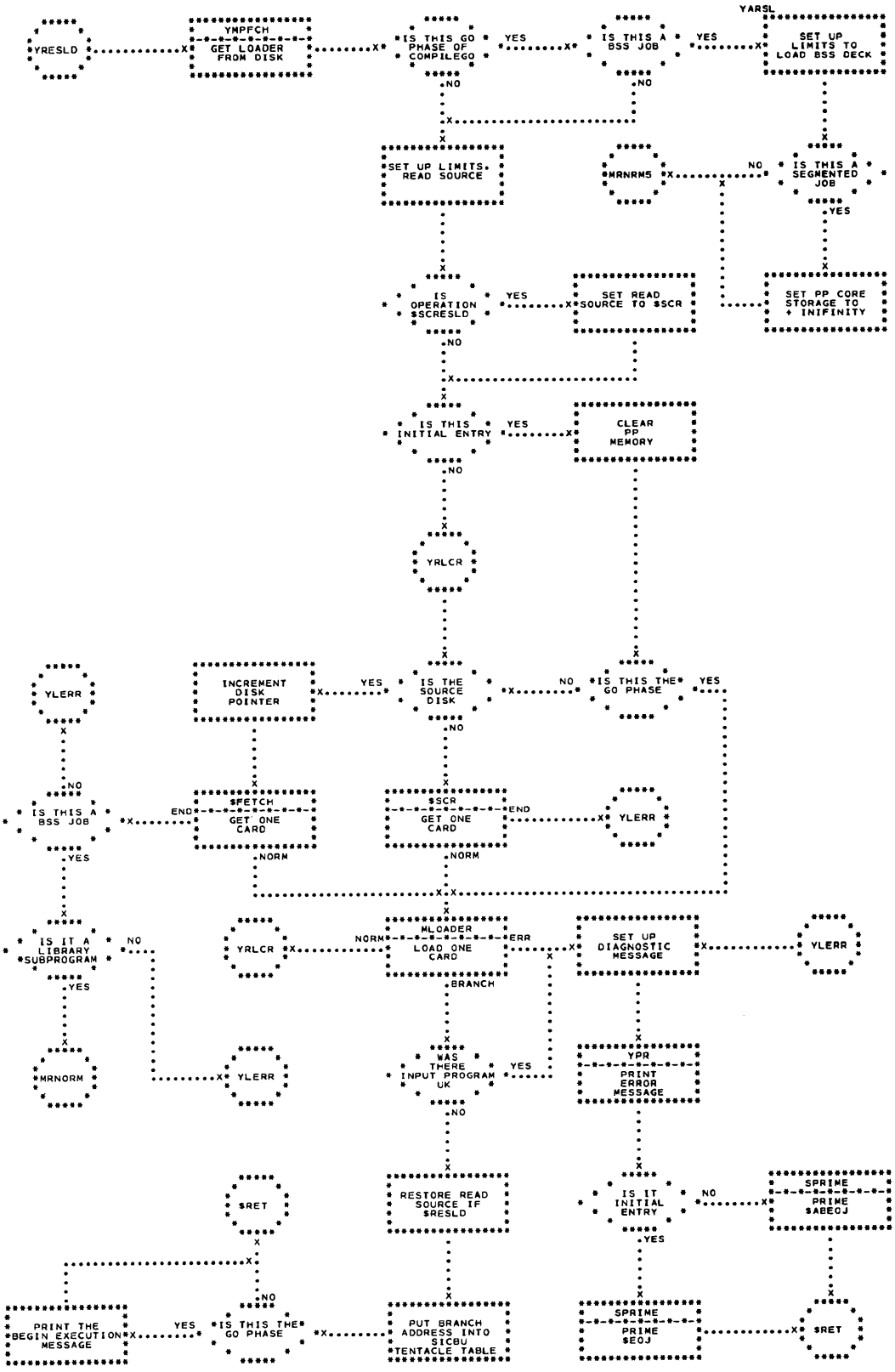


Figure 43. Resume Load Package

(see JC4) are set to indicate the job has been completed.

Tables and Flags Used by Resume Load: Below is a list of the various tables and flags used by resume load and their definitions. The first four items are all preset by phase 4 of job control, prior to priming $RESLD.

SREADS - the read source control is an XW whose flag bit is zero if the source is $SCR. If the flag is one, the source is disk and the value field contains the relative FWA in the disk type-area. Further, the count field, for compiled PP, contains the number of cards left on the current arc of disk. This XW is loaded into $13 by resume load. When the aforementioned count reaches zero, the value field is stepped 2 words and the count is reset to 34, the number of cards on one arc of disk.

YLLSAV - contains loader limits, i.e., those limits outside which the loader must not load. In addition, it contains the initial entry bit (.25), the T-card indicator (YTBIT, see JC4), and bit .57. This word is transmitted directly into the loader calling sequence by resume load.

YDFCS - contains the A6 type-area name to be loaded from, if disk is the source for cards.

SCORG - a bit in SCOMRG, which equals 1 when the current deck is a compiler and 0 when it is a go phase.

REJJOB - Bit 60 of this word is set by the input program when an uncorrectable unit check occurs during the reading of a job.

YBCBU - This 15-word buffer is used by both JC4 and resume load to read cards into.

KSILO - The processor communication region. The Fortran BSS loader and resume load define words 8.0-10.0 of this region jointly (see description of Processor Communication Region).

lower limit of PP	8.0	(BU, 18)
upper limit of PP	8.32	(BU, 18)
base address of blank common	9.0	(BU, 18)
address of relocation tables	9.32	(BU, 18)
branch address	10.0	(BU, 19)

MRCDCT - loader card counter for BSS jobs. It is set to 1 by resume load upon end return from $FETCH.

The MCP Loader

The MCP loader, MLOADR, is a binary, correction, dump, and patch card loader. It is used as a subroutine of the Resume Load pseudo-op, $RESLD.

$RESLD fetches the cards to be loaded and enters MLOADR, one card at a time, by the following linkage:

```
LVI, $15, $+1.0
B, MLOADR
, FWA
, L
, U
(Error return)
VF, D
(Branch return)
VF, A
(Normal return)
```

where

FWA is the first word (18-bit) address of the card to be loaded.

L is the lower limit of the PP. If XF=1, this is a new job and the loader must initialize.

U is the upper limit (protected) of the PP. If XF=1, the sequence counter is reset to contain the information in column 3 of the first binary card encountered.

Error return is a half-word location to which the loader returns when it cannot correctly load the card.

Half-word error code, D contains the sequence number, in bits 0-11, of the binary card in error or zero for an octal card. Bits 13-17 contain the following error codes:

1 -- Checksum error
2 -- Sequence or ID error
3 -- Illegal card type
4 -- Illegal non-BSS function
5 -- Out of bounds
6 -- C or P card incorrectly punched, e.g., hex character is not a 0-9 or A-F.
7 -- Blank location in first octal card.
8 -- First binary card not an origin card.

Branch return is a half-word location to which the loader returns when it encounters a branch card, or when the card count equals zero and there are no more relocation tables in a BSS job.

Half-word branch address is given unless the branch address field of the branch card is zero. In that case, A is the origin of the first origin card, or in a BSS job the branch address is taken from the communication region.

The loader does not save or restore index registers. A blank checksum means that the checksum is not to be used. The sequence number on relocatable cards is not checked.

BSS Jobs: For BSS jobs the MCP loader uses the communication region, KSILO, which has been set up by BSS. KSILO+2. contains the A6 name "BSSbbbbb" which is used by the loader to decide if the current job is a BSS job. KSILO+9.32 contains the address of the relocation tables which are used by the loader to

load BSS jobs. KSILO+10.0 contains the branch address which will be inserted in the resume load loader calling sequence when loading is finished, and the rest of the PP memory, (i.e., blank common) is set to plus (+) infinity for non-segmented jobs.

The format of the relocation tables is shown in Figure 44.

The loader (MLOADR, Figure 45) checks bit number 25 of the lower limit. If this bit is 1, MLOADR turns it off, initializes its counters, and sets up its internal boundary limits for the new job. A check is made on bit 25 of the upper limit to see if a T card has been saved for this job. If one has, MTCRDI is turned on and the sequence number of the first binary card is used as the base sequence number. Column 1 is tested to determine the card type, and control is transferred to the appropriate routine. These routines return to MNORMR to make a normal return, or to MERR3 to set up an appropriate error message and make an error return. A branch return is made when the validity of the branch is verified.

Loader Card Classes: The loader handles 18 classes of cards, identified by the punches in column 1:

Absolute origin	(7, 8, 9)
Absolute flow	(7, 9)
Absolute branch	(6, 7, 9)
Relocatable data	(6, 7, 8, 9)
Relocatable instruction	(5, 7, 9)
Fortran branch	(5, 6, 7, 8, 9)
Fortran program	(5, 6, 7, 9)
Common definition	(5, 7, 8, 9)
Correction	(12, 3)
Patch	(11, 7)
Dump	(12, 4)
T	(0, 3)
Super T	(0, 2, 3)
Loader adjustment (O)	(11, 6)
B	(12, 2)
Relocatable correction (K)	(11, 2)
Relocatable patch (A)	(12, 1)
Relocatable dump (Z)	(0, 9)

B, T, and Super T Cards: These cards are not loaded. For T and super T cards, indicator MTCRDI or MSTCRD is turned on. These indicators are later tested by the sequence test routine MSEQ.

Loader Adjustment Card: Loader adjustment cards (O in column 1) are handled by MLDADJ. This routine picks up the 8 Hollerith characters in columns 2-9, converts them to octal, and checks to see if they set an origin which is within the program's bounds. If the origin is within bounds, the loader's origin counter MLDCTR is set to the value on the O card. The branch address is also set to this value if a previous quantity has not been saved. These cards are ignored for BSS jobs.

Absolute Branch Card: The branch address on the card is checked to see if it is within bounds. If so, it is inserted into the calling sequence. If it is not, an error return is made to the calling sequence. If there is no branch address, a check is made to see if an initial origin has been saved, which will then become the branch address. If no origin has been saved, an error return is made. This routine gives control to the branch return in the loader calling sequence.

Common Definition Card, Fortran Branch Card, and Fortran Program Card: These card types use the routine MFPCRD (Figure 46). A check is made on MFPCI to determine whether or not a Fortran program card had previously been handled. If so, the card is counted in the routine MRNRM1. If the card count for this BSS job is not zero, normal return is made to the loader's calling sequence. If the card count equals zero, a test is made on the count field of the second word of the current relocation table to determine how many transfer vectors are to be transmitted to problem program storage. If SRNXAD is zero, the branch address is picked up from KSILO and stored in the branch address slot of the calling sequence and branch return is made. If SRNXAD is not zero, $RESLD is set up to fetch the next subprogram and normal return is made.

If no Fortran program card has been received, a test is made to see if this is a BSS job. If not, error return is made. If this is a BSS job, MFPCI is set to one and the address of the first relocation table is picked up from KSILO. This table gives the origin to the loader and enables it to compute the width of the set of relocation bits for each half word. This width is 1, 4, or 4+i where i is the field size necessary to express the highest-numbered labeled common block. For non-BSS jobs, the width of the set is either 1 or 4. If the current subprogram is not a library routine, the card count is stored in MRCDCT. Control is passed to the card count routine and eventually normal return is given.

Absolute Origin, Absolute Flow, and Relocatable Binary Data Cards: These card types eventually are treated as one, and use MORG1 to load the information from the card.

The absolute flow card routine (MFLOW, Figure 46) checks to see if an origin has been saved. If not, error return is given. The next location table loaded into is picked up from MLDCTR and put in $1 for MORG1. $6 is set up for the boundary test made in MORG1 and set-up is done so that a partial word will be loaded. Control is then passed to the absolute origin card routine at the boundary test instructions.

The relocatable binary data routine, MRBDCD, picks up a 19-bit loading base from column 10 and tests to see if it is program data or common data. If it is data to be loaded into named common and this is a non-BSS job, error return is made. If it is named common data, the common origin is picked up from the relocation table for this subprogram. Otherwise, the current location is picked up from MLDCTR. The relative origin is added to it and the total is stored in $1 as an absolute origin for MORG1. The origin saved indicator, MLL.3, is set to one, MTCRDI is set so that no sequence check is made, and MRBCDI is set so that the appropriate return will be made from the absolute origin card routine. Control is passed to the absolute origin routine at the origin checking point, MLD8.

The absolute origin card routine sets up the origin punched in columns 6 and 7 in $1 and tests for an out of bounds origin, in which case error return is made. The number of full words and the size of a partial word to load are set up. A test is made to see if the card will be loaded within bounds. If not, error return is made. If a partial word is to be loaded, the number of bits involved is set up in the field length and address fields of instruction. MLD1 and the succeeding instruction and the partial word indicator, MHFWDI, are set to 1. The sequence number and checksum are tested. The appropriate skipping or zeroing is done and the contents of the card are loaded one 64-bit word at a time. The partial word, if any, is loaded next, and any skipping or zeroing is taken care of. If this was a relocatable binary data card, control is passed to MRNORM which adjusts the appropriate counters and counts this card. Otherwise the current instruction counter is stored in MLDCTR, MHIWD is rounded up to a 19-bit address, if necessary, and normal return is made.

Relocatable Instruction Cards: These cards are processed by the routine MRBICD (Figure 46). The checksum is computed and checked. The absolute origin is computed by adding the relative origin in columns 6 and 7 to the program origin. The upper address to be occupied by this card is computed by adding the number of bits to be loaded to the absolute origin of this card; the result is then checked to see if it is within bounds. The number of half words to be loaded is stored in the count field of $4; the number of bits left is stored for use as the field length of MRBCDA and MRBCD6 and as the address field of MRBCD6. The location of the relocation bits is computed. A half word instruction is picked up and its associated relocation bits are tested to determine what type of relocation, if any, is to be done. The relocation bits are defined as follows:

0 - No relocation (one bit used).
1000 - Relocate leftmost 18 bits as lower address.
1010 - Relocate rightmost 18 bits as lower address.
1001C - Relocate leftmost 18 bits as common C.
1011C - Relocate rightmost 18 bits as common C.

where C is a binary integer, encoded in i bits, indicating the number of the common. For blank common references, C=0. If a reference to a named common is made and this is a non-BSS job, error return is given.

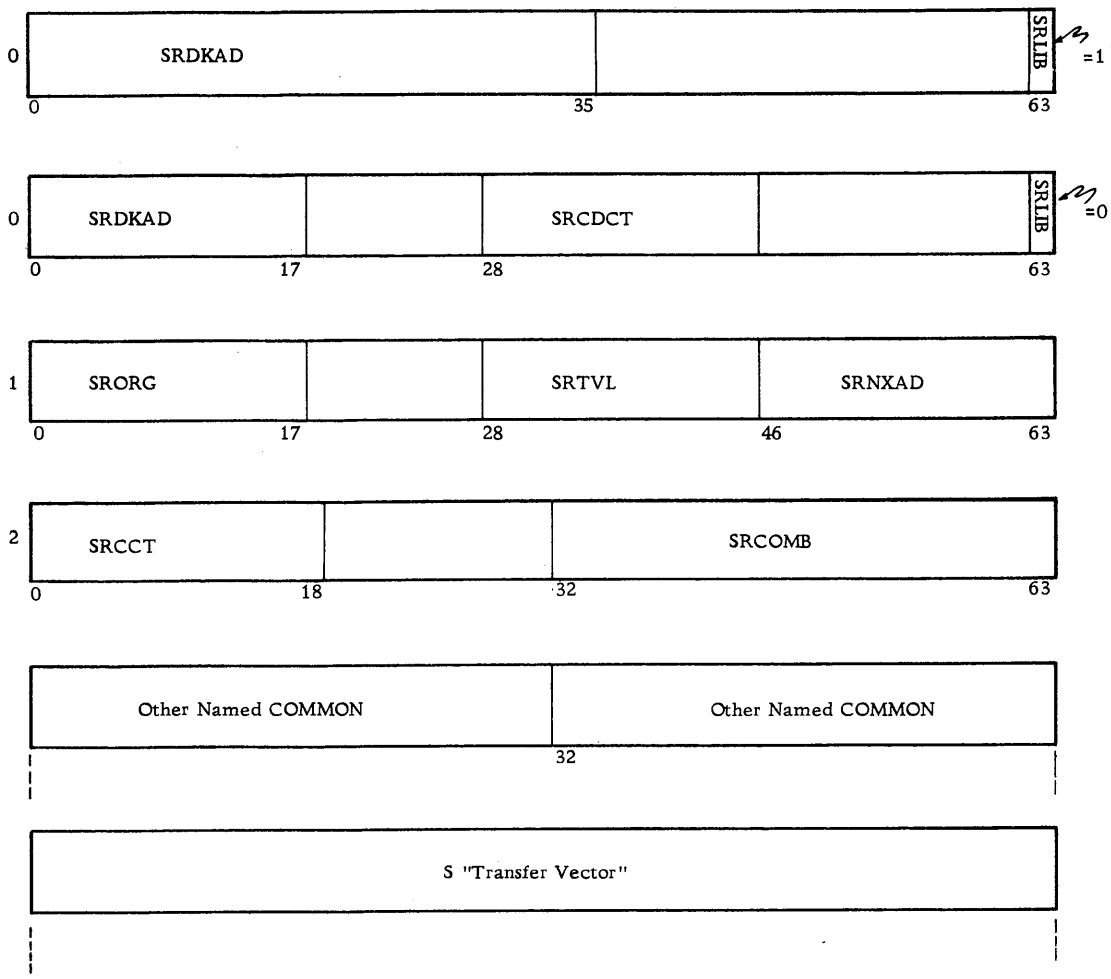
The card is loaded one half word at a time and any remaining bits are relocated and loaded. The loader's counters, MHIWD and MLDCTR, are updated. If this is a BSS job, the card counting routine is entered. Otherwise, normal return is made.

Absolute Octal Cards (C, P, and D): These cards are all processed by the routine MOCTLA (Figure 47), which computes and checks the origin and each half word to be loaded. The decimal point column is placed in the accumulator and compared for a decimal point (12, 3, 8 punch). For C cards, it is also checked for the optional one punch (12, 1, 3, 8, or 1 punches). If the one punch is in the origin field, no loading is done, but the location counter is stepped for each half word correction. If the one punch is in a half word correction, that correction is not loaded, but the counter is stepped. If the decimal point column does not contain a 12, 3, 8; 12, 1, 3, 8, or 1 punches a check is made, at MOCTL6, to see if this is the beginning of a continuation card with a blank origin field. If so, the contents of MLDTC are used as the origin for this card and the loading process continues with the first half word on the card. If not, then the contents of the card have been loaded and the loader's counters, MHIWD and MLDTC, are updated and normal return is given.

After checking the decimal field, the 7 Hollerith characters of the location field are converted to octal and checked to see if it is within bounds. If not, error return is made. The rest of the card is scanned as follows: (1) the location to be loaded into is checked to be sure it is within bounds, (2) the decimal point column is checked, (3) an octal-hex half word is converted, (4) the half word is loaded into core storage. This continues until a maximum of 4 half words per card have been loaded.

The P card routine, MACARD, also picks up the full or half word instruction at the location of the patch and moves it to the location specified by MHIWD before it loads the corrections. It inserts a branch to the patch area (and a NOP if the instruction being replaced is a full word instruction). After the contents of the patch are loaded, MNORMR returns control to MPCRND where a branch to the PP is stored in the patch area and MHIWD and MLDTC are updated.

For D cards, MZCARD performs the function of MACARD and uses MACARD to replace the instructions



LEGEND:

SRDKAD	SYN(BU, 36, 6), 0.0	If SRLIB bit = 1, meaning a library subprogram is to be loaded, then this is a 36-bit field containing a type-area name for a library subprogram. If SRLIB bit = 0, meaning TWS is to be loaded, then this is an 18-bit relative FWA in TWS.
SRCDCT	SYN(BU, 18), 0.28	Count of cards in subprogram when SRLIB = 0.
SRORG	SYN(BU, 18), 1.0	Origin of subprogram.
SRTVL	SYN(BU, 18), 1.28	Length of transfer vector.
SRNXAD	SYN(BU, 18), 1.46	Address of next relocation table. This equals zero when there are no more tables.
SRCCT	SYN(BU, 19) 2.0	Number of named COMMONS.
SRCOMB	SYN(BU, 32), 2.32, etc.	Base of named COMMON.

Figure 44. The Relocation Table Structure

with a branch or branch; NOP and uses MOCTL5 to load the dump parameters. In addition, it inserts the dump calling sequence in the patch area.

Relocatable Octal Cards (K, A, and Z): These cards are handled by the same routine as the Absolute Octal cards with the following additions: the relative origin is relocated with respect to the subprogram or named common specified to form the effective origin just prior to computing the half word corrections. Then, at the conclusion of each half word correction computation, it is relocated according to the relocation columns on the card. Also, because of the presence of the relocation columns, the maximum number of half words on a relocatable Octal card is three. (See the IBM 7030 Data Processing System Bulletin, Loader and BSS Processor, Form C28-6379).

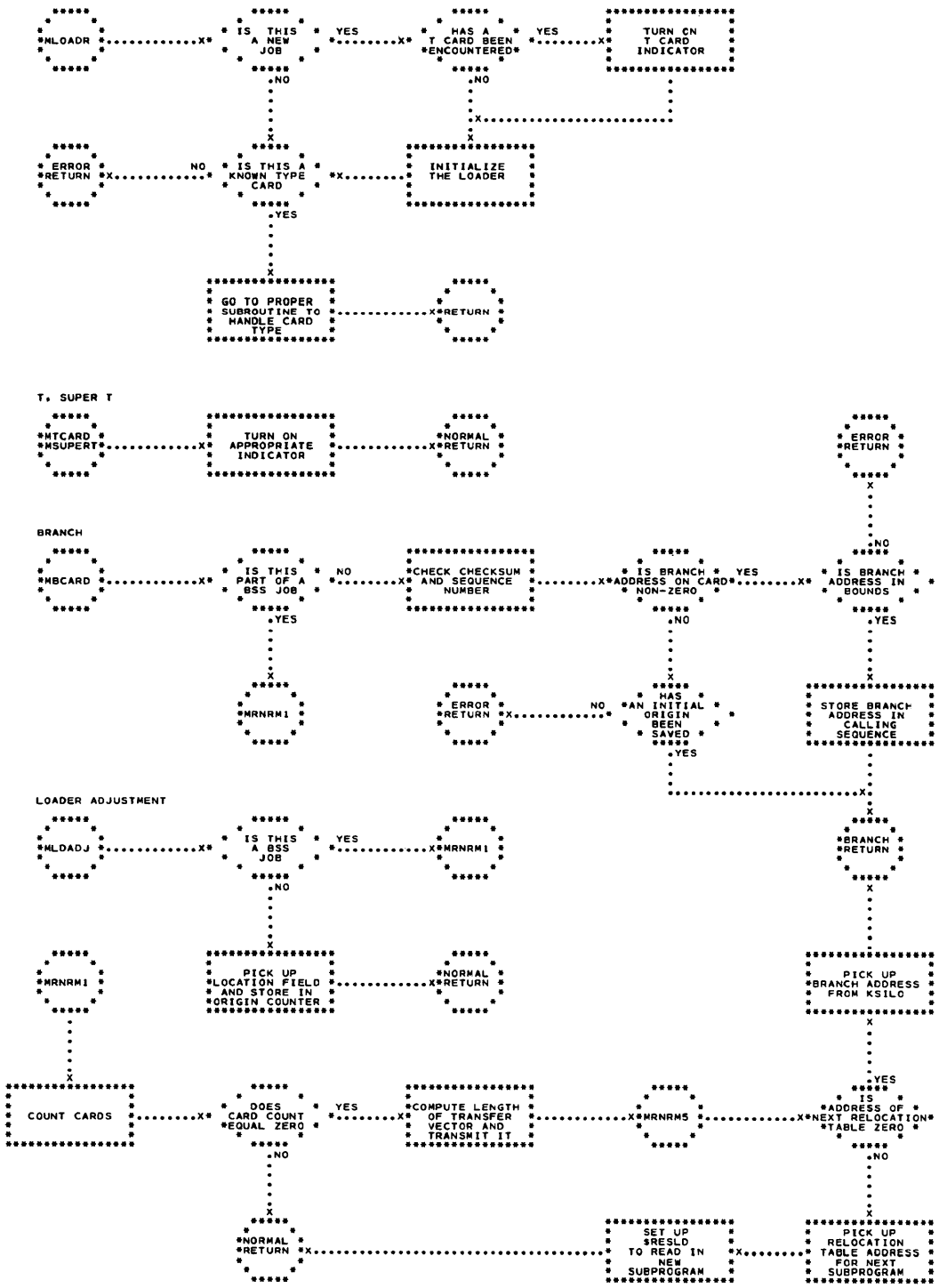


Figure 45. MCP Loader - Chart 1

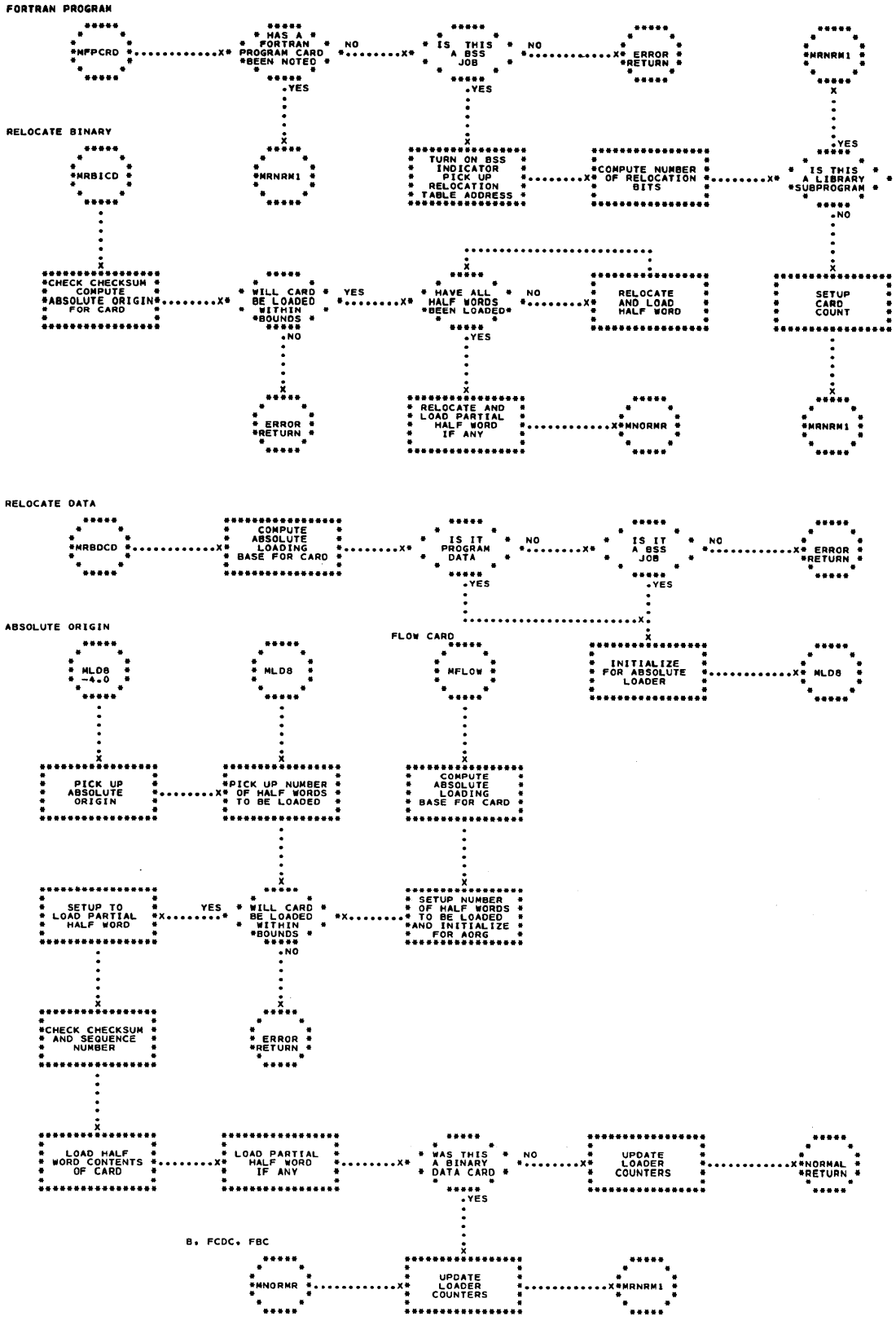


Figure 46. MCP Loader - Chart 2

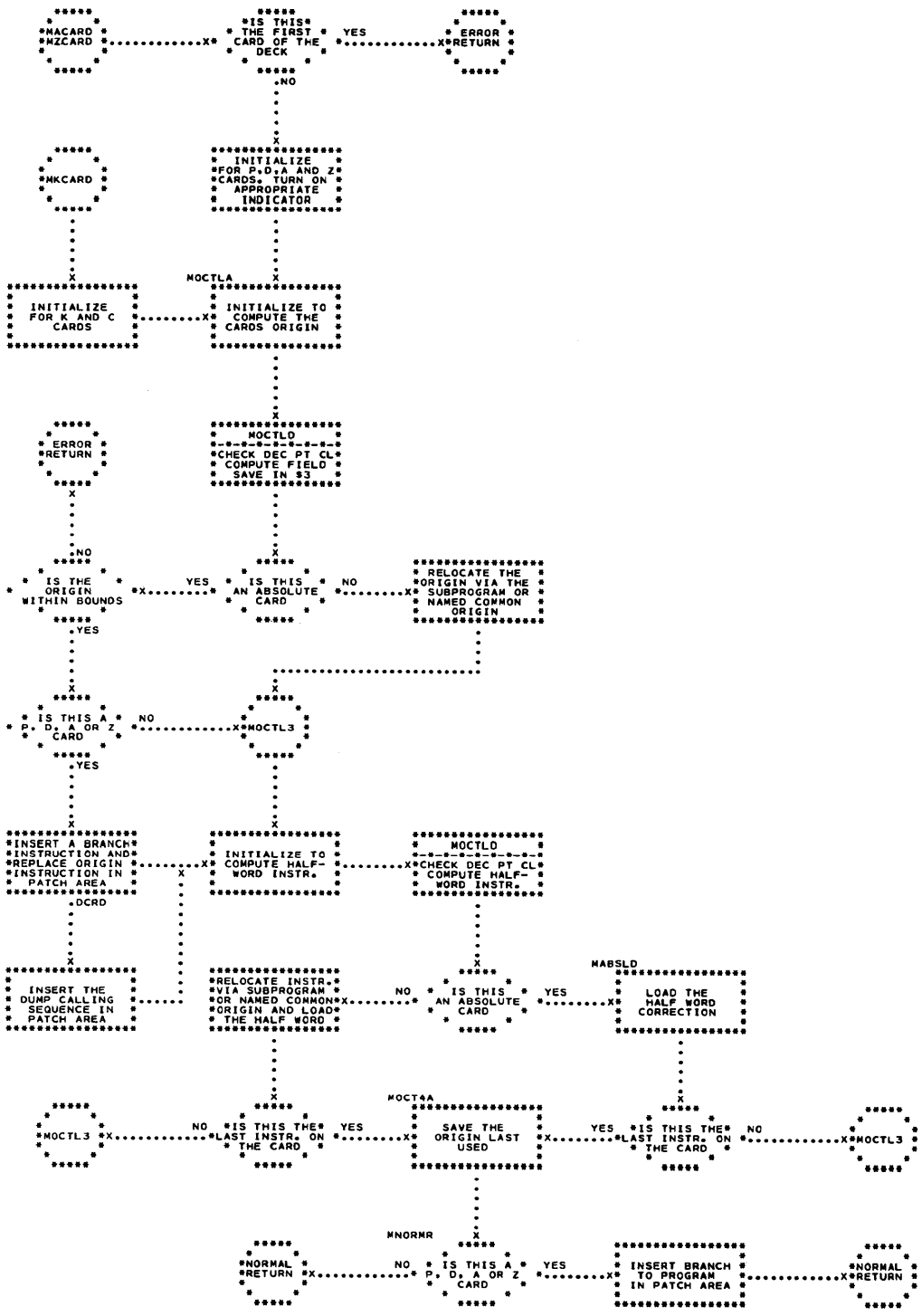


Figure 47. MCP Loader - Chart 3

Accounting Program Procedures (Logger)

The accounting program to be used must be supplied by the installation, and will occupy storage in excess of that allocated directly to MCP. Since MCP will have reference to the accounting program, and vice versa, appropriate addresses must be available upon compilation of both systems, or correction cards may be used when an MCP update is done.

In this section, the calling sequences for the accounting program entries will be given, as well as an explanation of the kinds of information that are available at the specific times used. The accounting program must have an initial state, since the system will start in the IPL phase. The job control sequence of events is critical, and will affect how the accounting program receives control.

```
B, D MCP
 , S LOG 1
 , Hollerith card FWA(I)
return
```

The entry through an S LOG 1 will provide a Hollerith card image address. The card will be either a JOB card or a COMD, EOF card. One of the initial settings of the accounting program may be to make use of the S LOG 1 JOB card entry to determine the time spent in the unoverlapped scanning of the original system input tape when the system is starting up in the overlapped mode. This procedure may be terminated when the first job is started by entry in S LOG 4 (described later), which will communicate to S LOG 1 the fact that overlapping is now in effect. The COMD, EOF card will be transmitted by the command package upon receipt of an EOF command card. This transmission will take place only if the EOF card is the last card in the card reader when in the online mode of operation, or the EOF card is the last file on a scan tape if in the off-line mode. This EOF will signify that the system is no longer overlapping the scanning and running of jobs, since it will come through job control 1 and S LOG 1.

```
B, D MCP
 , S LOG 2
return
```

The S LOG 2 entry indicates that a job is completed. No addresses are provided since the accounting program should have a record, from the last S LOG 4 entry, of the job name of the job just completed. There exists, for convenience of accounting purposes, a special program to assign a card punch or printer unit to the accounting program for use between jobs. The IOD tables will be set up at IPL time by having inserted an appropriate IOD card in the IPL tape for the desired unit. The calling sequence to the special assignment routine is:

```
SIC, Z COM 90EJ
  B, Z ASN 01EJ
 VF, IOD NUMBER
 return
```

The S LOG 2 function might be to calculate the time for the job, and then punch this information into a card along with the job name and other data, such as number of units used, etc. . It cannot be assumed that any of PP storage or associated IOD tables will contain valid information at this time.

```
B, D MCP
 , S LOG 4
 , Hollerith card FWA(I)
 , A8 FWA(J)
return
```

The S LOG 4 entry to the accounting program is used by the job control 4 at the beginning of a job or by commands to indicate an EOF condition arising when an EOF card appears as the last file on a system read tape or in the card reader as the last card in the BYPASS mode. The Hollerith and A8 FWA's make the contents of the card available if the accounting program wants to note the job name or determine whether the system is now idle.

It is the responsibility of S LOG 4 to disassign the punch or printer at this time. When the system is initially started, the initializing program will have assigned the unit to the accounter. Since S LOG 2 is not entered before the first job is started, the disassignment must be done by S LOG 4 when job control 4 gives it control just before the first or new job is started. The S LOG 4 entry is made before assignments are made to the PP so that the unit will be free when assignments are made. The calling sequence for the special disassignment package is:

```
SIC, Z COM 90EJ
  B, Z DSN 01 EJ
 VF, IOD NUMBER
 return
```

The accounting program may also wish to make use of the $TIME pseudo-operation afforded by the system.

INITIAL PROGRAM LOAD (IPL)

Initial Program Load (IPL) is used to read MCP into storage at the beginning of a work period or after an emergency shut-down. IPL is in two parts: the IPL bootstrap and the initialization program. It performs the following functions:

1. Fulfills the requirements of a power-on initial program loading (IPL).
2. Constructs I-O status tables according to the configuration at IPL time.
3. Assigns absolute I-O units to symbolic MCP requests.

4. Mounts MCP tapes.
5. Loads MCP into core.
6. Writes PROSA on the disk.
7. Starts job control.

The Master IPL Tape

The master IPL tape is prepared by the master update program (UPDATE-30). The tape consists of two files: (1) a record containing the IPL bootstrap memory usage, (2) a series of 513-word records in one-to-one correspondence with arcs of PROSA (permanent read only storage area) on the disk. The order of the first four type areas in this file is always the same:

Type Area	Content
11A11DIC	PROSA dictionary
11B11IND	PROSA index
11C11IPL	IPL initialization program
11D11MCP	MCP

The first word on the tape is a control word to read in the rest of the IPL bootstrap record. The tape unit on which this tape is mounted must be on a unit dialed to zero.

The IPL Bootstrap

The IPL bootstrap (XIPLBS, Figure 48) performs the machine initialization necessary at IPL time, locates the channel with the master tape, and prepares it for subsequent reading. The bootstrap consists of the code from XIPLBS to XBSEND. The bootstrap reads as many records from the master tape as core storage permits, and then (X9A3A) moves the initialization program (the third type area) to its operating storage. Control is given to the initialization program at XIN+1. to move MCP (the fourth type area) to its operating storage. The disk is located to arc zero, and the input buffer written on the disk. A loop is set up between the read portion of the bootstrap (X9A) and the initialization program (X11) to read tape and write PROSA until the tape is exhausted. The IPL tape is then unloaded, and initialization begins in earnest (X13).

The Initialization Program

The initialization process is controlled by certain control cards, which define the I-O configuration, MCP's I-O requirements, and furnish necessary parameters.

Control Cards

Symbolic cards are used to simplify any changes which have to be made. These cards describe standard conditions which can be modified through the use of console switches at initializing time. The control cards are included with the initializing program, and if they are to be changed, they must be updated on the master tape. Each of the three types of control cards used must contain a B in column 1. They are:

1. I-O Configuration Definitions (IOCD)
2. MCP IOD's (IOD)
3. MCP Parameters (MCPP)

I-O Configuration Definitions: The IOCD cards define the distribution of I-O units at initializing time. Each card describes a channel. Channels which are not so defined are assumed to have no attached units. The IOCD cards, which do not have to be in order of channel number, have the following fixed fields:

Cols. 10-13 -- Card Type: contains the characters, IOCD.

Cols. 14-15 -- Channel Number: contains the decimal number of the channel.

Cols. 16-17 -- Equipment: must contain one of the following codes:

CN	Console
DK	Disk
PR	Printer
PU	Punch
TP	Tape
RD	Reader

Col. 18 -- Channel Status: contains a "U" punch if the specified channel is up, or a "D" punch if the specified channel is down.

Cols. 19-26 -- Unit Status: contains a "U" punch if the corresponding unit is up, or a "D" punch if the corresponding unit is down. Col. 19 refers to unit 0, col. 20 to unit 1 ... col. 26 refers to unit 7.

MCP IOD's: The IOD cards define the I-O requirements of MCP and perform the same function for MCP as they do for the problem program. The initializing program will assign these symbolic requests to absolute units.

MCP Parameters: This card is used to provide MCP with initial parameters. It consists of fixed fields as follows:

Cols. 10-13 -- Card Type: contains the characters MCPP.

Cols. 14-17 -- Mode: defines the mode in which MCP will ordinarily start operating. It must be one of the following:

OFFLINE	-	Off-line overlapped
ONLINE	-	On-line overlapped
BYPASS	-	Bypass (Unoverlapped)

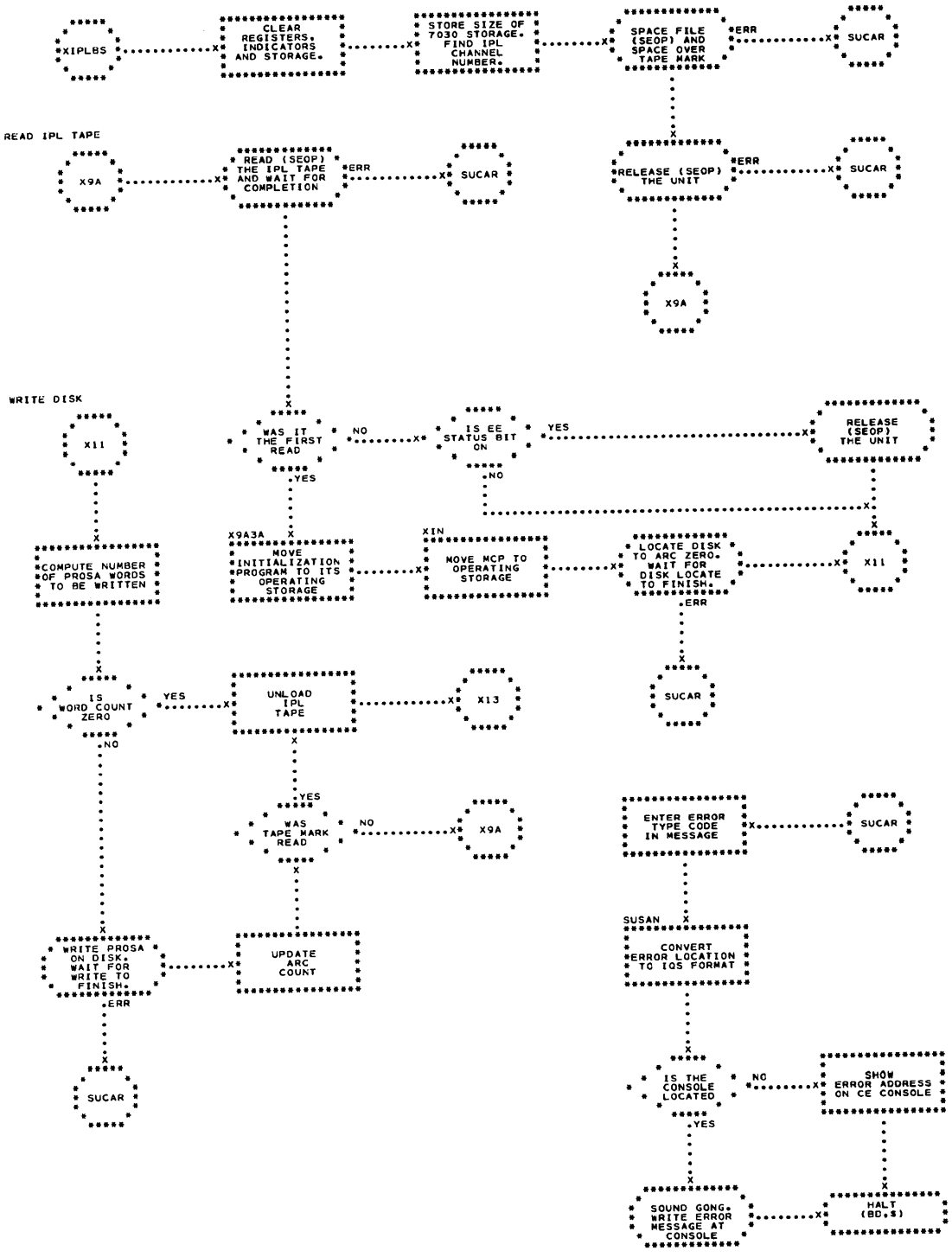


Figure 48. IPL Bootstrap

The I-O Status Table Set Up

At X13 (Figure 49), the initialization program processes the IOCD cards to form the Unit and Channel Status Tables (UST and CST). The IOCD cards need not be arranged in channel-number order, but there must be one card for each channel physically in the system whether that channel is available or not.

The tables are first formed in lower storage and then relocated (X19, Figure 50) to the area just below MCP (toward lower registers). This relocation allows the table size to vary from IPL to IPL. The tables which are set up by the assignment phase of the initializing program are just below the I-O tables. The lower boundary of the MCP table area is computed and saved as a system parameter (S MCP) in the communication region.

After processing the IOCD cards, the program types the date of the tape update (XDATE), and reads the console to determine if the operator desires any options. If binary key 29 is on, the program performs the requested status change and reads the console again, repeating until a status change is no longer requested.

MCP I-O Assignment

The assignment of absolute units from IOD cards will be done for MCP at IPL time through a modified decode-assign-move routine (SB22, Figure 50).

The assignments are made according to the I-O configuration as defined by the I-O tables, the type of IPL (normal or abnormal), and the operating mode (on or off line overlapped or unoverlapped). A summary of the action taken by this decode-assign-move sequence is as follows:

1. Normal IPL and Unoverlapped Mode: The card reader is assigned to MCP. A unit area table and file area table are set up for the card reader under MCP ownership. The output tape units are assigned to MCP and the necessary tables are constructed. Mounting messages are sent out to operator for scratch tapes to be placed on the assigned units. The unit and file area tables are formed for the input tapes, but they are not put under MCP ownership until a system command changes the operating status to overlapped. No mounting requests are made until the mode changes.

2. Normal IPL and On-Line Overlapped Mode: The input tape units are assigned to MCP and the necessary tables are set up. Mounting messages are printed out for tapes to be mounted on two units. These tapes will initially contain no jobs, but will be filled with problem programs read in from the card reader. Card reader and output tapes are assigned as they were in case 1.

3. Normal IPL and Off-Line Overlapped Mode: The input tape units are assigned as in cases 1 and 2. However, the mounting messages will be worded differently since the scan tape which contains the first job to be run must be mounted on the symbolic unit which will be first referenced by the input program. The other scan tape will be placed on the second input unit. Output tapes are assigned in the same manner as the two prior cases. The input card reader unit area and file area tables are formed but not put under MCP ownership until a system command changes the operating status to unoverlapped, on-line overlapped, or makes the unit the system input source.

4. Abnormal IPL and Off-Line Overlapped Mode: The abnormal IPL selection is made at the console and must be used when restarting with an input spool read tape which has been partly exhausted of its jobs, such as after a system failure or after an end-of-day shutdown. While in the on-line or off-line overlapped mode, the IPL must be made in the off-line overlapped mode because the old read tape must be rescanned. The assignments and the mountings will be the same as case 3 except the second input tape will be an unused scan tape which will be used later as a write tape. A system command must be entered before tape switch time to return to the on-line overlapped mode, if such mode is desired. The output tapes and the card reader are handled as they were in case 3.

When I-O assignment is complete, the program stores the base address of the I-O location table and the address of the next available arc on the disk in the MCP communication region, and moves the UAT and FAT to their proper locations.

Transfer to MCP

The program stores the number of jobs to be rejected if this is an abnormal IPL, and tests for a mode change request from the operator, (XMODE1, Figure 51). If none, the mode is taken from the MCP parameter card. The necessary bookkeeping is performed, and (X37) the command package and JC4 are primed. The MCP boundary register and interrupt address are stored, and the IPL messages printed if the suppress key is not on.

Final bookkeeping is done (X378), the disk located to arc zero and all tape units selected, and $RET issued. The system starts when the return routine empties the prime queue.

IPL Error Control

Any error detected by the IPL initialization or bootstrap programs results in a BD, $, with an accompanying error message typed if possible. The error control routine, SUCAR, will type an error message if it can identify the error and can access the console.

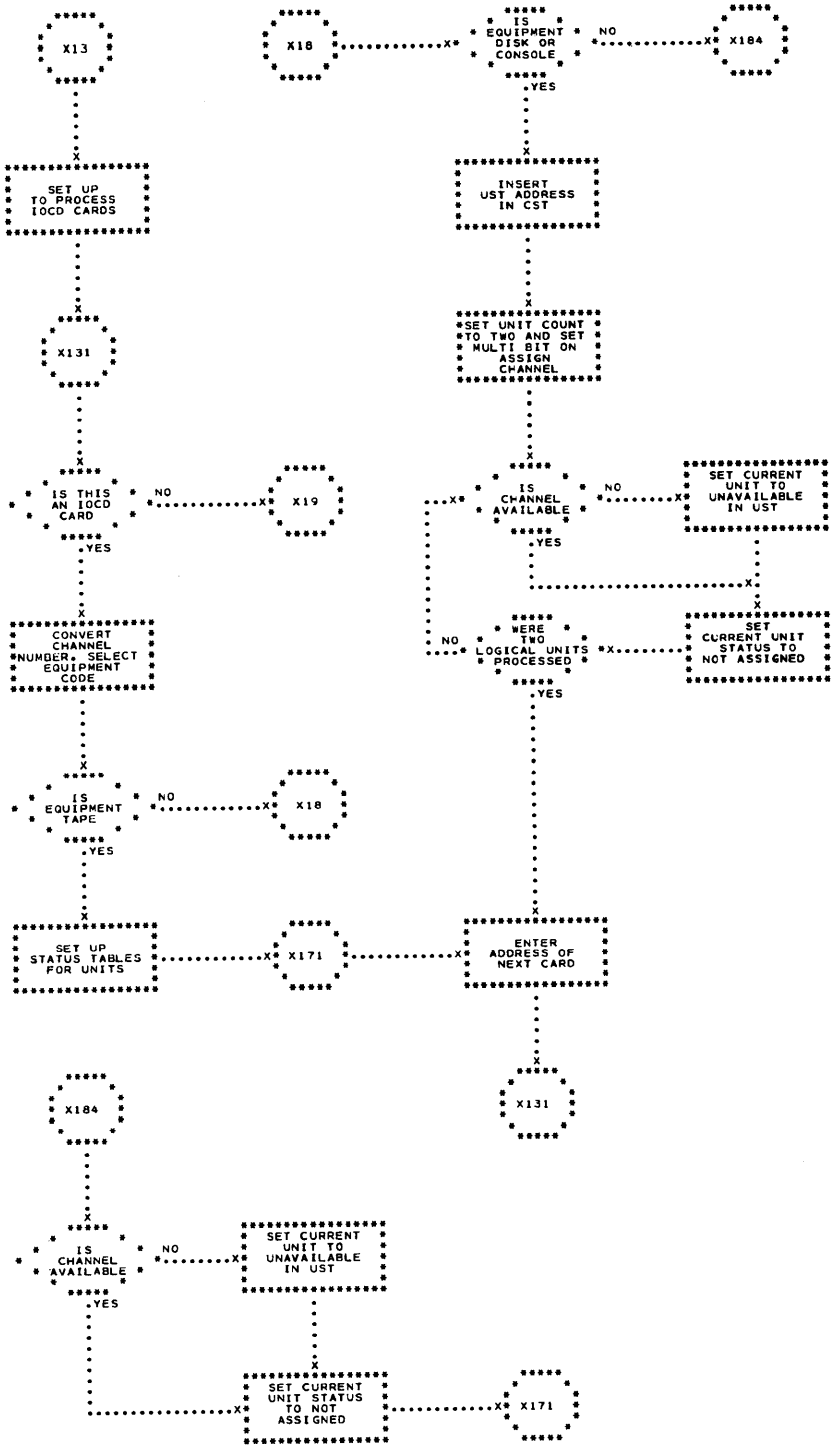


Figure 49. Initialization Program - Chart 1

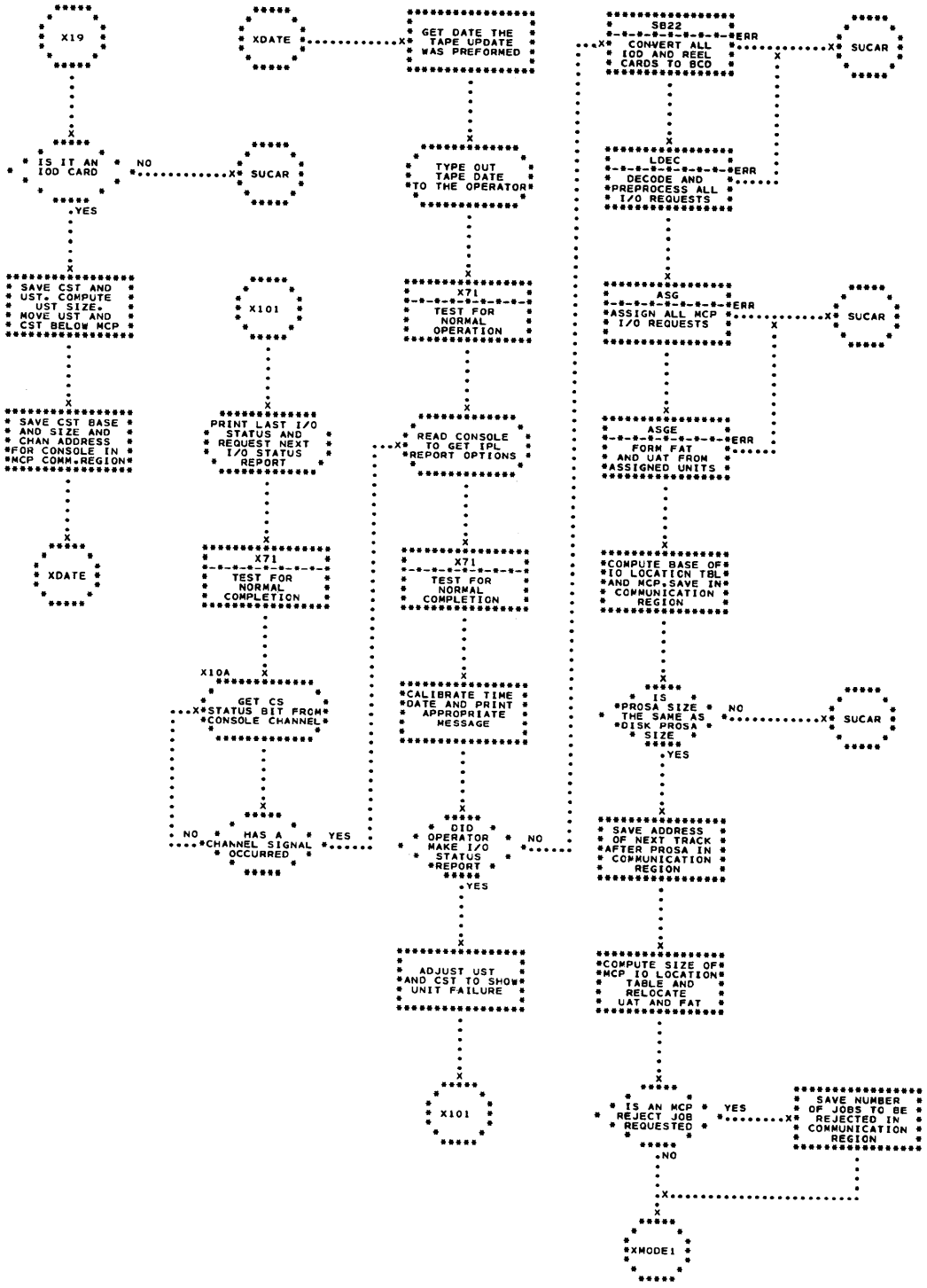


Figure 50. Initialization Program - Chart 2

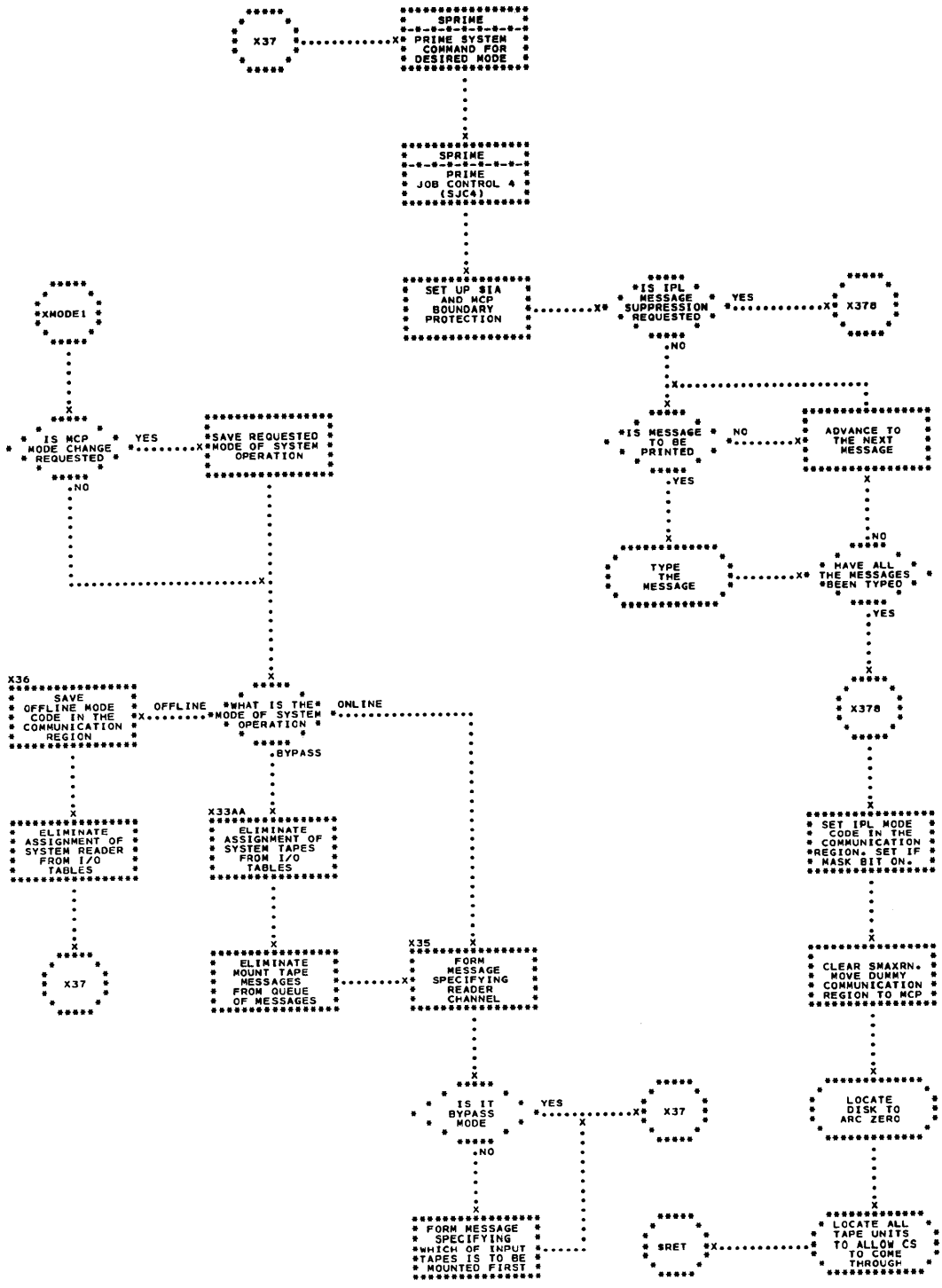


Figure 51. Initialization Program - Chart 3

RESTART

The Restart program is used to reinitialize the system without recourse to the master IPL tape once the IPL program has written the system on the disk in the PROSA area and has placed MCP in core storage. Restart is composed of two parts: the Restart bootstrap in main leg of MCP and the Restart type area (33RESTRT) on the disk. Some of the actions of Restart are controlled by the binary keys but Restart can perform all the following functions:

1. Give a console message if 33RESTRT is brought in successfully.
2. Give a console message stating the name of the current program, the system's operating mode, and the instruction counter when Restart was initiated.
3. Write out the commentator buffer on the typewriter.
4. Write the output program's print and punch buffers on the output tape.
5. Give a dump of core storage.
6. Reinitialize the system by bringing in the IPL bootstrap program from the disk and giving it control (Disk IPL).

Restart Bootstrap

The Restart bootstrap (YSICADD, Figure 52) first checks the instruction counter to determine if there is a call for Restart or a violation of protected storage area, octal location 41. If the instruction counter indicates a branch disabled to location 41, a type 16 error is given. Otherwise, Restart bootstrap saves the lower registers, and searches the dictionary in core storage for 33RESTRT to get the arc and word for the type area. If 33RESTRT is not found in the dictionary, Restart bootstrap halts with a B,$. If 33RESTRT is found, the track containing the Restart program is read from the disk into octal location 1000. Control then passes to the Restart program at the relative location of YARPBRP. This location is determined by the arc-word address of the Restart type area.

Internal Restart

The MCP program allows the option of initiating a Restart through programmed instructions in MCP rather than operator intervention. The instructions:

```
SIC, YIPLREQ
BD, YIPLREQ
```

pass control to the YIPLREQ routine in Restart bootstrap, which prints the console message:

```
$ SYSTEM ERROR. MCP WILL RE-IPL.
```

The gong is then sounded and a branch is made to YSICADD, the beginning of Restart bootstrap.

The Restart Program

At YARPBRP (Figure 52), the Restart program is transmitted into its place starting at octal location 1000 and is given control at YARPBF (Figure 53). In the YARPBF routine, two messages are given; one indicates Restart was brought in from the disk successfully, and the other gives the job name, system's mode, and IC. The console is then read to get the Restart options. If binary key 63 is set, the commentator buffer is written on the typewriter. At VJSYNX, binary key 62 is tested and if set, the output program's print and punch buffers are written on the output tape. The status of the buffers is checked in the VJTEST routine and the buffers are written out, starting at VJTAPE (Figure 54). Control eventually goes to YGOONX (Figure 53), a routine which reads the console; if binary key 61 is set, control goes to the YDMLOOP routine. At YDMLOOP, if a console channel signal is given in response to the dump request message, the program dumps on the selected output device, the locations specified in the numeric switches.

At the end of each dump, the console is checked to see if binary key 61 is still set; if so, a return is made to repeat the dump sequence. When binary key 61 is off, control goes to YSKPDMP where binary key 60 is tested.

If binary key 60 is set, the VCHKNO bit is set to indicate to the Disk IPL program that a check sum of PROSA should not be done. Control then passes to VCLEAR (Figure 55), the beginning of the Disk IPL program and the last subprogram in Restart.

VCLEAR halts the Restart program and gives a message suggesting that the IPL console options be set up in the binary keys. Upon console channel signal, Restart continues and the check sum of PROSA is calculated if the VCHKNO bit is zero. If the check sum calculated does not agree with the one that Update 30 calculated and stored in Type Area))))))). on the disk, Restart gives a console message stating that the check sum is incorrect; this is followed by another message to IPL the tape, after which Restart halts at a B,$. If the check sum is correct or if it is not to be calculated, the IPL bootstrap program is brought in from the disk, the PROSA arc count is stored into XVDK, the disk is located to arc zero, the Disk IPL indicator (PG5) is set, and IPL bootstrap is given control to renew the system.

THE COMMAND PACKAGE

At any given time during the normal processing of jobs, the operator may want to alter the automatic

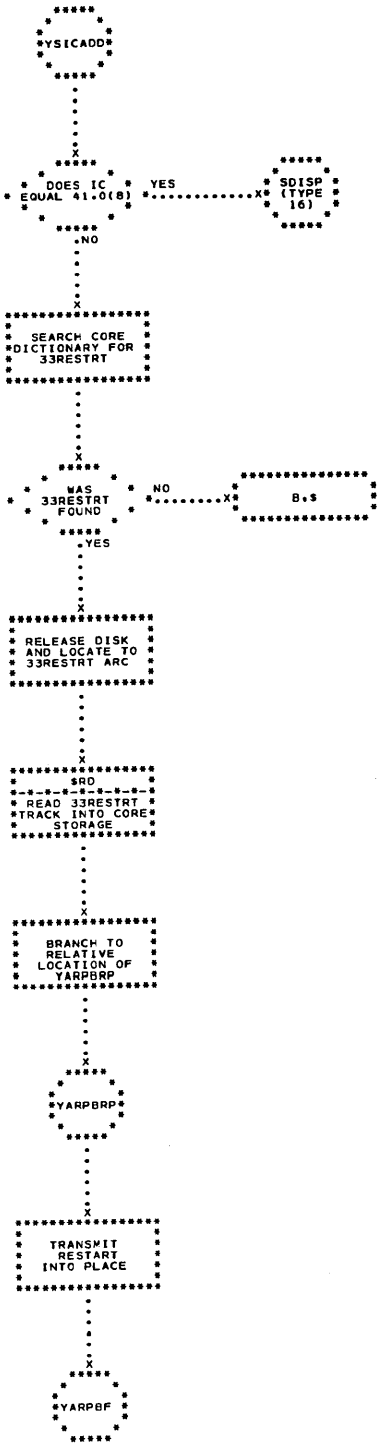


Figure 52. Restart - Chart 1 - Bootstrap

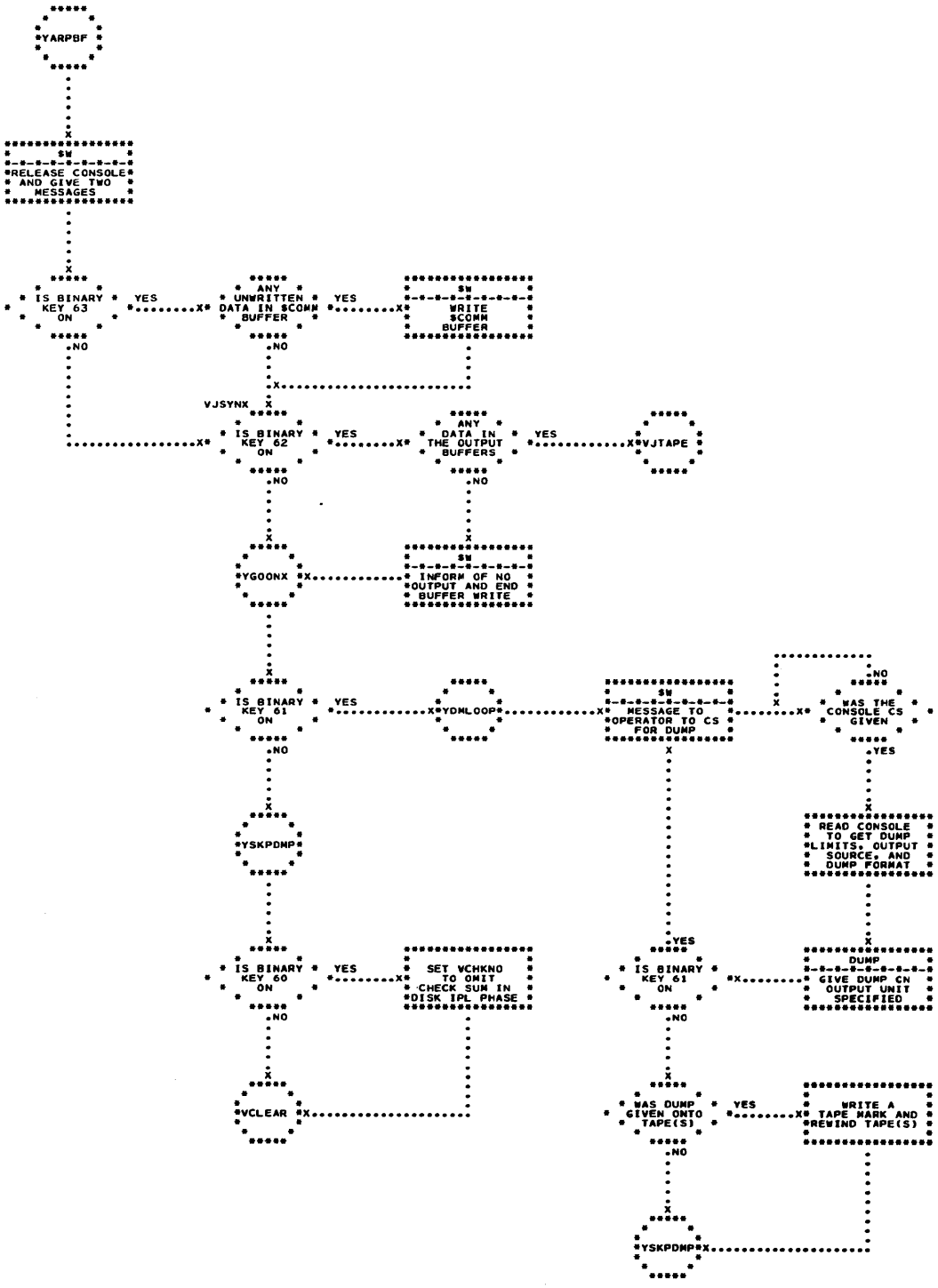


Figure 53. Restart - Chart 2 - Options

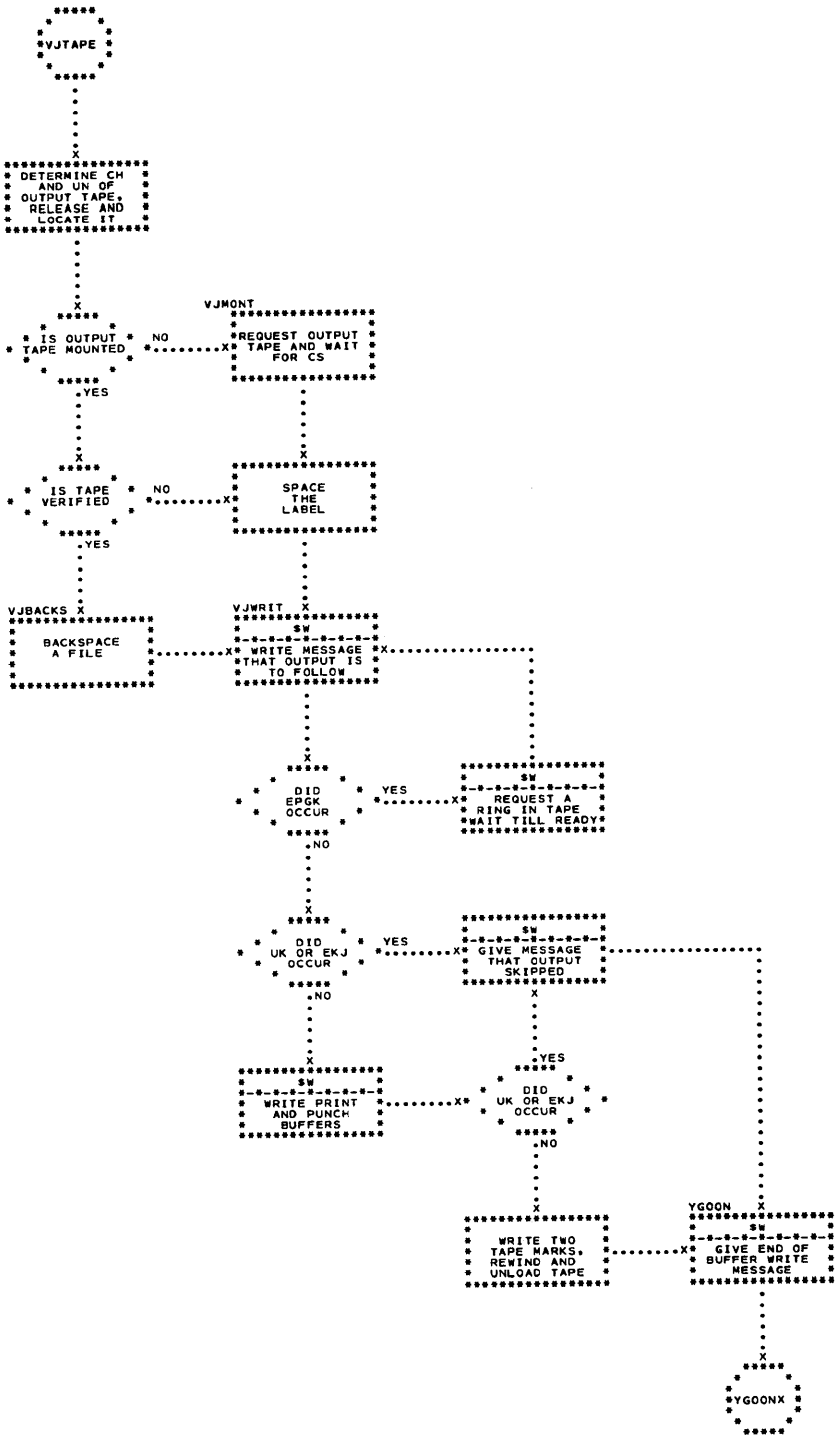


Figure 54. Restart - Chart 3 - Write Output Buffer

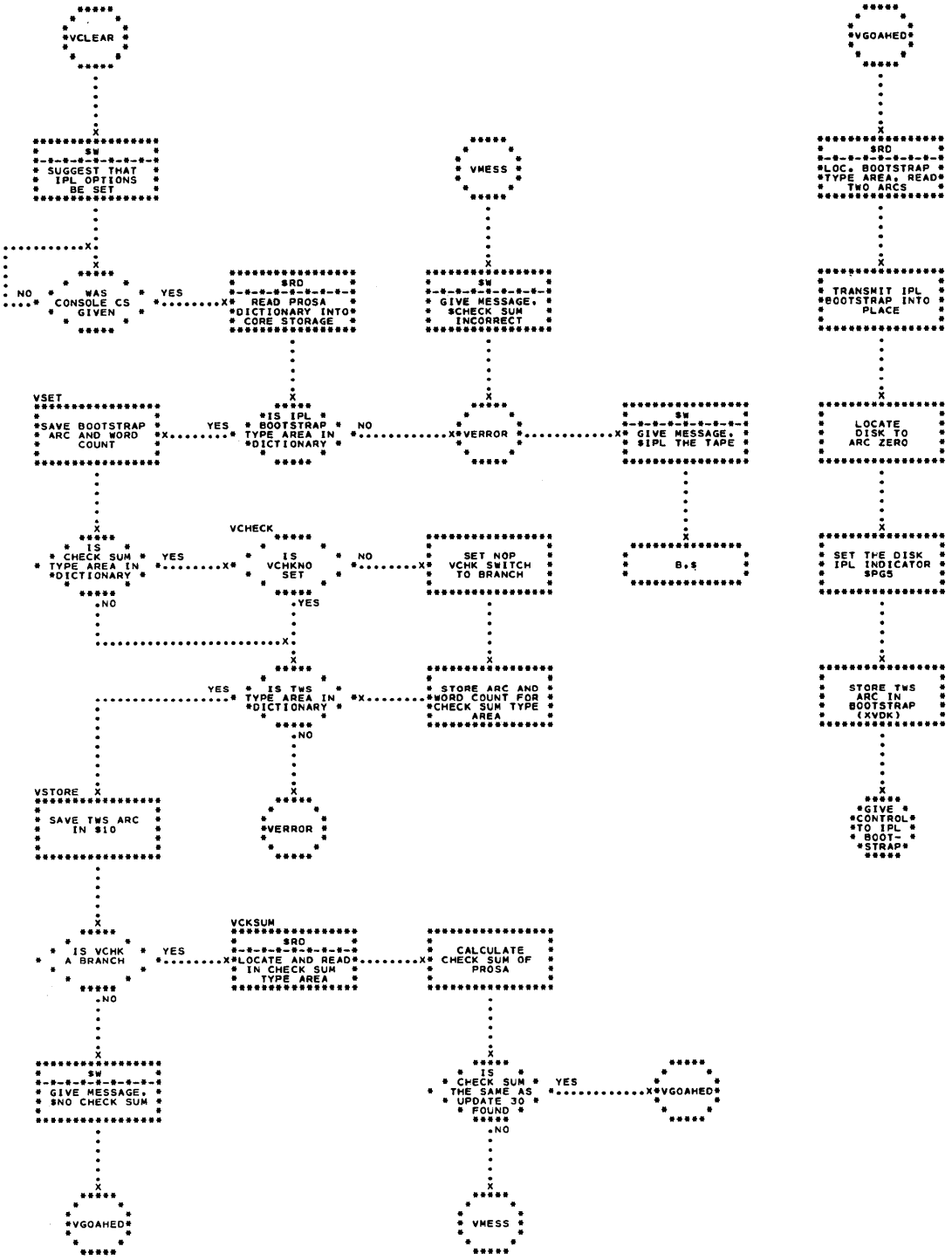


Figure 55. Restart - Chart 4 - Disk IPL

flow of work through MCP. The System Command Program interprets and acts upon these requests.

For some of these commands, the desired response may depend upon the operating mode of the system, e.g., overlapped or bypass. In order to avoid a conflict between intent and actual execution, the input source for some commands is restricted. The proper source is determined by the system mode and the nature of the command.

Command Mainstream

Before control can be passed to one of the individual command routines, certain general checking procedures must be performed. The command mainstream (JCOMD, Figure 56), decodes all commands into a standard format and breaks them into individual fields. It develops a matrix mask from the source and the operating mode of the system to test the legality of the command request. The mask is also used for internal control procedures within the routines. A normal exit routine is provided by the mainstream for all specific routines, and two error exits: ZEREX1, for use when the command is illegal; and ZEREX2, for use when a legal command is given in an illegal situation.

Sources

There are four possible entrance sources to the System Command Program. All use the calling linkage described under SCOMD pseudo-op. The most frequently used sources will be the operator's console, known as the operator source, and the input program card reader source. In this case, the linkage will be an indirect transmitting linkage between the operator and the command package.

Console: The operator may wish to enter a command via the console typewriter, providing this is a legitimate source for the command in the mode in which the system is operating.

When the CS is generated at the console, the interrupt is received and interpreted by the conceptor. The conceptor will issue a hardware read to the console; this releases the keyboard. The console reserved light will also be turned on. The conceptor then gives up control until the operator has finished entering his message and an EOP interrupt is generated by the END key. The console read portion of the conceptor gains control and examines the message for a DB or PP as the first two characters. If neither is there, control is passed to the signal return exit in the command console table of exits, the first location of which is JCFIX.

The signal return interrupt routine resets the SCNSSG indicators in the unit status tables, obtains a count of the number of characters in the message, primes system commands, and issues $RET. The eventual unstacking of the prime queue actuates the command. Since the console is reserved, the message may be left in the conceptor buffer PMCPBF with impunity.

Job Control: Commands entered via the input source will be discarded by job control unless they occur between jobs: i.e., after a job boundary and before the succeeding job card.

When job control detects a command card in phase 1 (JC1 source), it enters the command package directly with the command pseudo-op, SCOMD. Control is returned to JC1 when $RET is given by the command package.

When phase 4 of job control detects a command (JC4 source), it primes the command package (SCOMD), primes itself, and issues $RET. Thus, control will ultimately return to the beginning of JC4 after the command is activated.

SCOMD Pseudo-Op

The communication link between the separate sources and the command package depends mainly on the calling linkage; in the case of the operator, it depends on the interrupt mechanism and routines. The conversion routine and the prime mechanism are the primary system routines used.

The passage of control to the Command Program uses the following linkage:

```
B, $MCP    (or SIC,SPRIMR; BD,SPRIME)
, SCOMD
, S.
, L. (I)
, N. (J)
(Error return)
(Normal return)
```

in which:

S is the source of the command:

- 1.0 - Job Control 1
- 2.0 - Job Control 4
- 3.0 - Console
- 4.0 - Initializing Program

L. (I) is the bit address of the first IQS character (console) or of the broken-out field (Job Control) or of the BCD character string (Initializing Program).

N. (J) is the number of characters or fields.

(Error return) may be used only when the reject command is given by JC1.

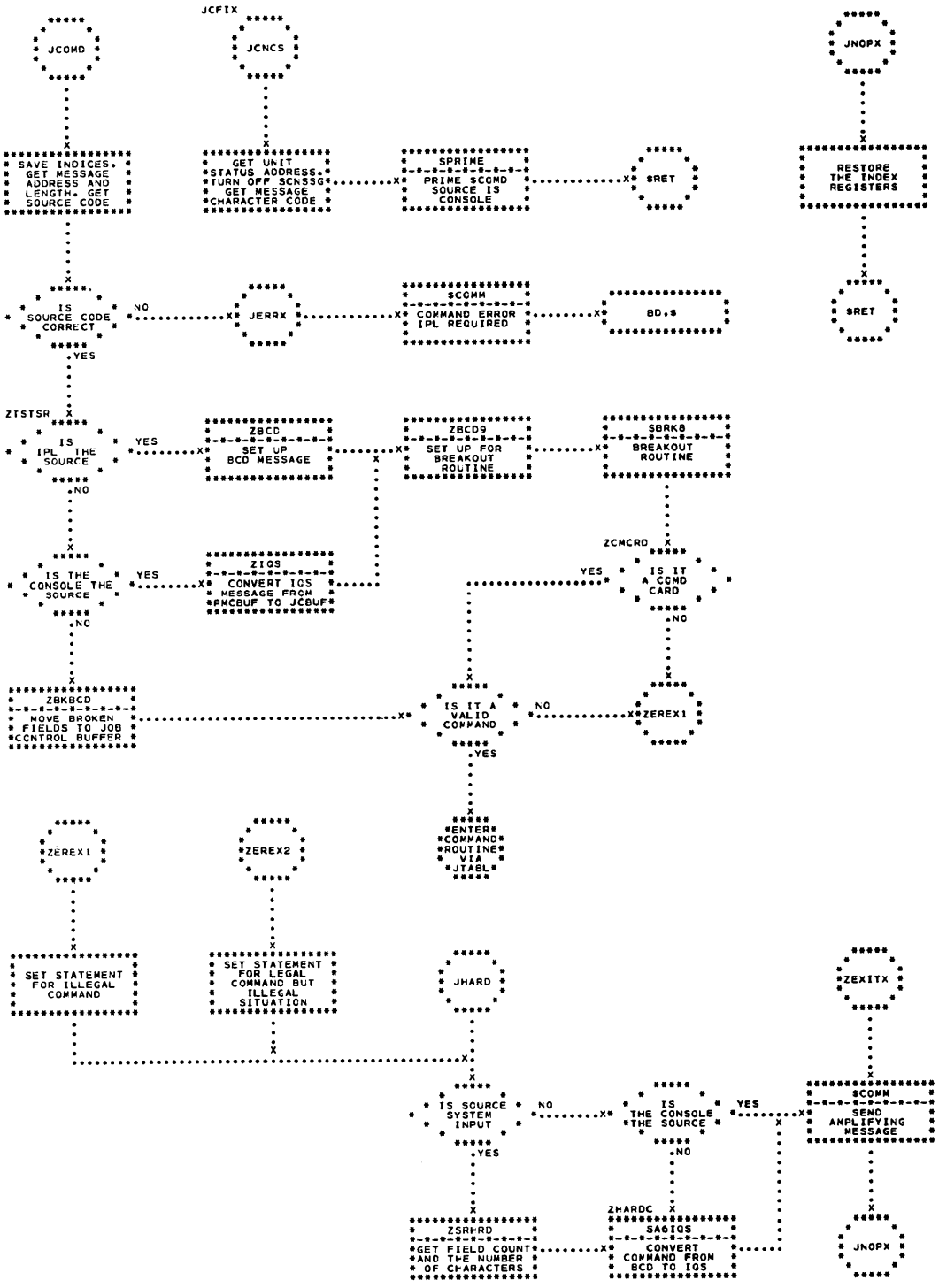


Figure 56. Command Mainstream

The format of the command message is essentially the same for all sources. Commands coming via cards must have a B punch in column 1 but the statement field, columns 10-72, contains the command statement as it is entered via the console:

COMD, command and parameters

Upon receipt of control, JCOMD (Figure 56) saves the index registers and then proceeds to construct the matrix mask from information in the calling sequence and the system mode bits, SYSMOD. Because the command message may be any one of three different formats depending on the source, a multiway branch must be made to specific calling sequence sets for conversion, breakout and/or moving routines. These sets of calling sequences will transform the message, if necessary, into BCD format, break it out, and eventually store it in an area JFLDB. The message in JFLDB is in eight-character broken-out fields. If the number of fields is less than two, the command request is in error, and the common error exit routine is entered with an explanatory message.

If there are two or more fields and the command is entered through a source other than the Job Control source, the first field will be tested for COMD at ZCMCRD. If rejected, exit will be made via the common error exit routine. After determining that a command message is acceptable, a table lookup is made to compare the second broken-out field, the command request op code, with a table of legal op codes for the commands. If the command request is not found in the legal list, the common error exit is entered, with an appropriate message. For all error messages, a hard copy is made on the console if the input source was not console. If an equal comparison was found, a branch is made to the requested individual command routine for further processing.

The common error exit routine, JHARD, will make a hard copy of the command message on the console typewriter, if the console was not the source. The routine uses the SA6IQS conversion routine to convert the original command message to IQS code. This, together with an amplifying error message is then sent to the console typewriter via the commentator. The normal exit routine, ZEXIT, is then used, which resets the console reserved light, if necessary, restores the index registers, and gives $RET. The general exit routine provides for handling normal exit messages as well as error exit messages by use of index registers.

The major system subroutines used by the command mainstream are the conversion routines from one format to another, and the breakout routine. The inputs can be in IQS, BCD broken out, or BCD. These must be transformed to broken out BCD fields.

The reserved light off pseudo-op is the only pseudo-op used, and the commentator is used for output. The hard copy maker converts from BCD to IQS for the commentator, using the SA6IQS routine.

Mode Control Commands

The operating mode of the system (bypass, online, offline) may be altered by the operators at any time via the system command package. The mode control commands may be entered via any of the sources, and the IPL source serves as the initiator for the system input program.

The BYPASS Command

The BYPASS command (ZBYPASS, Figure 57) will be actuated in several different ways, depending upon the source. If the source is IPL, SYSMOD is set to BYPASS(10), and the SPPBT1 indicators are set to 11. These indicators are tested by the overlapped mode change commands to determine whether tape IOD's have been previously assigned or not. The IPL sequence then leads to ZBA1.

If the source was Job Control 1 and the present mode is overlapped, the STRANB indicator is set so that on $RET, Job Control 1 will stop requesting cards from System Input. A branch is then made to ZBA1.

If the operator's console is the source and the present mode is offline overlapped a branch is made to ZBA1. ZBA1 is an MCP calling sequence linkage to the input program, requesting a transition to the bypass mode. Upon return, the index linkage is set for the command acceptance message and exit is made to ZEXITX.

If the source was Job Control 4 in the bypass mode and STRANB is 1, STRANB is set to 0 and a branch made to JNOPX (Figure 57). This is done because the input program will give the BYPASS command card to JC4 as the first card of the first bypass job. Since this command has already been actuated, it must be ignored. If STRANB were =0, an error exit ZEREX2 branch would be made. If the source was JC4 in an overlapped mode, the BYPASS command was on an offline tape, and return to overlapped has been accomplished after entering bypass from phase 1, JNOPX is entered. Any other source or mode is in error and branches to ZEREX2.

The ONLINE Command

The ONLINE command (ZONLIN, Figure 57), like BYPASS, is source-dependent for its actions. If the source is Job Control 4 in the bypass mode, a branch is made to ZONO1Y where STRANB is turned off. This is necessary because STRANB might have been

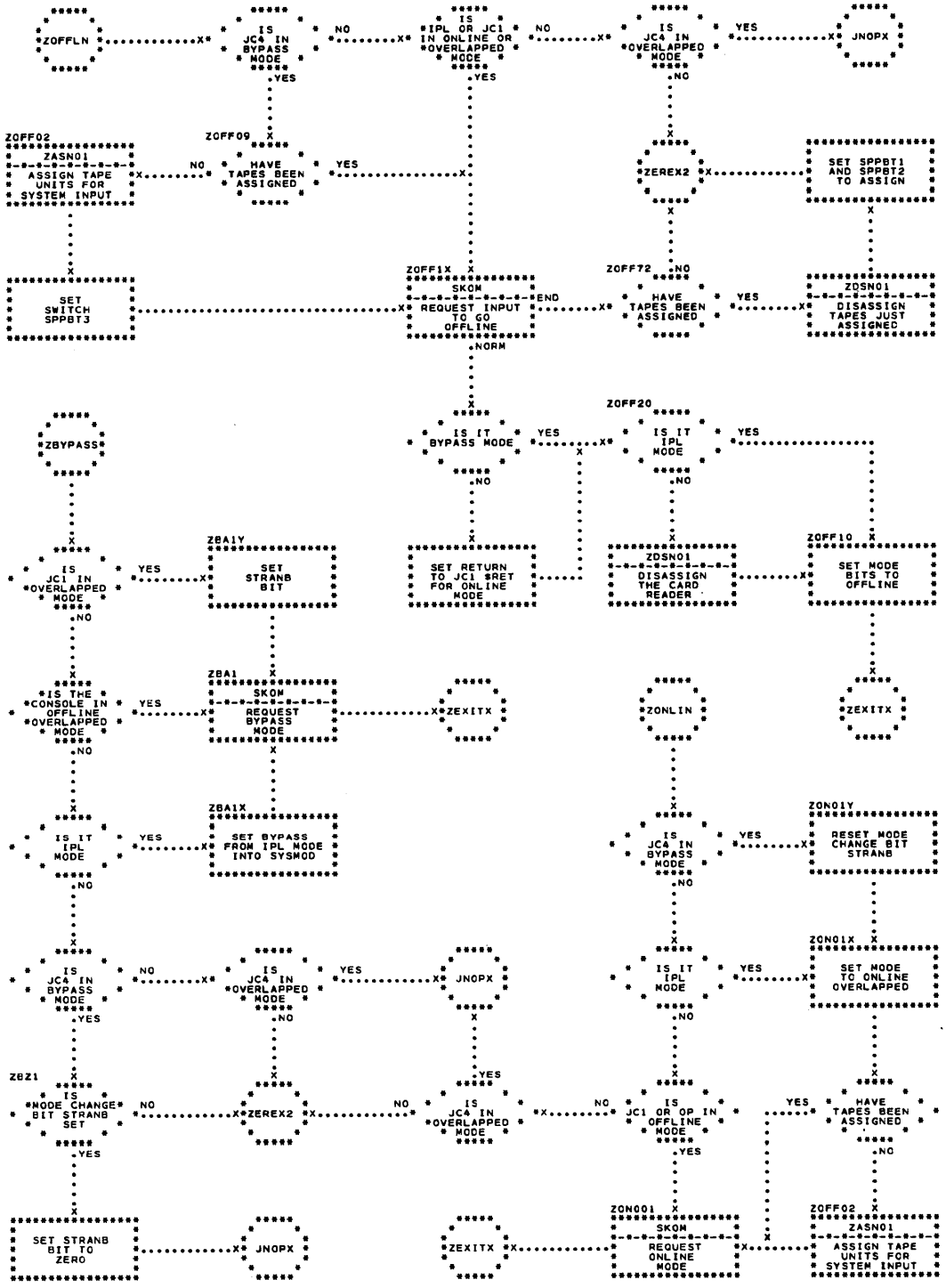


Figure 57. Mode Control Commands

set by a BYPASS command, appearing at the end of a scan tape, in the offline mode. STRANB would not be reset since the BYPASS command card in offline mode is not passed on to JC4 as the first card of the bypass job. Control then passes to ZONO1X.

If the source is IPL, a branch is made to ZONO1X, where SYSMOD is set to online mode (00), followed by a test of SPPBT2 to determine if tape IOD's have been assigned. If so, a branch is made to ZOFFO2, a subroutine consisting of two calling linkages to the special assignment routine. Both branches go to ZON001.

If the source is JC1 or operator's source and the mode is offline, a direct branch is made to ZON001 which is an MCP calling linkage requesting the input program to make a transition to the online mode. This is followed by setting up of the acceptance message and a branch to ZEXITX.

If the source was JC4 overlapped, the NOP exit JNOPX is taken. Otherwise the command is in error and a branch is made to ZEREX2.

The OFFLINE Command

The OFFLINE command (ZOFFLN, Figure 57) is also source-dependent for its actions. If the source is JC4 in the bypass mode, a branch is made to ZOFF09 to turn off STRANB. This is the same situation as ZONO1Y in the ONLINE command. SPPBT1 is then tested for previously assigned system tapes, and if not assigned, control is given to ZOFF02. An indicator, SPPBT3, is also set for test later to show that the tapes were assigned by this command. In either case, control eventually goes to ZOFF1X. If the source was IPL or JC1, online control is given directly to ZOFF1X. ZOFF1X is an MCP calling sequence to the input program requesting initiation of a transition to the offline mode of operation. There are two returns from the input program: one for normal and an end return if the command card is not the last card in the card reader.

If the normal return is reached, a test of mode is made; if online, the return address to JC1 is altered to go to a $RET sequence to JC1. In either case mode is then tested for IPL. If not IPL, the card reader is disassigned via the special disassignment routine ZDSN01. Both branches now lead to ZOFF10 which sets SYSMOD to OFFLINE(01), sets the present-command-assigned-tape indicators off (SPPBT3) and exits via ZEXITX.

If the end return is given by the input program, a branch to ZOFF72 is made. If tapes were assigned earlier by this command, they are disassigned and the assigned tape indicators SPPBT1 and SPPBT2 are reset to 1. Either branch leads to the error exit ZEREX2.

Job Control Commands

Five commands are available to operations personnel to control the flow of work through the system. They provide the following capabilities:

1. Change the time clock ($TC) calibration constant (CLOCK).
2. Cause the rejection of a job already on or partially on the scan tape (REJECT).
3. Cause the termination of the job in progress with or without an error dump (ABEOJ or EOJ).

The CLOCK Command

The CLOCK command may be primed by setting the binary keys at IPL time, or entered via the console typewriter or system input after IPL. If the command is accepted, the difference between the time in the command and $TC will be stored in STIMEK for use on all subsequent $TIME operations.

The program (ZTCC, Figure 58) prints on the console typewriter the time computed using both the old and new calibration constants and the date. The new time and date are saved to be written on the output tape with the job card.

The COMMENT Command

The COMMENT command is used for communication between the operator and the programmer.

The matrix mask determines the actions taken by the command (VVCOMJ, Figure 58). If the source is the operator, a branch is made to the VVOPTP routine where the message is converted to A8 and written on the output tape. Exit is made via ZEXITX. If the command comes from Job Control 1, online or offline, a normal return is made via JNOPX. A request from Job Control 4 overlapped causes the comment to be given to the operator as well as written on the output tape.

The REJECT Command

The operator may pre-reject a job by use of the REJECT command. This should be used only in the overlapped mode via system input, but it will act like the EOJ command if entered via the operator's source in the bypass mode.

The matrix mask is tested for validity of the request (ZREJCT, Figure 59). If the source is operator and the mode is bypass, a branch is made to the $EOJ routine for processing of end-of-job. If the request is from Job Control 4 overlapped, a normal return is made via JNOPX; if from the input program card reader in the unoverlapped mode, an error message is sent to the operator via ZEREX2. If the command is entered through Job Control 1, the return

address parameter in the tentacle table is set to error return and exit is made via ZEXITX. This will return control to Job Control, which will note the error return and the code, and reject the previous job.

The EOJ Command

The End of Job command is used to terminate the presently operating problem program without producing a dump.

The matrix mask is tested for the valid conditions for EOJ (JEOJ, Figure 59). These consist of a request from the console or from Job Control 1 in the online mode. If either of these conditions are met, $EOJ is primed, a message is printed through the output program, and the operator is informed of the operation. Bit YEOJS is set to permit JC4 to terminate a processor chain. A normal exit is then made through the mainstream exit routine via ZEXITX.

If the matrix mask shows that Job Control 4 is the source and the online mode is in control, the command has already been executed, and a normal return is made via JNOPX. For any other masks, an error exit is made via ZEREX2. If EOJ is given between jobs or before the first job, an error exit to ZEREX2 is made.

The ABEOJ Command

The ABEOJ command is implemented in the same way as EOJ, except that $ABEOJ is primed; if $ABEX is in effect, control is returned to the PP. If $ABEX is to get control, the problem programmer is informed that the operator requested an abnormal end of job and the operator is told that the PP has been given control through ABEX. The PP is only given one return through ABEX from an ABEOJ command (ZABEND, Figure 59).

I-O Control Commands

Four commands influence I-O operations. Two of these, OUPUT and REWIND, are concerned with terminating operations of the system input and output tapes. The EOF command is used to indicate an intentional EOF at the card reader in the online mode. The fourth, IOCHANGE, is concerned with changes in the availability of physical I-O devices.

The OUTPUT Command

The OUTPUT command is designed to permit the operator to force a change of the output tape before the beginning of the next job, thus starting the next job on a new output tape.

The command is accepted (JOUTP, Figure 60) if the matrix mask shows Job Control 4 in any mode, or if the source is the console. The pseudo-op, SOUTPT, is given to the output program. Upon return from the output program, KSILO+1.30 is tested to determine if the NWIPL option is requested. If not, a normal exit is made via ZEXITX. The request for the NWIPL option causes an automatic IPL of the NWIPL tape after a successful Update 30 run. The tentacle table for the MVDISK routine is entered at RNWIPL.

If the matrix shows that Job Control 1 is the source of the command, a normal exit via JNOPX is given since the command will be executed in phase 4. Any other matrix is an error and an error exit will be made via ZEREX2.

The EOF Command

The EOF command is used to permit the input program to rewind the write tape at tape switch time if necessary. It is a substitute for a non-existing job and means that there are no more cards available at the present time through the card reader, and that an end of file may be written. It also is used by the logger entries as an accounting device for idle time.

If the source is JC1 in online mode, an entry to SLOG1 is made at ZEOF19 (JEOF, Figure 60). Following this, a branch is made to JDOIT which is an MCP calling linkage to the input program to indicate an EOF condition. If there are more cards following the EOF card, an end return will be made, which causes the EOF command to take an error exit ZEREX2. Otherwise, the normal exit (ZEXITX) is made. If the source is JC4 in bypass mode, an entry to SLOG4 is made at ZEOF10; upon return, control is given to JDOIT.

If JC1 in offline mode is the source, ZEOF19 is entered and then exit made via JNOPX. If JC4 overlapped is the source, ZEOF10 is entered and then exit made via JNOPX. Any other source causes error exit via ZEREX2.

The REWIND Command

The REWIND command is a request to terminate and rewind the present online write tape.

The console is not considered a legal source for this command. The input program card reader is the only source, and the parameter REWIND is the only one following COMD, as in: COMD,REWIND.

The only acceptable matrix for the REWIND command is the Job Control 1 source and the online mode (ZREWCD, Figure 60). The pseudo-op, SKOM, is given to the input program and a comment to the operator precedes a normal exit from the routine. If the matrix is Job Control 4 online, a normal exit is made since the command has been executed already. All other matrices are considered in error.

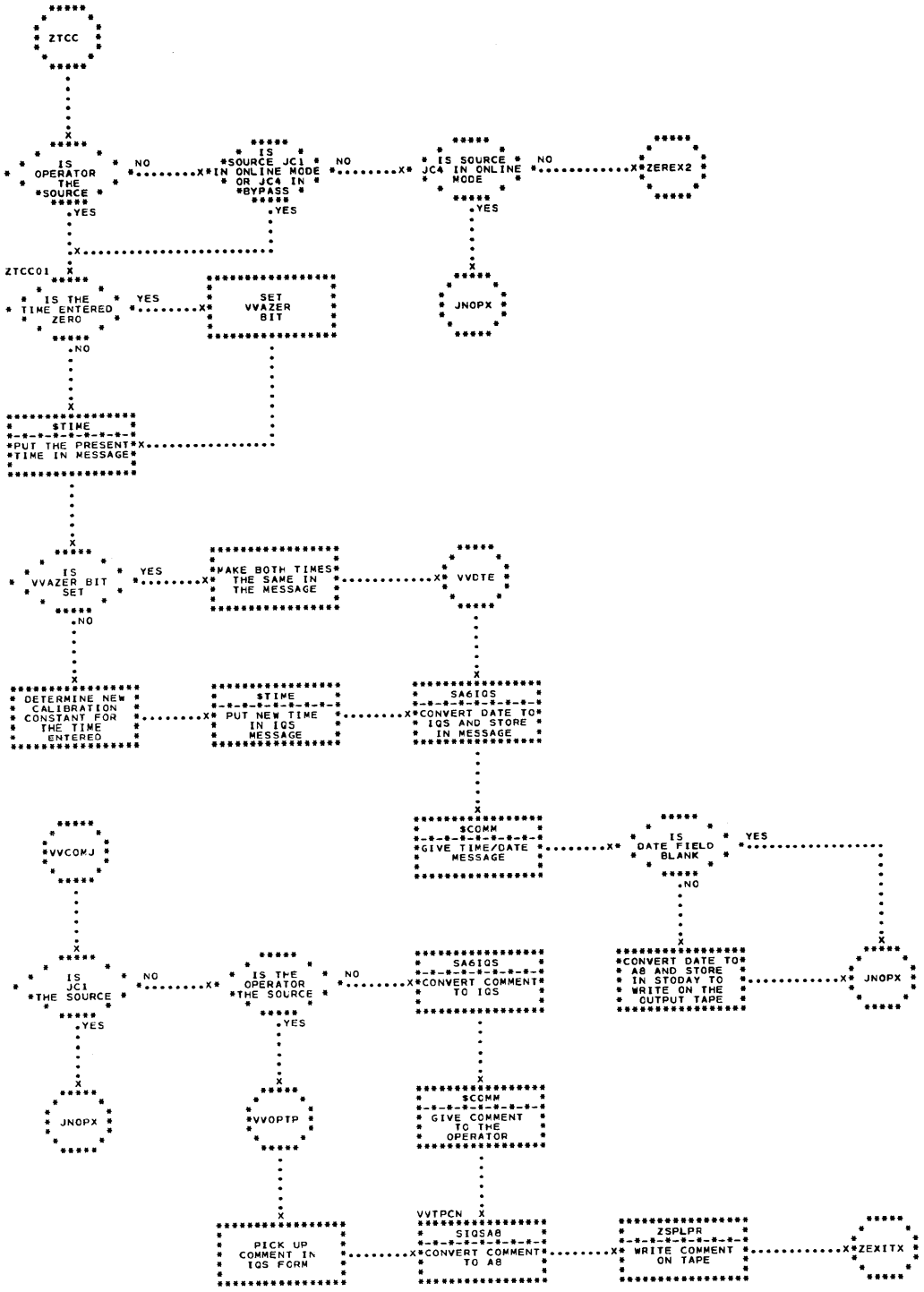


Figure 58. Clock and Comment Commands

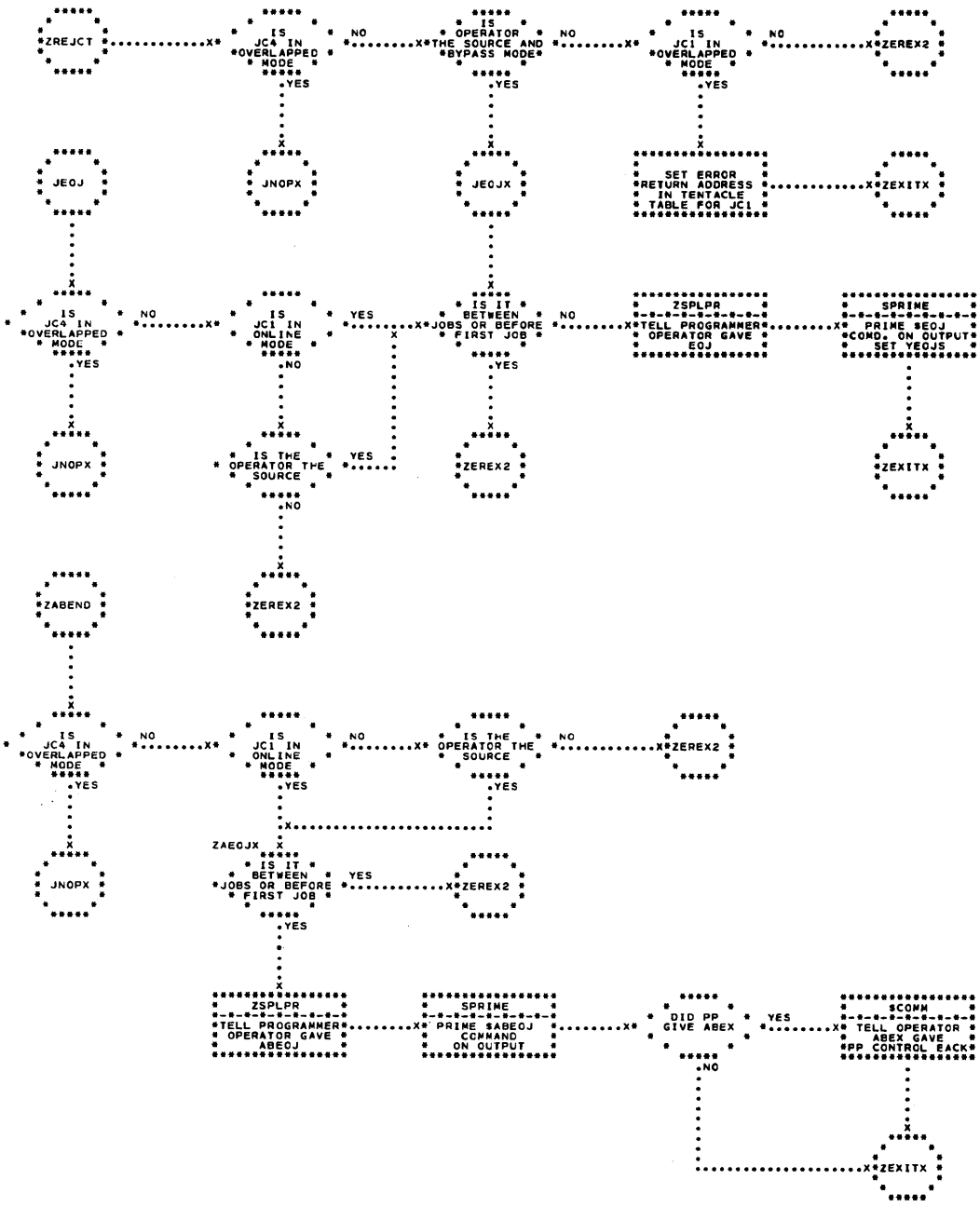


Figure 59. I-O Commands - Chart 1

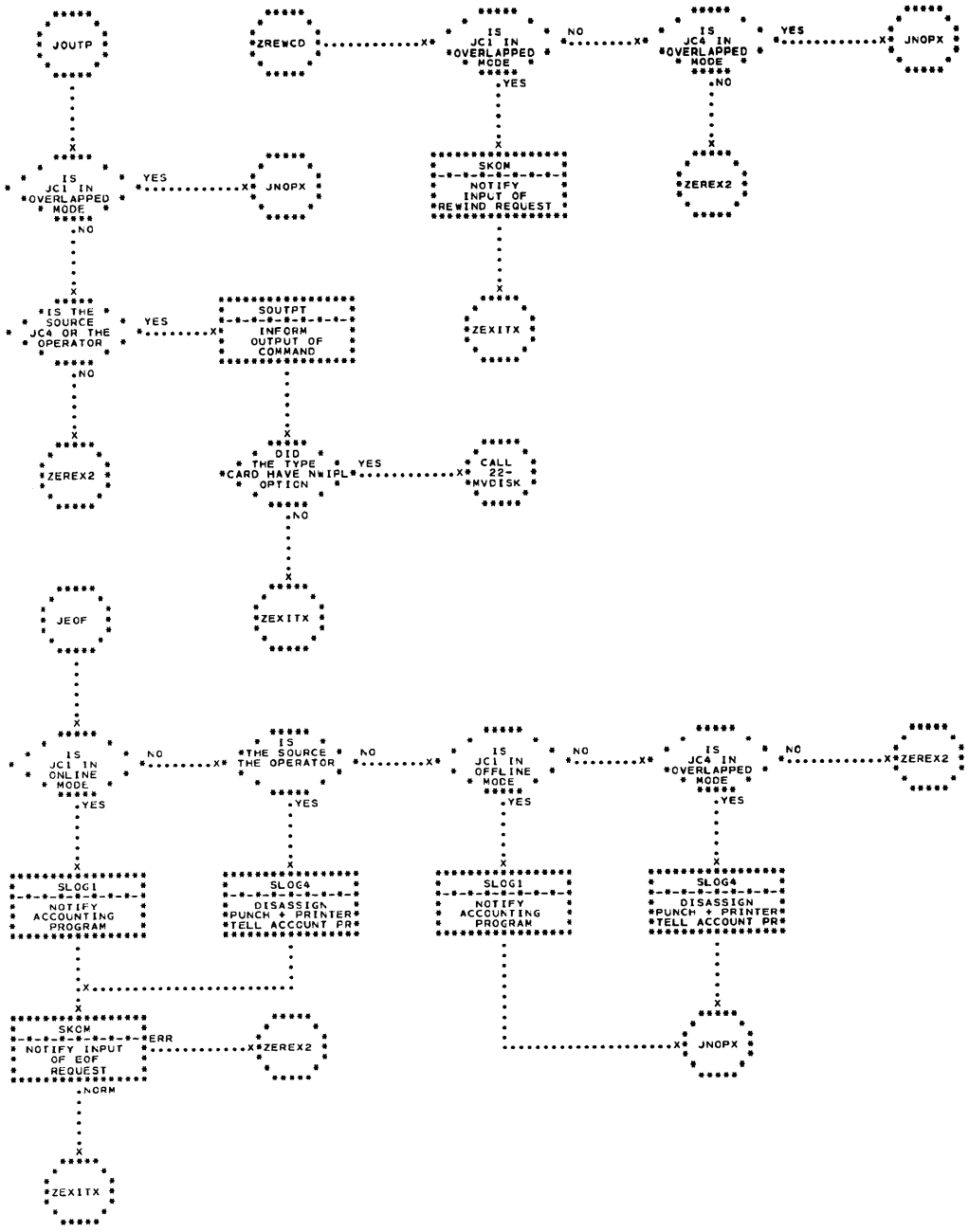


Figure 60. I-O Commands - Chart 2

The IOCHANGE Command

The IOCHANGE command is the method by which the operator may make a unit or an entire channel available or not available to the system. If an MCP unit or channel is made not available, an attempt will be made to assign a similar piece of equipment to MCP. A unit or channel is available if it is physically capable of being operated by the system. A unit is assigned if it is presently logically connected to either MCP, or the PP which is presently operating.

The format for both console and card is:

COMD, IOCHANGE, Channel, Unit, Code, Type

where:

Channel is the decimal number of the channel.

Unit is the decimal number, 0-7, of the unit. It is applicable only when a unit is being acted upon as distinct from a command relating to an entire channel. In the latter case, the symbol ALL should be used.

Code permits one of three available options to be requested:

ADD -- this makes a unit or channel available.

DELETE -- this makes a unit or channel unavailable and is normally given in response to a service request from a program which has had repeated failures on a unit or channel.

DELETM -- this makes a unit or channel unavailable for maintenance purposes. It is restricted to units or channels which are not assigned to MCP or to an operating PP. An error return will be given if DELETM is requested for an assigned unit.

Type is used only if the Code is ADD. It must be one of the following:

READER
PRINTER
PUNCH

If the operation is ADD and the channel is a single-unit channel, a different type of equipment may be attached. If the subfield is null, the Channel Status Table will reflect the same type of equipment. For both channel and unit numbers, if a request is given for a piece of equipment which has no corresponding Channel or Unit Status Table entry, an error return will be given.

Note: If a deletion of an MCP unit is requested, it may cause the ending of the currently operating PP.

Validity Testing: The matrix mask for source and operating mode is tested for valid IOCHANGE situations (ZIOCH, Figure 61). If the request is from Job Control 4 in the overlapped mode, it has already been processed, and a normal exit is made via JNOPX. If the source is operator or Job Control 1 online, or unoverlapped Job Control 4, the command will be implemented. If these conditions do not exist, an error return is given via ZEREX2.

After validity checking of the command request, the channel number is obtained from the command message. If a disk channel, a branch is made to a special disk routine ZDSKCH. If it is not a disk channel, the channel number is tested for a valid basic exchange channel number. An error exit is made if the channel number is not possible for the present configuration. This is determined by reference to SXCHAN.

Unit or Entire Channel Change: Upon completion of validity testing of the channel number, it must be determined if the request is for a unit change or an entire channel change. If the request is for a unit change, the unit number is tested against the number of unit status table entries for the channel. If the unit number is high, an error exit is made. If the request is for an entire channel change, a switch PG1 is set, and in either case, unit or channel, control is returned to the next test at ZACTON.

If the channel is a multi-unit channel, the system configuration change bit SCHFCG is set on. At this point, a test is made of the entire channel change switch. The action for a unit or entire channel is very similar, but the two must be performed separately because of minor differences. They will be discussed jointly, with minor differences noted.

The operation code is tested, and if it is ADD, a branch is made to the unit or channel add routines ZADDCH or ZADDUN (Figure 61). If not ADD, the op code must be DELETE or DELETM. The available bit for the unit or channel is tested, and if not presently available an error exit is made to ZEREX1. If it is available, the channel or unit is made not available. At this point, if the request is to delete an entire multi-unit channel, a branch is made to ZMULCH (Figure 62). If the unit is not assigned (ZQUES1, Figure 62), a test is made for DELETE or DELETM at ZMAIND. If either, a normal exit is made; if neither, error exit ZEREX1 is made.

If the unit is assigned, a test is made for DELETE. If not DELETE, an error has occurred, since an attempt was made to delete an assigned unit for maintenance. In this case, at ZMNERX (Figure 62), the unit or channel is made available again, the configuration change bit is reset, and if DELETM, exit is made to ZEREX2, otherwise to ZEREX1. If the unit was owned by the problem program, a branch is made to ZABEOJ, $ABEOJ is primed, a message is written through the output program, a comment is made to the operator about the change, and a normal exit is made. If the unit was owned by MCP, a scan of the status

tables by ZCHSCN (Figure 63) must be made to find a replacement unit. After return from the scan, a normal exit is made via ZDONE to ZEXITX.

Multiple Unit Channel Deletion: In the case of a multiple-unit channel, when an entire channel is to be deleted, the special case of both MCP and PP owned units on the same channel must be solved. If PP units are on the channel, the PP must be removed by $ABEOJ eventually. If MCP units are on the channel, replacements must be found, each one individually by the scan routine. The procedure for deleting an entire multi-unit channel at ZMULCH begins by the count of units on the channel being established and used as a control by index. A test is made to determine if the indexed unit is assigned. If not assigned, a branch is made to ZQUS11 which tests if the code has been examined for DELETE or DELETM. If not, PG6 is set and the operation is tested. If already tested, the units on the channel are made unavailable. At ZQUES22 a test is made to see if there are more units to test on the channel. If so, a branch is made back to ZQUES1 to see if the next unit is assigned. If there are no more units to test, a switch PG5 is tested to determine if any units on the channel were assigned. If not, the code is tested at ZMAIND for validity, and error or normal exit is taken, depending on the outcome. PG2 is tested; if there were assigned units on the channel and at least one was PP owned, a $ABEOJ should have been given for the PP. In either case, whether or not there were PP units on the channel, a comment is made to the operator about the change, and a normal return made.

Previously, at ZQUES1, if the indexed unit tested was assigned, a switch PG5 was set for later use and the code is tested for legality. If illegal, a branch is made to ZMNERX and the channel is made available again, the configuration change bit is reset, and an error exit is made. If the code is a legal DELETE, the unit is made unavailable and the question of ownership arises for the unit under examination. If PP owned, a switch PG2 is set for later use, and then a test is made for any more units to be examined on the channel at ZQUES22, described previously.

If the indexed unit was owned by MCP and not PP, the unit is set unassigned, and a branch is made to find a replacement via the ZCHSCN scan routine. Upon return, a branch is made back to ZQUES22 to determine if there are more units to examine on the channel.

For the ADD code, a test is made at ZADDCH or ZADDUN of unit or channel availability. If already available, an error return is given via ZEREX1. If not, the unit or channel is made available and the status table is cleared. If a single-unit channel, the type of equipment on the IOCHANGE request is entered in the status table via ZINSRT, and a comment is made to the operator before normal exit.

At ZDSKCH, the disk routine tests if the request is for an entire channel. If not, an error exit is made. If so, a test for ADD code is made. If it is an ADD request, and the channel is already available, an error return is made via ZEREX1. If unavailable, the channel is made available, the configuration change bit is set, and a normal return is given via ZDONE.

Replacing an Unavailable MCP Unit: The scan routine (ZCHSCN, Figure 63) for MCP unit replacement attempts to find an unassigned unit of the same type to replace the MCP unit being made unavailable. If unassigned units are not to be had, the PP will be given a $ABEOJ and one of its units will be assigned to MCP. If even removal of the PP will not produce a unit for MCP, the operator will be told by $COMM, the VF of $9 will be cleared to indicate an error and control returned to the user. These objectives are accomplished in the following manner.

Indexes are initialized for the channel and unit status tables as necessary. At ZLOPO a test of equipment is made on the indexed channel status entry against the type of equipment on the channel to which the deletion has reference. If not equal, the channel entry index is incremented at ZLOP1 and a new channel is examined until no more channels exist, at which point it would branch to ZREMOV. At ZREMOV if no more channels exist to be examined and no available channels had equivalent equipment, an error exit is made with the appropriate comment to the operator at ZNTFND. If there are available channels of the same type of equipment, they must have had assigned units. Therefore, a second pass must be made through the scan to remove the PP units and give them to MCP. A switch is set at ZREMV1 so that this sequence is followed only once, since the assigned units could have been MCP and would not have been obtainable, and a branch is made back to ZCHSCN.

If similar equipment is found in the search at ZLOPO, then (at ZOKEQP) the channel availability is tested. If the channel is not available, the search is continued at ZLOP1. If the channel is available, the availability of a unit is queried at ZIGZAG. If the unit is not available, other units on the channel are examined, until either a unit is found which is available or no more units exist on the channel. In the latter case, return is made to ZLOP1 to determine whether any more channels exist for test.

When an available unit is found, control passes to ZAVLUN and again the question must be asked if this

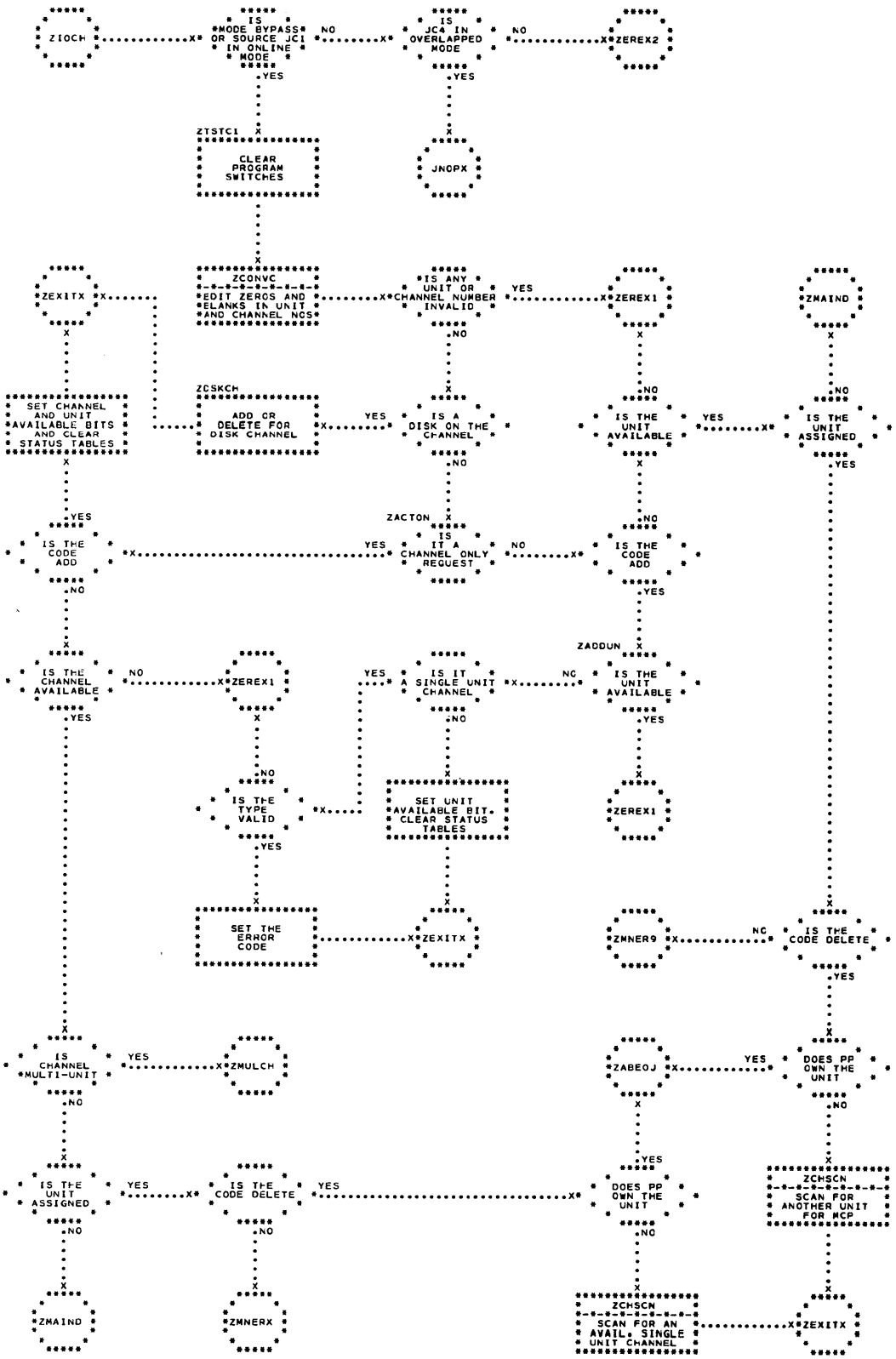


Figure 61. IOCHANGE Commands - Chart 1

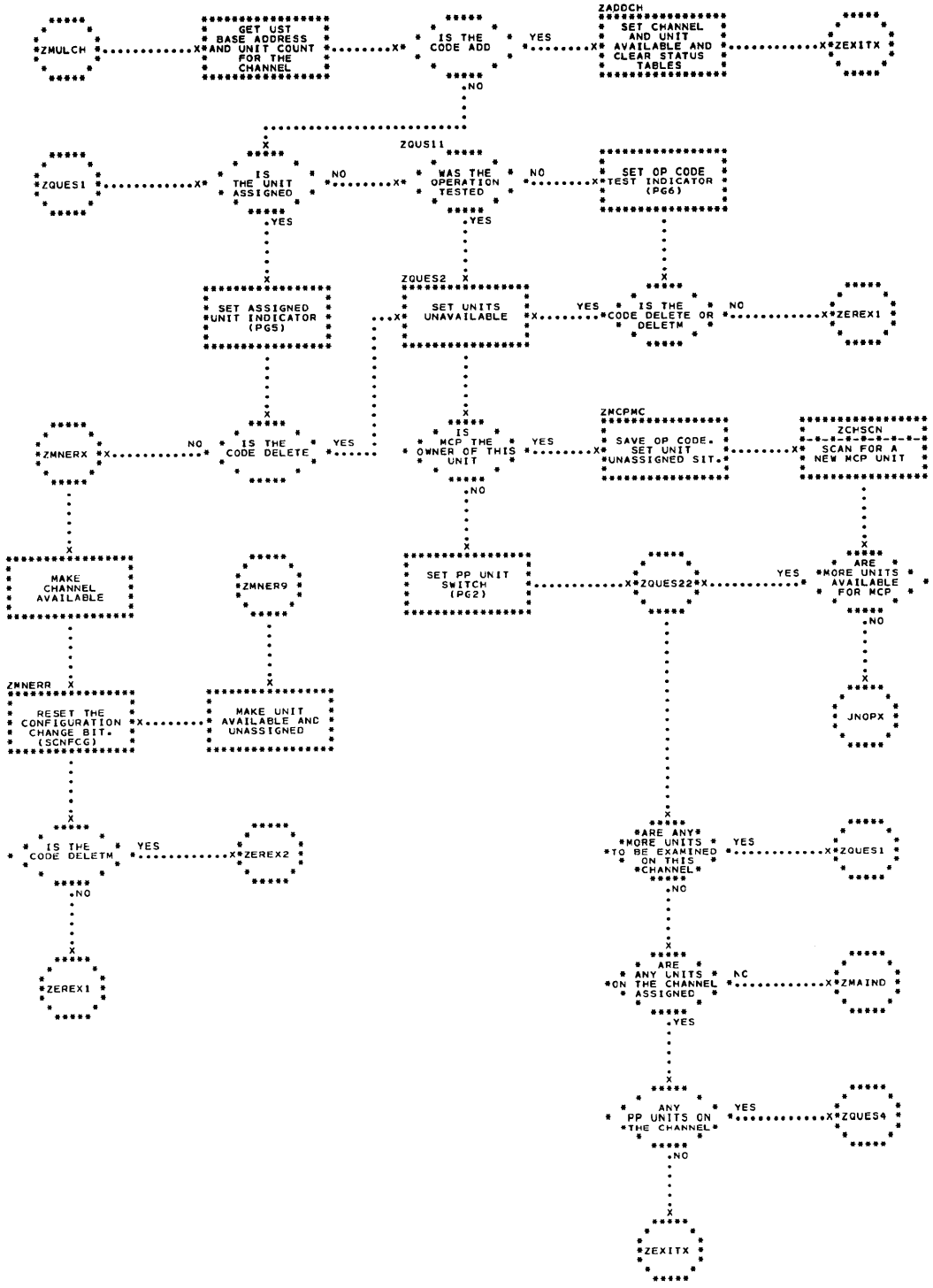


Figure 62. IOCHANGE Commands - Chart 2

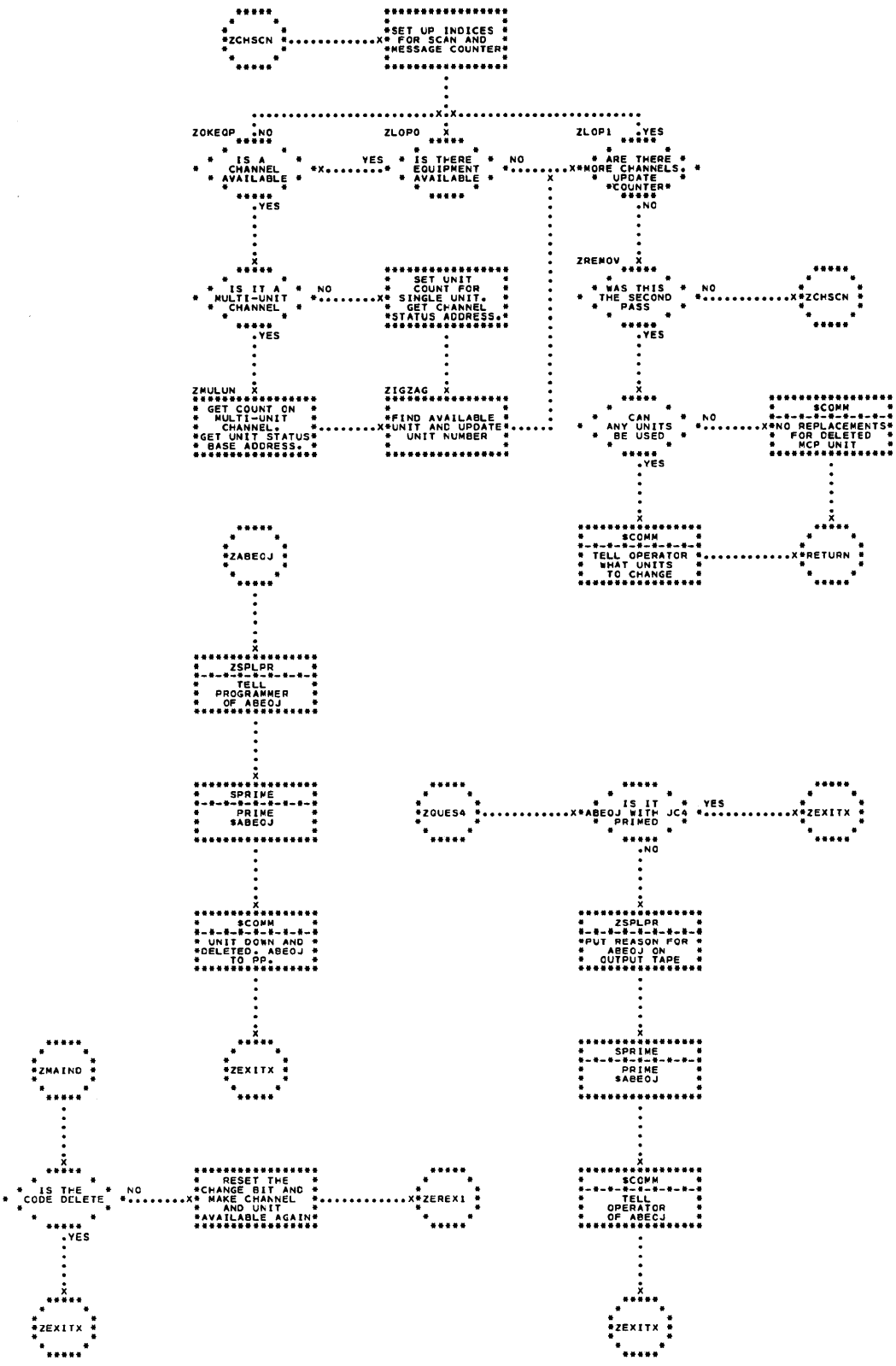


Figure 63. IOCHANGE Commands - Chart 3

is the second pass through the scan (PG0). If it is, the first PP assigned unit found will be taken. If the unit is owned by MCP, a branch is made to ZTSTPP to determine if any more units are to be tested. When a PP unit is found, the ownership bit is set to MCP and the unit is protected from unassignment when the eventual $ABEOJ is implemented. $ABEOJ is primed and a switch PG3 is set to tell other sections that the PP has been already primed for $ABEOJ. A message is also put out through the output program and a comment made to the operator. If the channel is in operation or in setup, the program will loop until the channel is not operating. A branch is then made to ZSETAS which will clean out the status tables of PP information, set a new unit area address in the unit status table, new unit and channel numbers in the unit area, and put out a message to the operator to change units. A return is then made to the user of the scan routine within IOCHANGE. Within ZSETAS there is a branch switch at ZLODLP for use by special assignment. This permits a branch out before the regular message generator of ZCHSCN is used.

At ZAVLUN, if it was not the second pass through the scan (PG0=0), and the unit being examined is assigned, a branch is made to ZTSTUN to test for more units. If an unassigned unit is found, unit ownership is set to MCP, and the above cleanup routine of tables, etc., is entered at ZSETAS.

The Special Assignment Routine

The special assignment routine (Figure 64) is used by the command package or the input program under the following circumstances:

1. A mode change has been requested which requires that tapes be assigned for use by the input program.
2. A mode change to off-line has been requested, and the card reader must be unassigned to be made available to problem program.
3. An IOCHANGE command has been requested which requires re-assignment of MCP units.

The special assignment subroutine consists of two routines using a common subroutine. When assignment is required, the linkage is:

```
SIC, ZCOM90
B, ZASN01
VF, (IOD number)
```

When disassignment is required, the linkage is:

```
SIC, ZCOM90
B, ZDSN01
VF, (IOD number)
```

The two routines, ZASN01 and ZDSN01, use a common subroutine, ZCOM01, to save indicies and set up status table references. Dispatcher error control is entered if the IOD number is zero (type 4), or if either a unit cannot be found to assign, or the unit to be disassigned is not presently assigned or is suppressed (type 82).

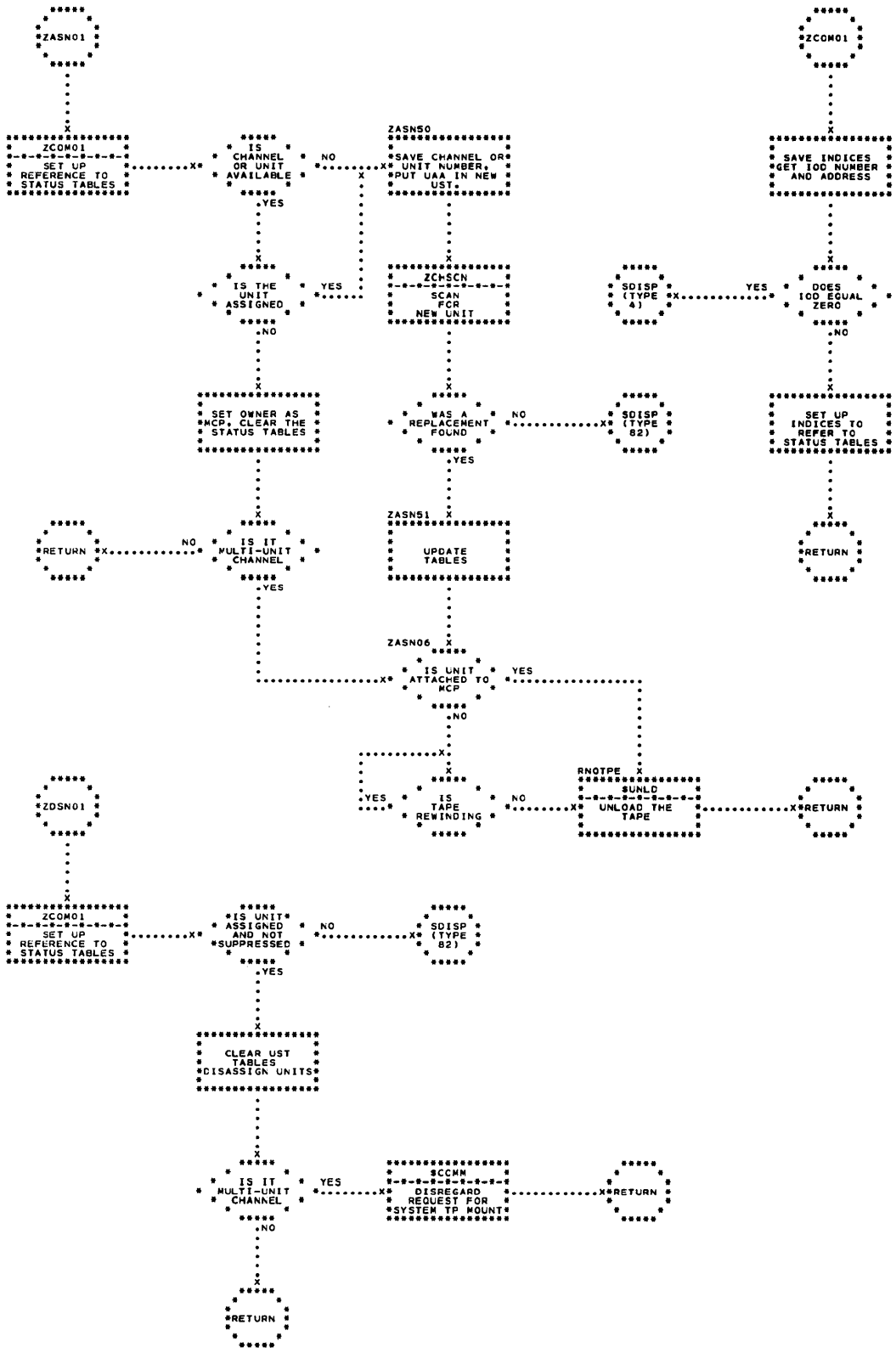


Figure 64. Special Assignment Routine

The three major packages in MCP that perform system I-O functions for both PP and MCP are:

1. The input program
2. The output program
3. The disk fetch program

The input program is designed to overlap preprocessing with input data transmission. It keeps two input sources active, one for phase 1 and one for phase 4, and handles the details of tape switch for the input sources and mode change. It performs one pseudo-op, $SCR, for both PP and MCP; and two others, SCR4 and SKOM, available only to MCP.

The output program blocks information into print and punch records on the output tape, and attends to output tape control and separation of the output of jobs. It performs the pseudo-ops $SPR and $SPU; the output command; and its special EOJ pseudo-op, SSPEOJ.

The disk fetch program performs the $FETCH pseudo-op to read named material from PROSA.

THE INPUT PROGRAM

The input program is a major package which performs three pseudo-ops. It uses two tentacle tables, and operates enabled but usually in SIO mode. Its basic function is to pass input (cards) to JC1 and JC4 (see system operation programs), and to the PP.

The logic of the input program is almost completely determined by system mode definitions. Therefore, before discussing the program itself, system input modes will be reviewed from the point of view of the input program.

System Input Modes

There are two logical modes of system input: overlapped and bypass. The overlapped mode is so named because two functions are being performed at one time:

1. Scanning of input (jobs) for purposes of preassignment of tape units.
2. Buffering of input (jobs) in order to execute jobs.

The processing of a given job is considered in two phases: Phase 1, scanning the I-O requests for the job; and Phase 4, the execution of the job.* The overlapped mode is further divided into two modes: online and offline, according to the phase 1 input source. If the phase 1 input source consists of cards from the online card reader, the overlapped mode is online. If the phase 1 input source consists of card images from a magnetic tape prepared by one of the system peripheral input programs on the IBM 1401, the overlapped mode is offline.

The second of the two logical system input modes is the bypass mode. The bypass mode is so named because the phase 1 function of scanning I-O requests is bypassed. All I-O assignment is done on a per job basis, immediately prior to execution of the job. The bypass mode is provided to allow interruption of overlapped operation under priority or emergency circumstances. Accordingly, the only input source in the bypass mode is the online card reader.

Overlapped Operation

The input programs use two buffers, the phase 1 buffer (VBF1) and the phase 4 buffer (VBF4). Each handles the buffering requirements of its phase. Each has a capacity of 34 card images, or two 17-column binary card blocked records.

Input from the phase 1 source is read into the phase 1 buffer as buffer space becomes available. If job control 1 is not currently processing cards, the SJC1 pseudo-op is primed. This results in an entry to JC1, which will resume its scanning of B cards. The buffer is used (scan pointer moved) only by SCR1 and SCAN requests.

The phase 4 input source is the read tape. All information on the read tape has passed through phase 1. Input is read from the read tape as space in the phase 4 buffer becomes available. However, a program will not be primed as a result of phase 4 input. The buffer is used (card request pointer moved) by phase 4 card requests ($SCR, SSCR4) and by the SEJSCN request.

If the overlapped mode is offline, the phase 1 input source is the scan tape. This tape will become the read tape for phase 4 when it has been completely scanned and the current read tape is completely processed.

If the overlapped mode is online, a tape must be prepared for use as the phase 4 read tape. This tape is called the write tape. In the online mode, phase 1 input is blocked onto the write tape as a part of phase 1 operation. In this mode, all tape mounting for the input tapes is eliminated since the read tape becomes

---

* The titles "Phase 1" and "Phase 4" were chosen to allow insertion of intermediate phases for multiprogramming purposes at such time as this is desired.

available for use as a write tape after it has been processed.

Bypass Operation

Only one functional buffer is required for bypass operation, since phase 1 is eliminated. Physically, the phase 1 buffer is used, since it must serve as a buffer for the card reader in online operation and card reader controls exist for it. Functionally, it is used as the phase 4 buffer to service $SCR, SSCR4, and SEJSCN requests.

Input Program Pseudo-Ops

The input program services three pseudo-ops: the two card requests, $SCR and SSCR4; and the input command pseudo-op, SKOM. The card request pseudo-ops differ slightly, as described in System Operation. However, they use a common tentacle table and are carried out by one program, the card request routine.

The SKOM pseudo-op is handled by another program and tentacle table. This pseudo-op has eight subtypes, referred to as the input commands. (See System Operation). They are:

SCR1	Phase 1 card request
SCAN	Phase 1 scan request
SEJSCN	Phase 4 scan request
SONL	Transition command to online
SOFFL	Transition command to offline
SBYP	Transition command to bypass
SEOF	Phase 1 command to communicate intentional EOF from the card reader (online mode).
SREW	Phase 1 command to terminate the write tape at this point (online mode).

General Organization

The input program may be considered in four categories:

1. The functional programs
2. The tape switch routines
3. The mode transition routines
4. The I-O fixup routines

The functional programs do the card processing work of the input program. The others do necessary housekeeping functions.

The Functional Programs

The seven functional programs are:

1. Card reader
2. Scan tape
3. Job boundary scan
4. Write tape
5. Read tape
6. Card request (overlapped)
7. Card request (bypass)

The input program processes information through two buffers, the phase 1 buffer and the phase 4 buffer; it is possible to associate each of the functional programs with one of these buffers according to system input mode (Figure 65).

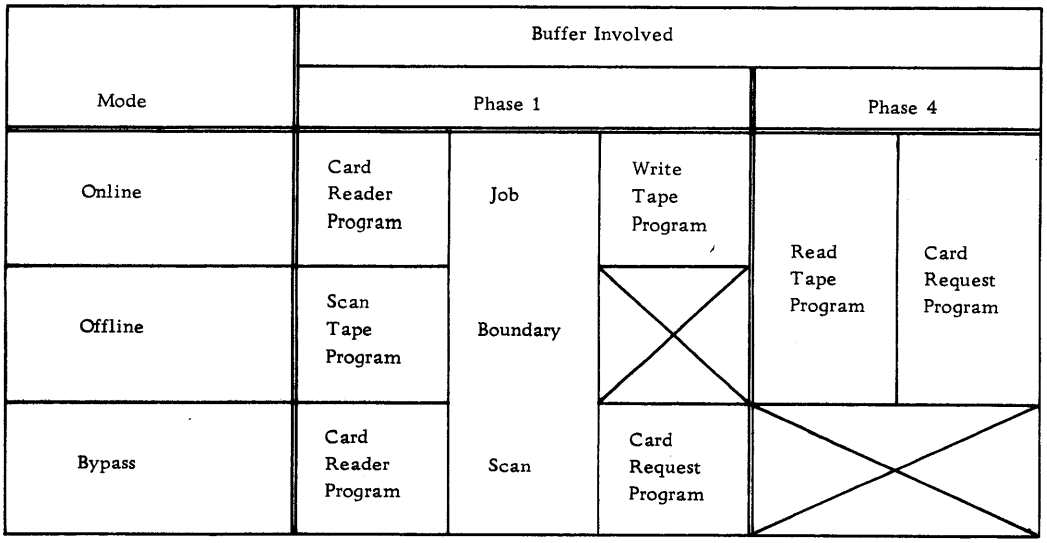


Figure 65. Functional Programs

A bookkeeping system involving control tables and queues is used by the functional programs to keep track of information and control flow through the input program. Each of the seven functional programs has an associated control table and queue. All control tables have the same format and usage, unless a type of information does not apply to the functional program. The seven queues are of various lengths, but all use half word entries of similar information.

The Tape Switch Routines

The input program has two tape switch routines: VTSA for the phase 1 tape, and VTSB for the phase 4 tape. VTSA rewinds the phase 1 tape and notifies the phase 4 programs of the availability of a new read tape. VTSB rewinds the phase 4 (read) tape, and either (online mode) makes it available as a new write tape or (offline mode) makes the drive available for mounting a new scan tape. Interlocks are provided between the tape switch routines and the functional programs.

The Transition Routines

There are six transition routines in the input program; one for transition from each mode to each of the other two. The choices, along with the names of the routines, are summarized in Figure 66.

Mode		Transition	
Requested	Present	Name	Symbol
Online	Bypass	Online from Bypass	VONLIB
Offline	Bypass	Offline from Bypass	VOFFLB
Bypass	Online	Bypass from Online	VBYPAN
Offline	Online	Offline from Online	VOFFLN
Bypass	Offline	Bypass from Offline	VBYPAF
Online	Offline	Online from Offline	VONLIN

Figure 66. The Six Transition Routines

Each transition routine has two basic functions: to put the requested mode in effect, and to insure that operation in the previous mode is terminated in an orderly manner with no information lost. This will generally require that the transition routines influence the operation of the functional programs and the tape switch routines.

The I-O Fixups

The input program uses three I-O units: the card reader, the phase 1 tape, and the phase 4 tape. The fixup routines must attend to error recovery, and must notify the appropriate functional program of the end of an I-O operation, either by EOP or EE.

Program Operation

The input program receives control by two means: a pseudo-op request, and an I-O interrupt. Control is given to the functional program most directly affected by that entry (e.g., the buffer scan program is entered when SCR1 or any scan operation is requested). If that program does anything to change the buffer situation, then control must be given to other programs affected by that change. When it is established that nothing further can be done by any programs, the input program gives up control with $RET.

Note that the same programs will be used whether the entry is a pseudo-op request or an I-O interrupt. Thus, it is necessary to run in the SIO mode so that input I-O interrupts will not be released.

The Functional Program Operation

Each functional program has a control table and a queue. The control words for the queue are stored in the control table. The control table is referenced by use of the symbolic index register, VU. In general, a functional program is entered when it is necessary to make entries in its queue, and, once entered, it performs as much processing as possible before yielding control.

Control Tables

Every control table has the same format and uses the same symbolism to specify a particular field in the table. Its format and symbolism are shown in Figure 67.

Transfer of Control

All functional programs ultimately return control to their predecessor. For a given mode, at most three functional programs will be involved in a normal pass through the input program. However, any one of the three may have been the one originally entered by a pseudo-op request or I-O interrupt, and must be the one to give $RET.

The functional programs use a scheme for passing control to successor programs which insures proper flow back through predecessors and, ultimately, issuance of $RET. Multiple passes are also allowed around the three programs, resulting in multiple entries to the same program. The scheme consists simply of a table of return addresses and a pointer which (effectively) points to $RET when it reaches zero. An index register, symbolically designated VP, is initially loaded with XW, VPDB, 0. A successor program is entered with the linkage

SIC, (VP); CB+, VP, VNX(VU)

or, in the event the table pointer, index VU, is one not permitted in the J field of a CB instruction, the linkage

SIC, (VP); CB+, VP, VNXT

is used. VNXT is B, VNX(VU).

The result is the development of a table of return addresses at VPDB. Return linkage then is

CB-, VP, -1.0(VP)

Queues

Each functional program uses a queue. In normal flow, a program is entered with an entry for its queue specified. It then tries to process as many of the entries in the queue as it can.

The length of the queues varies with program. An entry is a half-word, and is either a positive integer or a negative address. A positive integer represents the number of card positions ready for processing by that program in the buffer it uses. A negative address indicates the logical end of cards (i.e., job boundary), and the location to which the program should branch when the entry comes up for processing.

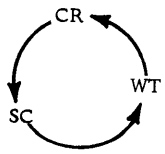
Parameter Flow

Each functional program has two basic responsibilities:

1. Performance of its basic function, such as initiating a card or tape read or transmitting card images, and
2. Advising its successor program of changes in the buffer situation.

The first responsibility is peculiar to the particular functional program. The second is standard among all functional programs, and involves the passage of parameters to the queue of the successor program.

Consider operations on the phase 1 buffer in the online mode. The three programs involved are the card reader control, the buffer scan program, and write tape control. For convenience of discussion, let these programs be represented by the symbols CR, SC, and WT, respectively. Then CR must advise SC of cards read and available for scanning, and SC must advise WT of cards scanned and available for writing, and WT must advise CR of buffer space (card positions) available for card reading. Control of the phase 1 buffer in online operation may be seen as a cyclic passage of parameters from one program to the next as illustrated by the circle.

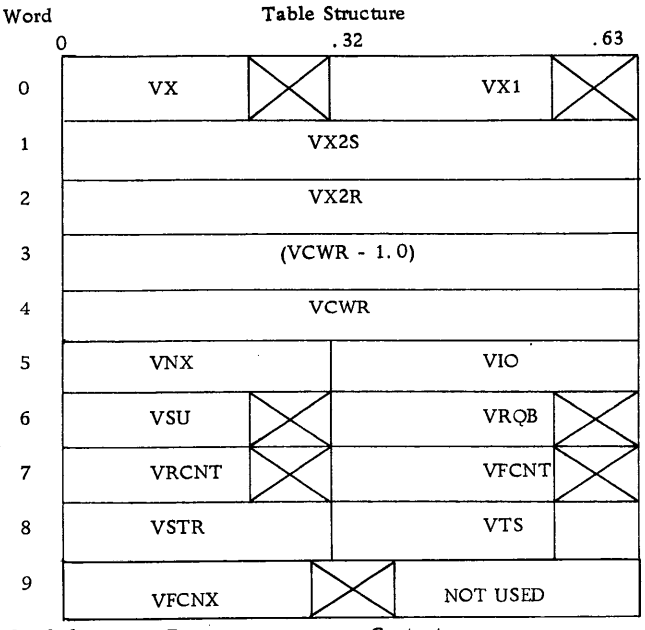


The function of the queue now becomes apparent. A parameter is passed by giving control to a program with the parameter specified for entry into its queue. Similarly, the functions of the control table entries VX, VX1, VX2S, VX2R, and VNX should now have assumed significance. Figure 68 illustrates this passage of parameters for both buffers in all modes, using the following symbolism for the functional programs:

Card reader program	CR
Job boundary scan program	SC
Write tape program	WT
Read tape program	RT
Scan tape program	ST
Card request program (overlapped)	Q
Card request program (bypass)	E

When a program receives an item for its queue, it stores it in the first available position in the queue. Then, if the program is not busy on a previous operation, the next parameter in the queue (not necessarily the one just entered) is removed and action based on the parameter is started. If this action results in a change in the buffer situations, the succeeding program must be notified by an appropriate parameter entry in its queue, and the process repeated.

There are several situations in which a functional program determines that a special action must be taken at a time which depends on the operation of its successor. The program then may pass a negative address to that program identifying the special action



Symbol	Format	Content
VX	VF	An integer between -34 and zero locating a card position in a 34 card buffer (phase 1 or 4). Not used in the table for the buffer scan program.
VX1	VF	An integer between 0 and 34 specifying the number of cards currently being processed by the program in the buffer. Used by all programs.
VX2S	XW	An XW to control storing in the program's queue. The VF specifies the next store address, and the count and refill fields are used to cycle around the queue. Used by all programs.
VX2R	XW	An XW to control removing entries from the program's queue. The VF specifies the next entry to be removed, and the count and refill fields are used to cycle around the queue. Used by all programs.
VCWR	CW	A CW to control transmission of data into or out of the buffers (VBF1 for phase 1, VBF2 for phase 4). If two control words are required, the word at VCWR-1.0 is used to control the transmission starting at the beginning of the buffer, and the first CW chained to it. These slots are not used in the buffer scan table, since that program does no data transmission.
VNX	Half Word	A half word branch to the logical successor program for the current mode. (See program linkage.) This is fixed for every program except buffer scan, in which case the successor is set by the transition routines according to mode.
VIO	Half Word	A half word branch to be used after an EOP to initiate additional I-O. Is not used in non-I-O programs; i.e., not in buffer scan or card request tables. The branch is permanent.
VSU	Address	IOD reference number for unit currently assigned to that program. Not used in the tables for buffer scan and card request programs. Variable in tape control programs, and set by the tape switch program.
VRQB	VF	Used by tape I-O control programs to store the residual after blocking; e.g., in the scan tape table the number of cards positions in the buffer still available after initiating a read of a block (17 cards).
VRCNT	VF	Used by the tape control programs to store the number of records processed in the current file.
VFCNT	VF	Used by the tape control programs to store the number of files processed on the current tape.
VSTR	Half Word	Permanent branch to proper entry point after tape switch. Used only by tape control programs.
VTS	VF	Used by read tape and scan tape control programs; contains a parameter for queue entry when end of tape is encountered.
VFCNX	CF	Used only in the table for the read tape program. Contains the number of files (jobs) remaining on the read tape.

Figure 67. Control Table Format and Symbols

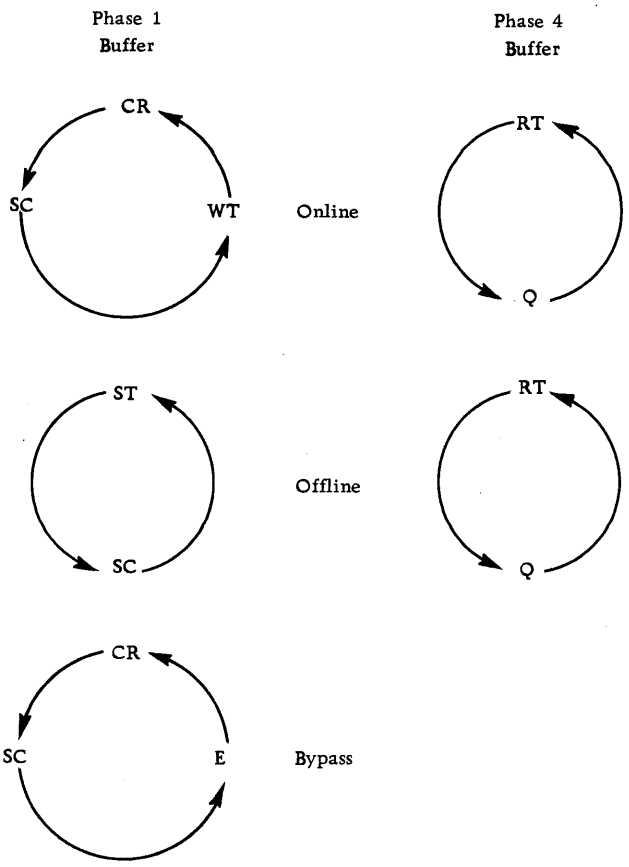


Figure 68. Passage of Buffer Parameters

Program	Table Symbol	Queue Symbol	Entry Point Symbols: Q-Entry	Pseudo-Op	EOP	EE	CS
CR	VCR	VRD2	VRDQ	VEOFB	VRDEOP	VRDEE	VRDCSS
SC	VSC	VSC2Q	VSQ	VSCAN, VSCR1	VSC10	---	---
WT	VWT	VW2	VWQ	VSREW, VEOFN	VWEOP, VWFEOP	RWEE	---
E	VE	VE2	VEQ	VREQ, VBEOJ	---	---	---
RT	VRT	VR2	VRQ	VOEOJ	VREO	VREE	---
Q	VQ	VQ2Q	VQQ	VREQ	---	---	---
ST	VST	VS2T	VSTQ	---	VREOP	VRTEE	---

Figure 69. Functional Program Symbols

to be taken when that point in its queue is reached. For example, if the job boundary scan program received 10 cards from the card reader program and found a job boundary after the third card, it would pass to the write tape program the parameters 3, -VWEFT, 7, in that order (one at a time). The write tape program would branch to VWEFT to write a file mark when it encountered the -VWEFT in its queue.

Each functional program has a subroutine whose function is to enter a parameter in the queue. The parameter is given in index register, VT5, and the various programs are entered at the Q entry point for this purpose. The symbols associated with each functional program are listed in Figure 69.

Queue Manipulation

Considering a queue as composed of N consecutive half words, then the queue store and remove references, VX2S and VX2R, are of the form:

K XW, Queue, N, K.

As the queue is used, VX2S and VX2R are occasionally updated by means of the instruction, V+ICR, (XR), .32 so that the references move upward in memory over the queue and then recycle starting at the low-order position. VX2S and VX2R are indirectly interlocked so that VX2R never passes VX2S. Thus, the queue is maintained on a first-in first-out basis consistent with processing buffer areas and negative parameters in proper order.

The VCOM Subroutine: VCOM (Figure 70) maintains the VX2S pointer so that it is always pointing to the last positive parameter entered. When positive parameters are received they are simply added into the position designated by VX2S, which remains unchanged. When a negative parameter is received VX2S is updated, the negative parameter is stored, VX2S is again updated, and zero is stored. An example is given for a queue of 5 positions:

Parameter Just Entered by VCOM		Queue after Parameter is Entered by VCOM				
Initially		0	0	0	17*	0
3		0	0	0	20*	0
-VWEFT	Step 1	0	0	0	20	-VWEFT*
	Step 2	0*	0	0	20	-VWEFT
7		0*	0	0	20	-VWEFT
2		9*	0	0	20	-VWEFT

* Indicates the cell referenced by VX2S

The accumulation of successive positive parameters is significant because it overcomes the many influences tending to fragment buffer areas (for example, the job boundaries).

The VSETX Subroutine: The VSETX routine (Figure 70) performs the function of removing from a queue a single parameter, returning to the calling program with the parameter if it is positive, or branching to the location addressed by the absolute value of the parameter if it is negative. The cell containing the parameter removed is set to zero. VSETX uses VX2R in removing parameters from the queue and updates VX2R in a manner that preserve the order and continuity of the queue. First, like VX2S, VX2R never remains on a queue entry, which is negative. If VSETX finds a positive parameter, it is removed and VX2R remains unchanged. If the entry is zero, the next entry is tested and if zero, then zero is returned to the calling program and VX2R again remains the same. If the next entry is positive VX2R is advanced to that slot and the parameter is removed. If the entry is negative, it is removed and VX2R advances to the following slot. The following example illustrates this process:

Queue after Parameter has Been Removed						Parameter Just Removed by VSETX
3+	-WEFT	7	6	0		Initially
0+	-WEFT	7	6	0		3
0	0+	7	6	0	Step 1	-WEFT
0	0	7+	6	0	Step 2	
0	0	0+	6	0		7
0	0	0	0+	0		6
0	0	0	0+	0		0

+ Indicates position of VX2R pointer

The interaction of the entry and removal process is exhibited in the following example:

Parameter Just Entered by VCOM	Queue After Parameter has Been Removed or Entered						Parameter Just Removed by VSETX
Initially	0+*	0	0	0	0	0	Initially
3	3+*	0	0	0	0	0	
2	5+*	0	0	0	0	0	
	0+*	0	0	0	0	0	5
-WEFT	0+	-WEFT	0*	0	0	0	
4	0+	-WEFT	4*	0	0	0	
	0	0	4+*	0	0	0	-WEFT
	0	0	0+*	0	0	0	4
2	0	0	2+*	0	0	0	
	0	0	0+*	0	0	0	2
1	0	0	1+*	0	0	0	
-WEFT	0	0	0+	-WEFT	0*	0	
	0	0	0+	-WEFT	0*	0	1

+ Indicates position of VX2R
* Indicates position of VX2S

It was mentioned previously that a functional program would, after inserting a parameter, remove and act upon the next parameter if it were not already busy. This action centers about the active parameter, VX1, whose value has a double significance. First, zero or non-zero respectively indicates whether the program is not busy or busy. Second, the value, if positive, is the number of cards in the buffer area being processed, and if negative, identifies the special operation now in progress. If VX1 is zero, the interface routine enters VSETX, which will remove a single parameter from VX2. If the parameter is non-negative, control returns with the parameter index register VT2, which is stored in VX1, and then the function characteristic of the program is performed on the buffer area. If the parameter is negative, VSETX makes a CB+ type entry to the location given by the parameter. Since VSETX is entered by a SIC,..;B,..; sequence, the net effect is as if the program calling VSETX had instead made a CB+ type entry to the special sequence addressed by the negative parameter. Each special sequence can thereby maintain the continuity of control. There are various alternatives that may be used depending mainly on whether the special sequence completes its operation immediately or only initiates it. In the former case, the instruction CB-, VP, SETX may terminate the sequence. That is, a CB- type entry to VSETX is effected, the net effect being as if a SIC,..;B,... type entry was made from the original calling program. This preserves the continuity of both program control and parameter flow. In the case when a special sequence only initiates an operation, parameter flow must be suspended until the operation is complete. This may be done by "faking" a return from VSETX with a zero parameter, or, subject to certain conditions, by returning one level higher than the program that called SETX. Here, the continuity of control is maintained but that of parameter must be held over in VX1, thus suspending flow continuity until an operation is completed.

The VSETUP Subroutine: VSETUP (Figure 70) is the subroutine usually used to maintain parameter flow. It is entered by a SIC,...;B,... type entry, and returns a zero parameter to the calling program if the program is busy; otherwise, it behaves like VSETX. The normal calling sequence is:

```
SIC, (VP); B, VSETUP
SV, VT2, VX1 (VU)
BXVZ, ---    'do no more work
---          'process new active parameter
```

VSETX is normally used when it is desired to resume parameter flow after an extended operation has been completed. In this case the normal environment is:

```
SIC, (VP); B, VSETX
SV, VT2, VX1(VU)
BXVZ, ---       'do no work
---             'process new parameter
```

The basic difference between VSETX and VSETUP is that VSETUP is used when the program may already be busy, as when entered at the queue entry point, and VSETX is used when the program is known not to be busy as at the EOP entry.

The VBLOC Subroutine: The VBLOC subroutine (Figure 70) is used in the same manner as the VCOM subroutine, but it has additional functions. It is used by those programs concerned with blocked tape records. Instead of entering a parameter in the queue directly, it adds it to VRQB, the residual after blocking. If the result is less than 17, it is stored in VRQB and control returned to the user. If the result is 17 or greater, 17 is entered in the queue, VX2S is stepped, and subroutine VACTX used to attempt an I-O operation. When control is returned, the process is repeated until the residual is less than 17, where upon it is stored in VRQB and control returned to the user.

Buffer Processing

Every positive quantity put into VX1 represents an amount of buffer area which must be processed by the functional program. Generally, the processing routines must give up control after initiating an operation, and, in order to maintain the continuity of overall operation, the input program must have control when the processing of a buffer area is complete. This control comes about in three ways:

1. The input program completes the processing of the buffer area solely under its own control. This occurs when job boundary scanning in bypass mode and in overlapped mode when it is not necessary to prime job control one (JC1).
2. The input program has control as the result of an externally generated request for a card or cards which results in the last card of an area being used. The requests possible and the related functional programs are SSCR4 (E or Q), $SCR (E or Q), SCR1 (SC overlapped) and SCAN (SC overlapped).
3. The input program receives control through one of its tables of exits after the completion of an I-O operation.

All of the buffer processing programs have to update VX to reflect changes in buffer area origin. Initially all of the VX parameters in a circle of functional programs are in phase, reflecting the fact that each functional program will begin processing on the same card image. By the time each has finished processing a buffer area, VX will be updated by the amount VX1.

The VMD Subroutine: To process a buffer area it is usually necessary to convert VX and VX1 into an explicit representation of memory area which usually takes the form of a control word or words. This conversion is done by the VMD subroutine (Figure 71) which modifies the two control words in the program table according to two parameters given it representing buffer origin and buffer length. The two control words are located at VCWR-1 and VCWR within the program tables of CR, WT, E, and Q, and a typical set is:

```
CW, VBF1, 0, $      VCWR-1.
CW, 0, 0, 0         VCWR
```

All but the count field of the first word is a constant which serves as a model for VMD in constructing explicit control words; in particular, the value field contains the origin, VBF1 (phase 1) or VBF2 (phase 4), to which the program applies. If x, an integer between -34 and -1 inclusive representing buffer origin, and C, an integer between 1 and 34 inclusive representing buffer length, were the parameters which VMD received, VMD would operate as follows:

1. The value field of VCWR is replaced by 15* (34+x) plus the value field of the model.
2. The multiple bit of VCWR is set equal to that of the model.
3. The count field of VCWR is replaced by 15*C if $C+x \leq 0$ or 15 * (-x) if $C + x > 0$. If $C + x > 0$, it means that the buffer area wraps around the end of the physical buffer and therefore an additional step (4) is needed to produce the necessary chained control words.
4. If $C + x > 0$, set the count field of VCWR-1. equal to 15 * (C + x), turn on the chain bit of VCWR, and set the refill field of VCWR equal to that of the model.

In the case of RT and ST where only blocked read operations are done, it is only necessary to perform the instruction R, VCWR(VU) to obtain the control word for the next buffer area. SC does not require control words in its functioning and employs a special technique which is discussed later.

Buffer Process Categories: The processes performed on buffers fall into two categories: CPU processes and I-O operations (CR, RT, WT, ST).

Buffer processes involving only CPU operations are:

1. Servicing of request for card transmission (SCR, SCR4).
2. Servicing of requests for the address of a single card in the phase 1 buffer (SCR1).

3. Scanning to job boundary (SCAN, SEJSCN).

The first item uses VMD generated control words to control the transmit instruction used to move card images from the buffer. As each request is processed, the VX and VX1 parameters are updated and a parameter passed on to the successor. Thus processing of a given active parameter takes place on a piecemeal basis which differs from the I-O situation in which an active parameter is processed completely in the EOP procedure. However, both are similar in that when processing of a buffer area is complete, an attempt to resume parameter flow is made. Notice that a request for more than one card may exhaust the present VX1 but still not be completely satisfied, in which case the updating, resumption of parameter flow, etc. must be accomplished before the rest of the request may be processed. Here the flexible linkage mechanism is very helpful.

The second item is similar to the first except that control words are not used and the VX parameter does not exist in the integer form previously discussed; rather, an index word of the format,

K XW, BF1, 34, K,

serves the same purpose and is updated singly by the instruction V + ICR, VS1, 15. as each card is processed. The third item is a control process involving the positioning of buffer pointers at job boundaries on request.

The I-O operations are performed by three I-O actuation routines: VRTIO, VRDIO, and VWIO (Figure 71). All are entered through the VIO slot in the table from the VACT, or VACTX subroutine which sets up appropriate parameters, x and C, in index registers VT3 and VT2. Routines VRDIO and VWIO derive the appropriate control words using VMD, and perform the appropriate MCP I-O pseudo-op. The VACT, VACTX subroutine and VEOP routine perform basic functions in this process.

The I-O Actuation Subroutines: The VACT and VACTX subroutines (Figure 71) perform several functions necessary to start an I-O operation on a buffer area and are used by the WT, RT, ST, and CR functional programs. First either VSETX or VSETUP (VACT or VACTX) is used and a CB- type return is given if a "do no work" condition exists. Otherwise a branch is made to VIO(VU) after VX has been adjusted. At the VIO slot in the table there is a branch to the appropriate I-O actuation routine.

The distinction between VACT and VACTX is that VACT presumes that the I-O unit is available for actuation if the need exists, whereas VACTX presumes the need exists but that the I-O unit may be in use. The distinction is analogous to that between VSETUP and VSETX.

The Standard EOP Routine: When an operation is completed (as signalled by an I-O interrupt, or the request for the last card of a buffer area by SCR1, SCR or SCR4) parameter flow through the program is resumed. The VEOP routine is used to perform the following three steps:

1. The VX parameter is updated by VX1 to reflect a new buffer origin (If VX1 >0 only).
2. A parameter is removed from the queue, thereby reinstating parameter flow.
3. The old parameter is passed on to the successor program, thereby maintaining parameter flow. This is not done for most negative parameters because these do not propagate continuously around the functional program circle, but rather originate within one and end with the successor.

The Card Reader (CR) Program

The first functional program to be discussed is Card Reader (CR). The CR reads cards into buffer areas under the control of the parameter flow process. When an out-of-material condition at the card reader arises, it is recognized and appropriate measures taken. The EOP procedure is standard, using the VEOP routine.

The out-of-material condition is signalled by a return through the EE slot in the table of exits. Control goes from there to the VRDEE routine (Figure 72) which effectively passes on to the successor program those cards successfully read, saves in the address of VREE1 the number of cards not read (residue), and updates VX by the amount read but leaves VX1 unchanged. The last step inhibits the issuance of subsequent read instructions as the result of parameters entering the queue. Finally, a special negative parameter, -VEES, is passed to the successor program, job boundary scan (SC).

This is the first step in a set of events which may result in the message, "Service the card reader" being given to the operator. This message cannot be given immediately because the EOF and OFFLINE commands, which can inhibit the message, are recognized only later in SC, if overlapped, and still later in E, if unoverlapped. At the appropriate time, VEES effectively tests whether or not the EOF condition is in effect and if not, it then puts out the message via the commentator.

Upon the receipt of a CS, control goes to the VRDCSS routine. This resumes card reading using the residue previously stored if the unit ready bit in the control word is on, and it also removes the EOF condition if it existed. The I-O routine, VRDIO, and the queue entry, VRDQ, perform as previously described.

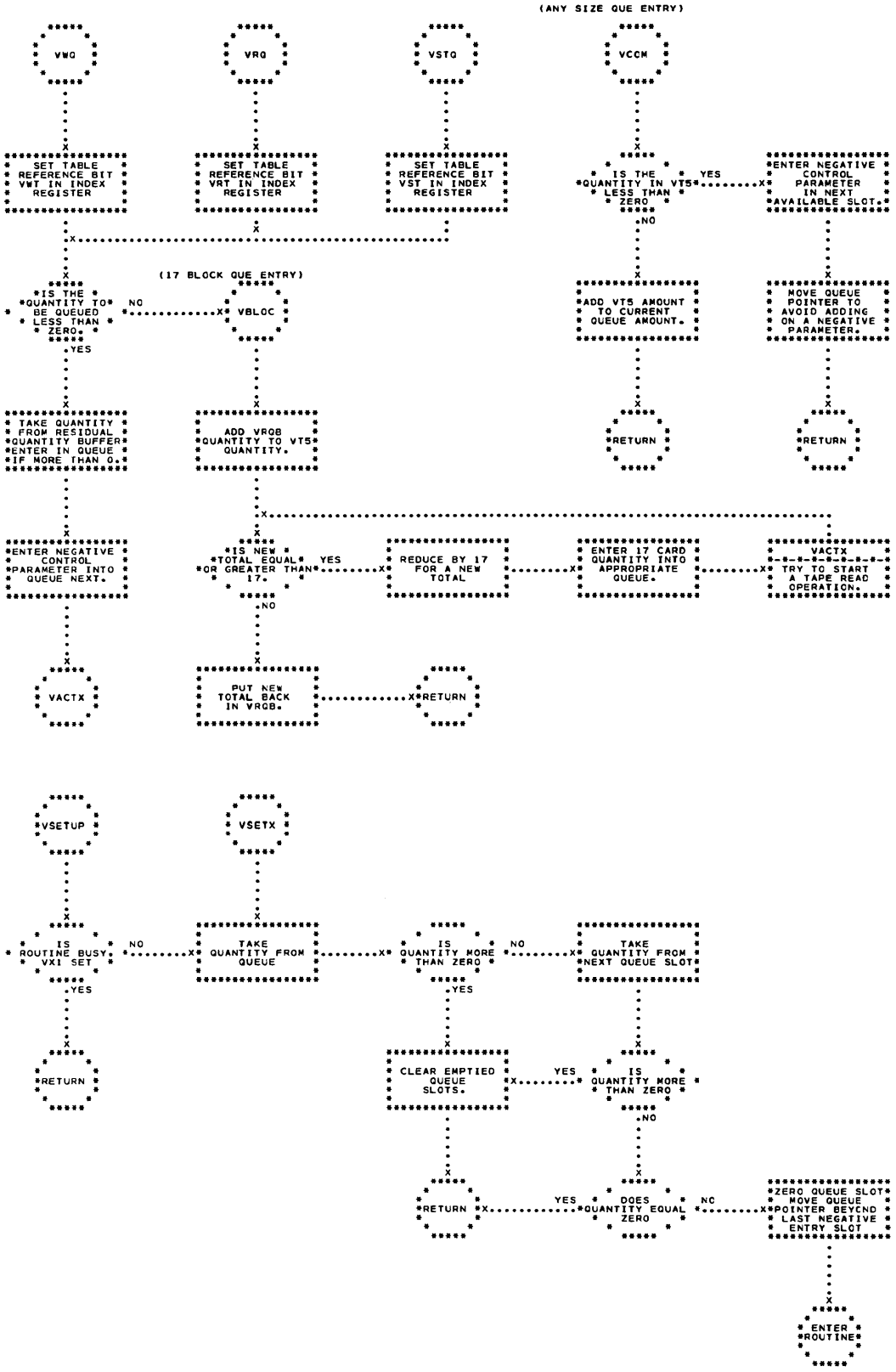


Figure 70. Input Chart 1 - Queue Manipulation

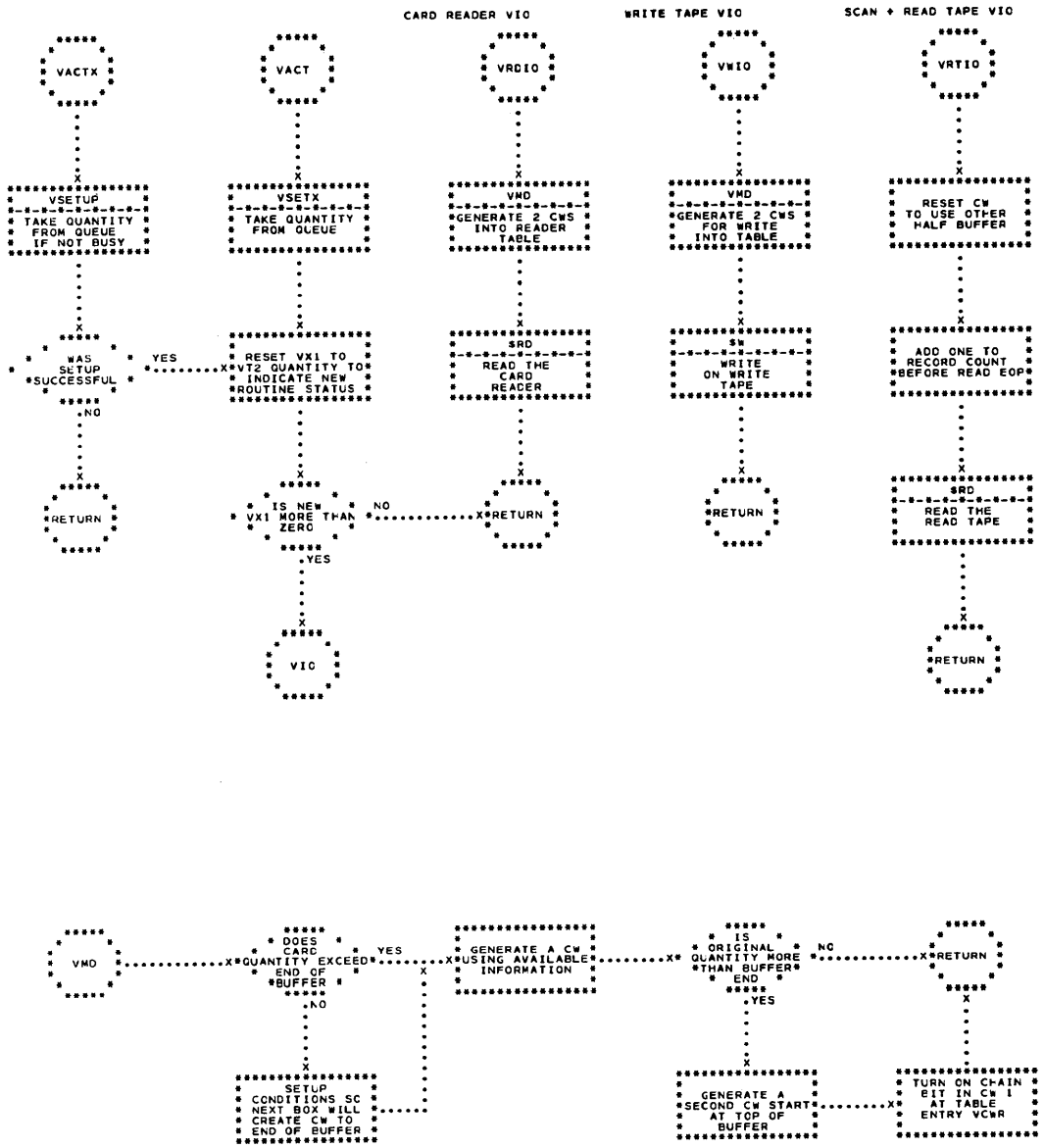


Figure 71. Input Chart 2 - I-O Actuation

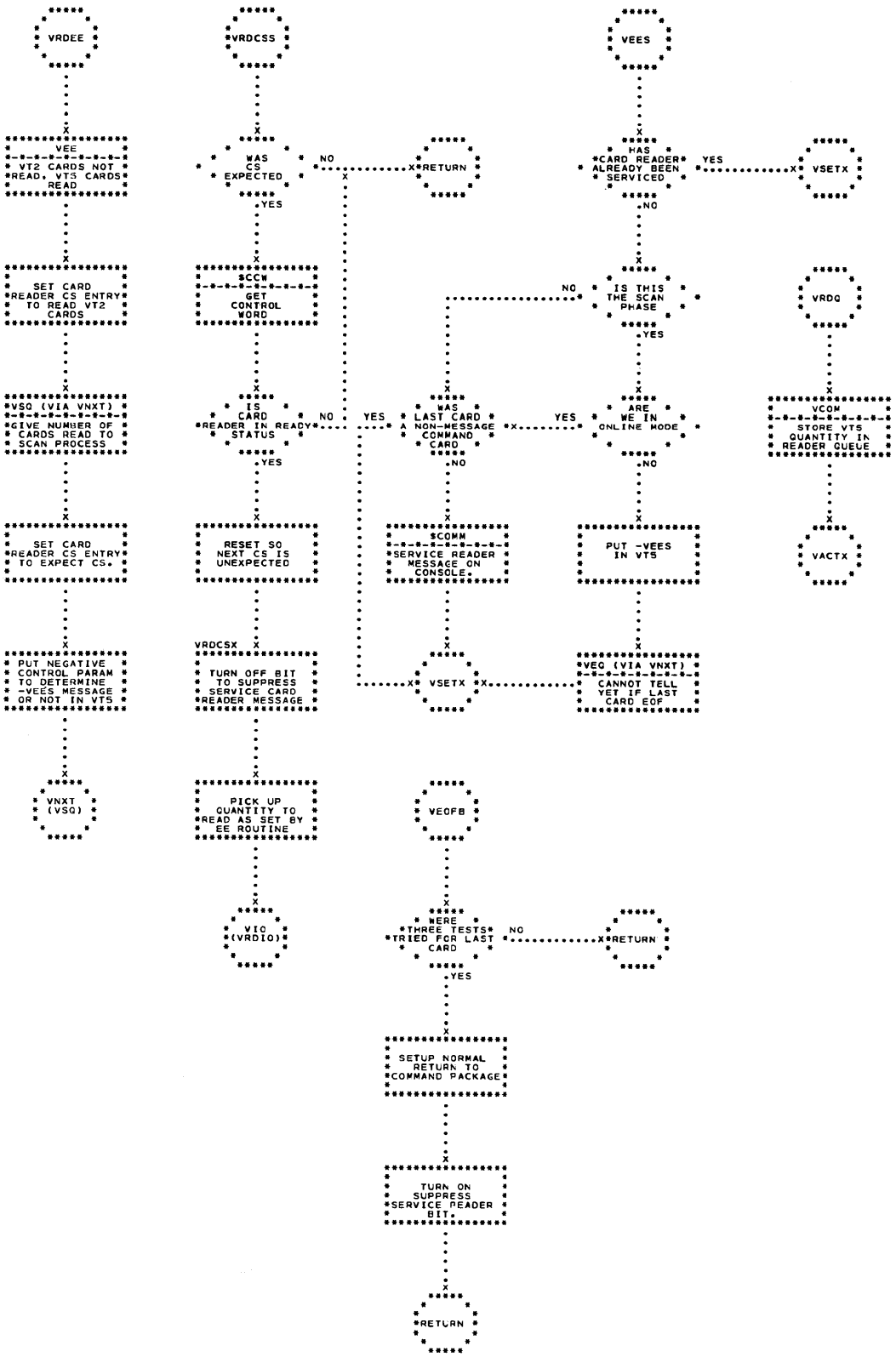


Figure 72. Input Chart 3 - Card Reader Program

The Job Boundary Scan (SC) Program

The scan routine (Figures 73, 74) scans the column 1 positions of cards in the buffer for job boundaries which are indicated by the presence of a B, JOB or B, COMD card following a non-B card. Cards up to and including the non-B belong to the previous job, and a positive parameter representing cards up to that point is passed on to the successor program: card request (bypass), write tape, or scan tape. A negative parameter follows in order to establish a definite break, since otherwise subsequent positive parameters would add into the parameter representing the previous job in successor's queue, thereby losing the identity of the jobs.

After the job boundary, SC action depends on whether the mode of operation is bypass or overlapped. If bypass, SC scans through the B cards until a non-B card is encountered, at which point scanning to a B card is resumed to identify the next job boundary. If overlapped, SC primes JC1, enters the SCR1 mode and returns, doing no further scanning until a SCAN pseudo-op is received. JC1 gives an SCR1 command in order to get the actual address of the next card in the buffer. (This enables JC1 to look at certain cards for control purposes while they are still in transit to their ultimate destination.) JC1 may continue to issue the SCR1 command until SC has no more cards in its buffer, and the end return is given. However, receipt of more cards by SC while in the SCR1 mode will cause it to again prime an entry into JC1.

When JC1 finds the first non-B card following a B card, or rejects a job, it issues the SCAN pseudo-op which causes SC to resume scanning until it finds another B card. As each SCR1 command except the first is executed, the parameter 1, representing the previous card, is passed on to the successor program, and VX is updated. By retaining custody of the card last introduced to JC1, SC prevents that buffer area from being reused prematurely. It also takes into account the possibility that the disposition of such a card may be affected by its own contents. For example, if the card contained a BYPASS command, it would have to go to E rather than WT.

The specific negative parameter passed on by SC at job boundary time depends on the mode of operation in force. If offline, then no parameter is passed on to the scan tape program inasmuch as the job boundary condition has no phase 1 significance in this mode. In the online mode, SC passes the parameter -VWEFT to the WT program, which will cause a tape mark to be written on the write tape when it is removed from the WT queue. The parameter, -VNOP, is passed to E in the bypass mode. The instruction at symbolic location VSC5 determines which action is to be taken.

The SCAN pseudo-op and SC mechanism operate slightly different in the offline mode than in the online mode. The online mode must scan every card to determine job boundaries, whereas, the offline mode has a tape mark to convey this information. On a SCAN request, in the offline mode, a $SPFL is issued to the tape if normal reading does not encounter the tape mark (as might be the case for a very short job).

The Write Tape (WT) Program

The WT program (Figure 75) writes card images onto the write tape in a blocked format, separating jobs with tape marks. The blocks within a job consist of 17 cards each, except for the last block of a job, which may contain fewer cards. The number of jobs written on a tape is limited by a tape switch criterion. When this criterion is satisfied, an extra tape mark is written, making two consecutive tape marks, and the tape switch routine is entered.

The EOP Entry: The WT end of operation routine, VWEOP, updates an in-job record count and then gives control to the subroutine, VEOP, which completes the standard EOP procedure. The in-job record count is maintained in the WT table at VRCNT.

The Queue Entry: The queue entry routine, VWQ, maintains the queue in a manner different from the VCOM subroutine due to the blocking function which WT performs. The subroutine VBLOC is used to control the entries.

When the negative parameter, -VWEFT, representing the end of a job, is given to WT, the contents of VRQB, if not zero, are placed into the queue and VRQB is set to zero. Then the negative parameter is inserted into the queue. Following this procedure of entering parameters, the parameters removed by VSETX will be a succession of, (1) an arbitrary number of 17's, (2) a single parameter between 1 and 17 (inclusive); and (3) the negative parameter. After the incoming parameter has been entered into the queue as described, control is transferred to VACTX in order to actuate the write tape if it is not then active.

The End of Job Routine: The end of job routine is entered when VSETX removes the negative parameter, -VWEFT, from the queue in the normal course of WT processing. Accordingly, control goes to the subroutine VWEFT (Figure 75), which (1) sets VX1 non-zero, making WT busy; (2) zeros the in-job record count, VRCNT; (3) alters the table of exits of the symbolic tape unit currently acting as the write tape so that control will be returned to the write tape mark EOP procedure, VWFEOP; and then, (4) issues the $WEF pseudo-op.

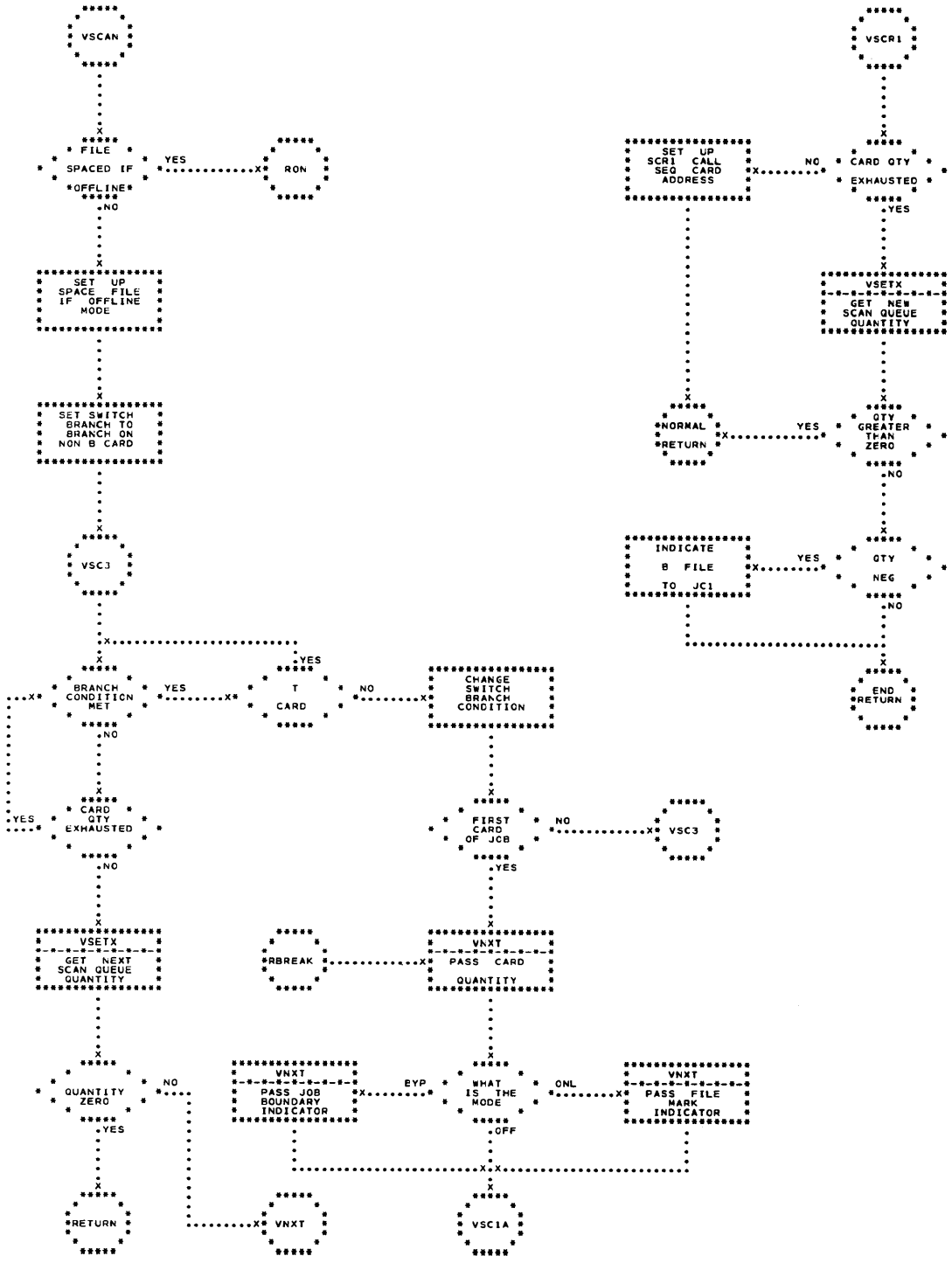


Figure 73. Input Chart 4 - Scan and SCR1 Programs

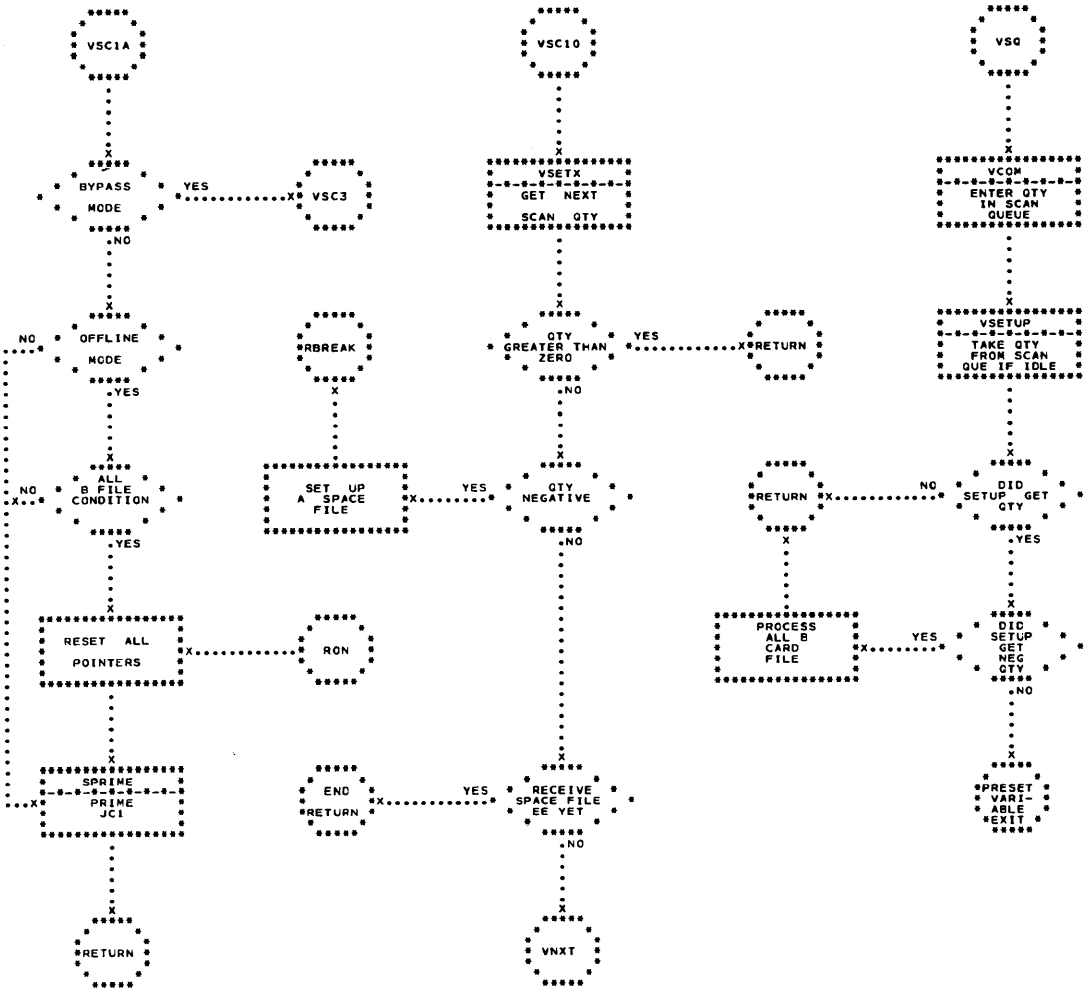


Figure 74. Input Chart 5 - Scanning Subroutines

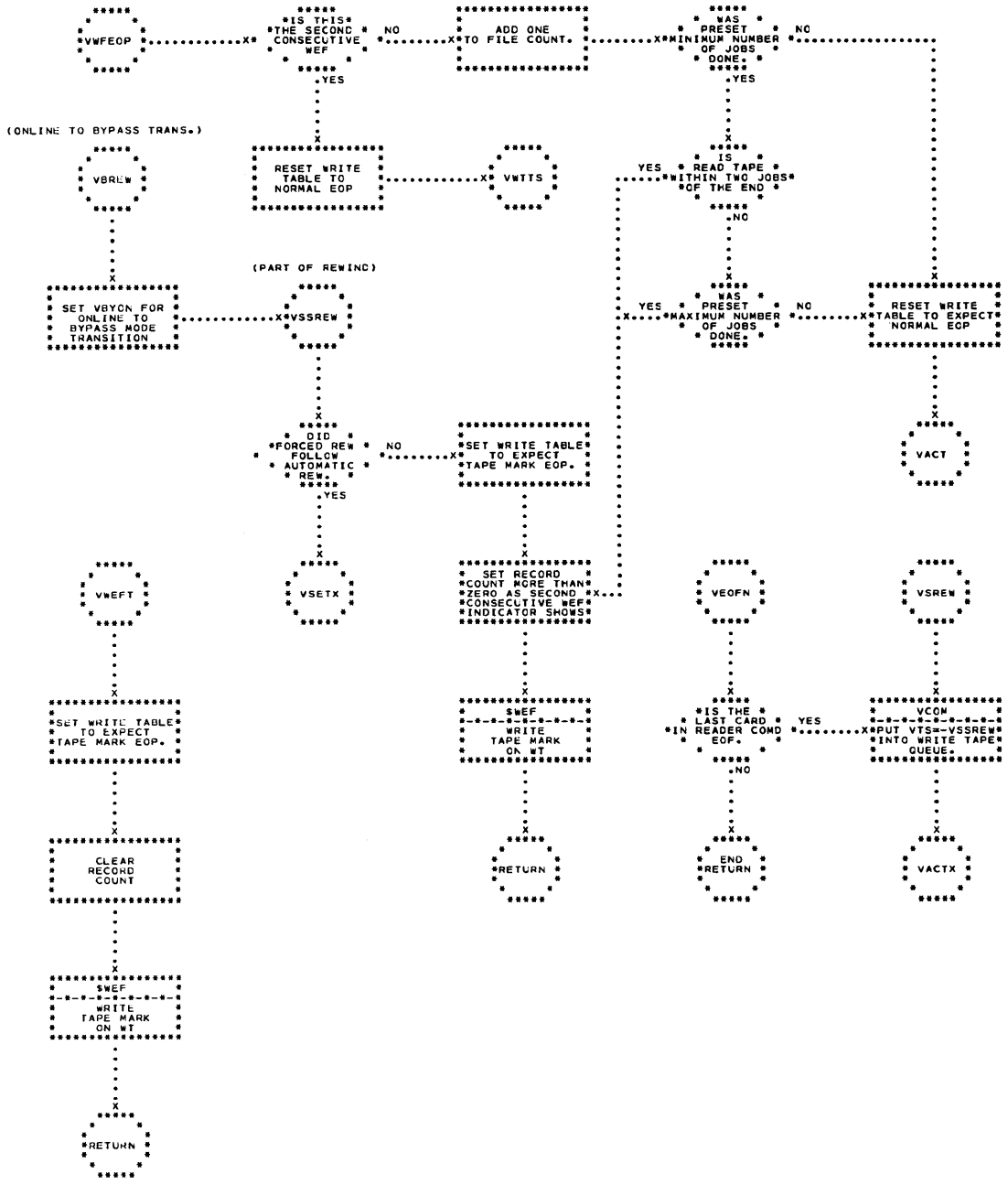


Figure 75. Input Chart 6 - Write Tape Program

Upon completion of the $WEF, control passes to VWFEOP (Figure 75) which determines, from the zero record count, that this is a job-terminating end of file. After updating the file count, VFCNT, the tape switch criteria are tested and in the usual case when none are satisfied, the table of exits is restored to normal and parameter flow resumed by going to VACT.

The Tape Switch Criteria and Procedure: The tape switch procedure consists of writing a tape terminating file mark, and, upon completion of the operation, restoring the table of exits to normal and going to the WT tape switch routine, VWTTS. Since VX1 is still non-zero, all write tape activity is blocked until restarted by an outside stimulus, namely the tape switch routine.

The tape switch (TS) criteria are tested (VWFEOP), immediately after each job-terminating file mark has been written. Whenever the TS procedure is used in these circumstances, the result will be two consecutive tape marks.

The TS criteria for terminating a tape are designed to meet two conflicting objectives: first, to start write tape rewinding sufficiently in advance of exhausting the read tape program to maintain an uninterrupted supply of data, and second, to put sufficient information on each write tape so that rewinding time does not become a large negative factor in overall performance. The first criterion is that a tape is not terminated until a specified minimum number of jobs (VMNJB) have been written on it, and the second is that it is always terminated when a specified maximum number of jobs (VMXJB) have been written. Between these extremes, in addition, the tape is terminated after the read tape has VFCD or fewer jobs left to read. VFCD is a parameter defined by a SYN card. At present, VFCD is 2, VMXJB contains 10, and VMNJB contains 5.

The Read Tape (RT) Program

RT reads cards from tape into phase 4 buffers, identifying the job boundaries to the request processor (Q) as they occur. It also indicates when it has read all but VFCD jobs on a tape (part of the WT tape switch criteria). The RT routine consists of a queue entry, VRQ; the I-O fixup routines, VREOP and VREE; and the I-O actuation routine, VRTIO. The queue entry is identical to that of WT since block recording must correspond to the blocked format of the write or scan tape.

EOP and I-O Actuation: The EOP procedure, performed by the VEOP subroutine, is standard. The actuation routine, VRTIO, derives the next control word, updates the record count (RCNT), and issues the $RD pseudo-op. The control word is obtained by refilling the present control word, R, VCWR(VU), which in turn alternately selects one of the two control words referring to the top and bottom halves of the phase 4 buffer. It would seem that the control words bear no relation to the VX parameter, but actually the parameter flow is controlled so that only positive parameters having the value 17 are inserted in the RT queue. Since this is so, the VMD subroutine need not be used.

Since RT control words have the multiple bit on, the reading of short records is terminated with an EE interrupt caused by detecting the tape mark which follows short records. Thus an EOP termination occurs only when a full block (17 cards) has been read.

The EE Fixup: The EE fixup (VREE, Figure 76) will be entered N+1 times if there are N jobs on the read tape. There are N entries corresponding to tape marks terminating jobs and the N+1th entry for the tape mark terminating the tape. The RT program distinguishes the last entry by testing whether any cards were read subsequent to the last EE interrupt.

VREE first decrements a count in VFCNX of the RT table, and when the count reaches zero, sets up VTSQ, which influences when WT starts tape switch. Initially, VFCNX is set to N-VFCD for each new read tape received.

To establish the significance of the tape mark the VTEE subroutine is entered which starts by making two tests:

1. Was this the first read instruction in a new job?
2. Were there no cards read by this instruction?

Affirmative answers to both mean the end of tape has been reached, and the procedure discussed in the next section is followed. Otherwise, VTEE proceeds with bookkeeping functions:

1. It updates the VX parameter to the next half buffer.
2. It sets VRCNT to zero and updates the file count, VFCNT.
3. It enters VACT, in an attempt to reactuate the tape.
4. It passes a parameter to the successor program giving the number of cards successfully read.
5. It returns to the calling program (in this case VREE), which passes a negative parameter, VQEND, signifying job end. This is followed by a positive parameter which is the number of cards not read

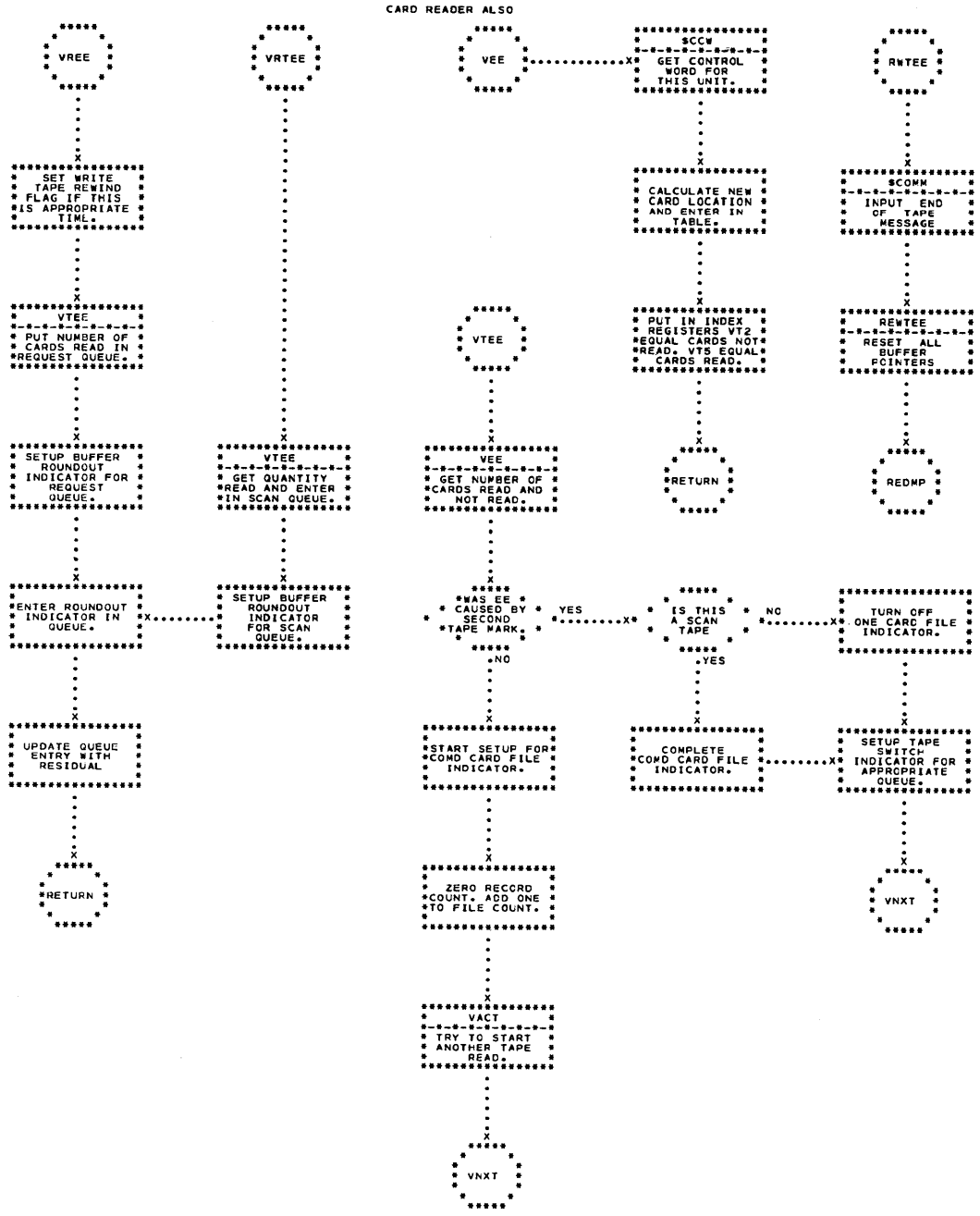


Figure 76. Input Chart 7 - Tape EE Fixup

because of the tape mark. This completes the job end procedure within RT.

After the procedure above the queue of the card request program (overlapped) would contain K, -VQEND, 17-K if the number of cards read successfully was K. The parameter K will become an active parameter in Q making K cards available for transmission. When this is exhausted, -VQEND will be removed from the queue and control will go to QEND, which will remove the next parameter, 17-K from the queue and pass it back to RT at the same time updating VX(Q). Thus the parametric representation of one entire buffer half will have passed through Q in proper sequence though the parameter division provides that only a specified portion will be made available for transmission. Finally, VQEND returns to the program which called VSETX with a negative parameter, which is recognized by the request processor as a job boundary.

The Tape End Procedure: The tape end procedure, culminating in tape switch, is initiated from the VTEE routine when the tape switch criteria are met. The routine, VSTTSX, simply sends to the successor program a negative parameter which it finds in the VTS slot of the program table. This parameter, -VRTTS, causes an entry to VRTTS to occur only after the last job read from the read tape has been completed with an SEJSCN command given to the input program.

The Scan Tape (ST) Program

The scan tape program is virtually identical to the read tape program. This is because both perform an almost identical function, namely; (1) to supply a buffer with card images read from a tape of standard format; (2) pass on parameters representing the buffer areas to a successor program, identifying to the successor the occurrence of each job boundary as it is detected; (3) to recognize the logical end of tape; and (4) initiate the tape switch procedure. Whatever differences exist are absorbed in the tables, such as operating buffer (VCWR), tape switch routine, symbolic I-O unit (VSU) and successor program (VNX).

The job end procedure, however, differs in two points:

1. The count down of jobs and communication to the WT for tape switch purposes is omitted.
2. Instead of VQEND, the negative parameter, -VSCEND, is used for parameter partitioning of incompletely read buffer areas. In the offline mode, SC is adjusted so as not to feed any negative parameters to its successor at job boundaries, since ST, in effect, already knows of job boundaries.

The Request Processors (E and Q)

The card request processors provide cards under control of externally generated requests, with the process governed by the parameter flow mechanism. The input program is entered at VSCR when $SCR or SSCR4 is requested (Figure 77).

The queue entry routines perform the standard functions except that the actuation phase consists simply of placing a non-zero parameter into VX1. That amount of cards is then transmitted under the control of and to memory areas designated by one or more externally generated requests. The equivalent of EOP is not so clear. Recall that normal EOP procedure consists of passing on the contents of VX1, updating VX, and reactuating. The request processors perform the first two functions on a piecemeal basis depending on the occurrence of requests. In other words, for every transmission, VX1 is decremented and VX incremented by the amount of cards just transmitted, and a parameter representing this amount is forwarded to the successor program. However, when VX1 is reduced to zero, reactuation occurs, consisting of obtaining the next parameter from the queue and putting it in VX1.

The Request Processor Subroutine: The work of processing requests is done by the subroutine VREQ (Figure 77). Externally, the routine has the following characteristics:

1. It is entered with the following quantities:
   a. The program table origin (E or Q) in index register VU and location VREQXT.
   b. The first word address (FWA) and number of cards (VNREQ) of the request in the SCR-SCR4 tentacle table, VVSCR.
   c. The location of the M slot of the originating calling sequence in $0.
2. When control is returned:
   a. The number of cards requested has been transmitted unless the number overlaps the end of job file, in which case only the remainder of the file was transmitted.
   b. $0 contains the location of the normal return or end return in the originating calling sequence, according to whether or not the requested number of cards was delivered.
   c. If the request was terminated due to file end, the number of cards actually transmitted is placed in the calling sequence at the M slot.

In its internal operation, VREQ first tests VX1 to see if it is greater than zero. If it is not, control goes to VREQ2 for further tests described later. If VX1 is greater than zero, the VMD routine is entered to construct the control words for transmission, the

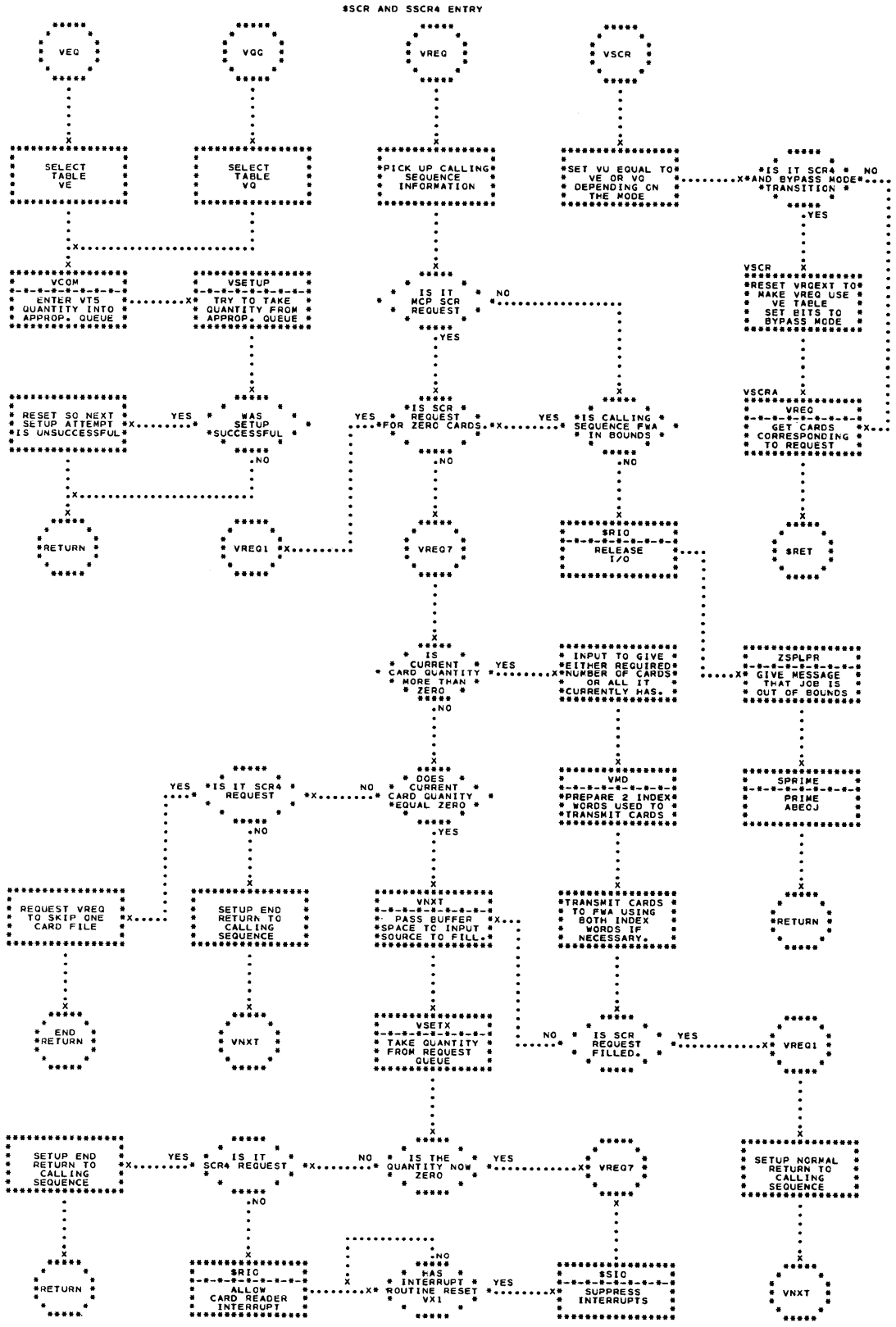


Figure 77. Input Chart 8 - Card Request Processors

number of cards transmitted being governed by the lesser of VX1 or VNREQ. After transmission, VX1 and VNREQ are decremented by the lesser amount. The VNREQ residue is tested for zero, meaning the request processing is complete, and if it is, VREQ returns control with $0 properly adjusted. On the way back, the parameter representing the last transmission is passed on to the successor. If the request processing is not complete, control goes to VREQ10 to get, if possible, another positive parameter from the queue. VREQ10 first forwards to the successor a parameter representing the number of cards just transmitted, and then goes to VSETX. The value returned from the queue by VSETX is stored in VX1. If it is non-zero, the above process is repeated. If it is zero, the action depends on whether the original request was a SCR or SCR4. If SCR4, $0 is adjusted for an end return and VREQ returns control. If SCR, the $RIO pseudo-op is given to allow interrupts from the read tape or card reader. The first parameter which is passed on to Q or E as a result of an interrupt will be placed into VX1 by the normal queue entry function since VX1 is now zero. Consequently, after giving $RIO, a loop is started which tests VX1 repeatedly until a non-zero condition is identified. $SIO is then issued to protect the request processor from uncontrolled interference from future interrupts, and the entire procedure is repeated from the beginning.

If, back at the beginning, VX1 was not greater than zero, control was sent to VREQ2 which first tests whether VX1 is zero, in which case VREQ10 is entered to obtain another parameter. Should VX1 be negative, the request is up against the job end and the end return procedure is applied. This consists of deriving the number of cards actually delivered by subtracting the residual from the original request, storing the result in the M slot of the calling sequence, and adjusting $0. However, VX1 itself remains unchanged in case it is negative, to act as a block to all future requests.

The request routine includes logic to simulate an SEJSCN operation when a one card file is encountered on a read tape. This situation may be originated by an offline input program, and appears to be a one card job consisting of a command card. Unless the situation is recognized and SEJSCN performed, the system would hang up in phase 4 in an SSCR4 - end return sequence.

The SEJSCN Command: SEJSCN is an input command to recognize the next SCR4 request as a request for the first card of the next job. The first stage in the execution of this command, in either overlapped (VOEOJ) or bypass (VBEOJ) mode, is done by the VSEOJ subroutine (Figure 78). This routine looks for a negative parameter, beginning at VX1 and proceeding into the queue, removing successive parameters by means of VSETX. It continues until either a negative parameter is returned by VSETX or the end of the queue is reached, at which time the sum total of the positive parameters passed over in the search is passed on to the successor and VX is adjusted to preserve phase. Control returns from VSEOJ with either the negative parameter or zero, if none was found.

When VSEOJ returns control, it tells whether or not the job end is in the buffer, and it has swept out all parameters not after such job end. If the job end was in the buffer, the command SEJSCN has been completed, since the removal of the negative parameter unblocks the VREQ routine. If the job end was not in the buffer, further action is required to set up a mechanism to skip over the remainder of the job at the source, either the read tape or the card reader.

To consider the bypass case first (E), the queue entry routine is modified so that rather than inserting parameters in the queue, it tests them for equality with -VNOP. If the incoming parameter is positive it forwards it to CR after adjusting VX. By shunting all of the parameters related to the defunct job back to CR, a continuous scan is set up, to terminate only with the end of job signalled by the parameter, -VNOP. When this occurs, the interface is restored to its normal state and the execution of the SEJSCN command is complete.

In the overlapped situation (Q), the remainder of the file is skipped, including the file mark, and normal reading resumed by RT. At this time the read tape must be busy with a $RD instruction because having just evacuated the queue of Q with VSEOJ, RT has a full complement of 34 card images and must certainly be reading into 17 of them. The skipping procedure consists first of changing the table of exits mechanism so that control comes to one of two special fixups, VSREO and VSRE, when the $RD is terminated. If an EE occurs, which is ultimately desired, VSRE is entered. VSRE restores the normal table of exits configuration and restarts the same $RD instruction but this time the record coming in will be the first of the next job, and the normal fixups, just restored, will forward it to the request processor. If an EOP occurs, a $SPFL command is issued by VSREO and the resulting EE interrupt will cause VSRE to be entered later.

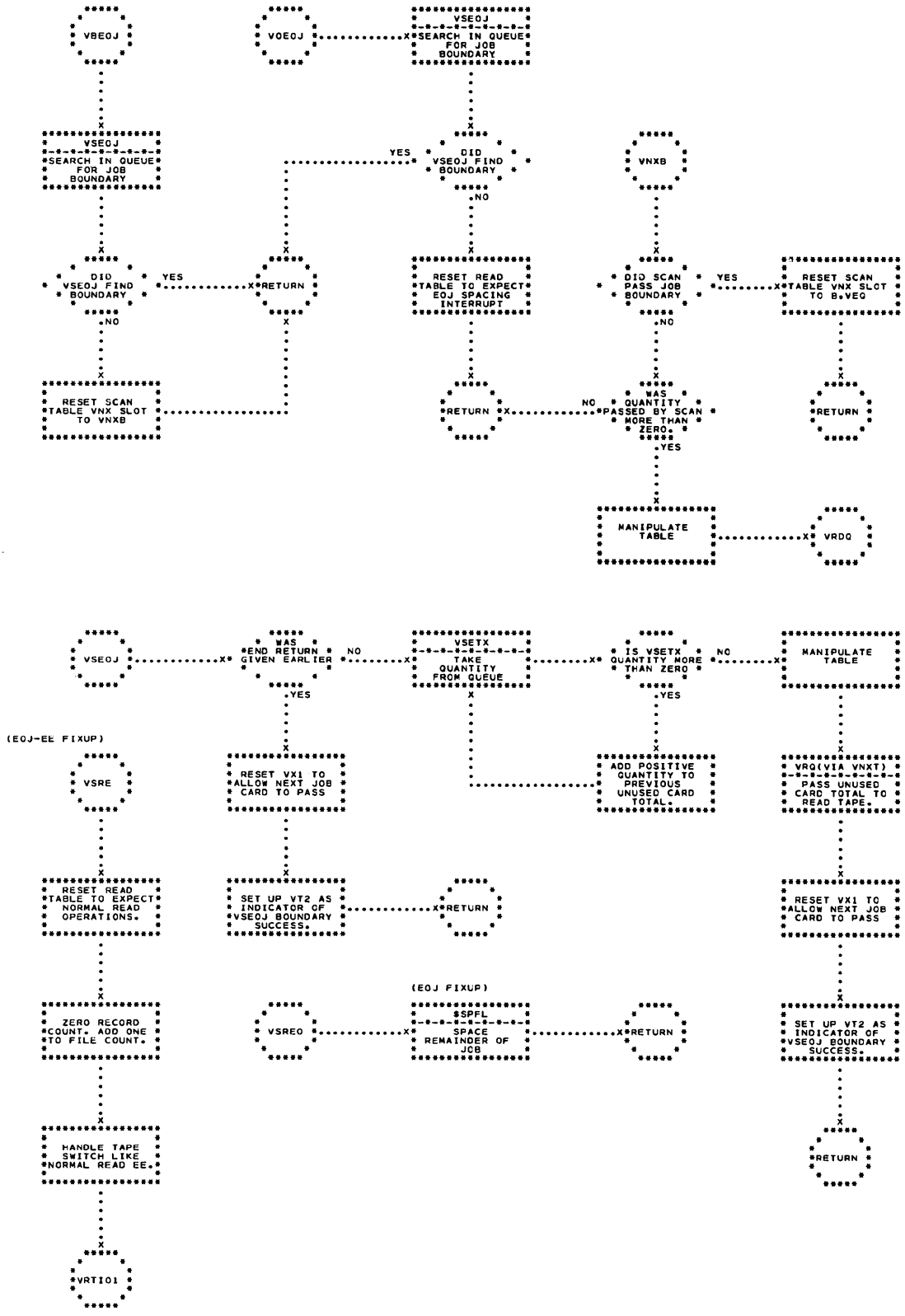


Figure 78. Input Chart 9 - EOJ Scan

In the above situations, while a protracted file skipping operation is in progress, VX1 remains zero. Should a request for cards occur, it will hang up in a loop with JC4 via the end return in the SCR4 calling sequence, since SCR4 gets an end return if there are no cards immediately available.

The Tape Switch Program

The tape switch program consists of two routines, which resemble the functional programs in their overall characteristics. One, VTSA, accepts write tapes and scan tapes from phase 1 operations and feeds them, in sequence, to phase 4. The other, VTSB, takes the units given up by phase 4 and makes them available for phase 1 use.

The two TS routines are patterned around a parameter flow mechanism similar to that of the functional programs, in which the parameters are the IOD numbers of the tape units used by the input program. The routines each have a table of similar format to that of the functional programs, as shown below.

Symbol	Table Position	Content
VX	0	Unused
VX1	.32	Active parameter
VX2S	1.	Queue maintenance parameter
VX2R	2.	Queue maintenance parameter
VOP	3.	Bit specifies REW or UNLD
VSQNC	3.1	Bit sequence critical
VDS	3.32	Program destination of tape
	4.	Constant
VNX	5.	Successor program
VIO	5.32	Tape initiation routine

The first four entries are concerned with the parameter flow mechanism which is similar to that of the functional programs with the following differences:

1. VX, though "updated" by the VACT subroutine has no significance and is never used.
2. VX1 will contain the IOD number of the tape currently active in the respective phase.
3. The identity of incoming positive parameters is maintained in the queue by placing them in consecutive slots rather than accumulating them in the same slot. Therefore, VCOM is not used in the TS routines.

The VOP bit specifies whether the proper operation to terminate the tape is $REW or $UNLD. For phase 1 (TSA) this is always 0 indicating $REW, whereas the setting in VTSB depends on which overlapped mode is in effect or, if the bypass mode is current, was last in effect.

The VSQNC bit specifies whether the sequence in which tapes are released from a phase determines the sequence in which they are made available to the other phase. In VTSA, this bit is always 1, indicating the critical sequence case which arises because of the necessity of maintaining phase integrity with the tables constructed during the phase 1 scan. Since the released phase 4 tapes carry no useful information, their sequence of return to phase 1 is not significant. Another word, VTQ is also used in the mechanism for preserving sequence in transferring tapes from phase 1 to phase 4.

The VDS slot contains the address of the table of the functional program which is the ultimate destination of tapes being retired from a particular phase. For VTSA this is always RT whereas for VTSB this may be ST or WT depending on the overlapped mode. As each IOD number is forwarded to its successor, the intended destination, derived from the VDS slot, is assigned to the unit.

VNX contains a branch to the queue entry routine of the successor, the entry points of which are VTSAQ and VTSBQ for VTSA and VTSB respectively. VIO contains a branch to a routine, VTA, which initiates an extended operation during which VX1 is non-zero, indicating the TS routine is busy. The operation referred to is that of attachment of a specific unit to one of the two phases. This operation continues while the phase processes the tape and is terminated when the phase transfers control to one of the tape switch initiation routines.

The action of tape attachment consists mainly of placing the IOD number in the VSU slot in the tape program table and branching to the VSTR slot in that table. VSTR contains a branch to a small routine which restarts the tape program, which had blocked itself pending assignment of another unit.

The Tape Switch Initiation Routines

The TS initiation routines are the routines which initiate the action of transferring a tape to the other phase. They are some what analogous to the table of exits for the functional programs through which control returns at the end of an operation, the operation being continued attachment of a tape unit to a tape functional program. The three routines and the tape programs they are related to are: VWTTS(WT), VSTTS(ST), and VRTTS(RT). See Figure 79.

All three routines load index register VU with the origin of the TS table appropriate to the phase (VTSB for phase 4, VTSA for phase 1) and call the VTOP routine, which is the analogue of the VEOP routine for functional programs. VWTTS and VSTTS also provide VTOP with the count used by RT in deciding when to request tape switch from phase 1.

VRTTS and VSTTS, which are called as the result of removing the negative parameter from the queue

of Q or SC, return a zero parameter to Q or SC after calling VTOP. Thus, Q and SC are starved and their operation ceases. Meanwhile, RT and ST are blocked by the parameter 17 in VX1 which represents the $RD that could not be completed because the end of tape was reached. When a new tape is assigned, the processing is resumed simply by reissuing this $RD.

Operation in the WT program would be inhibited by some negative parameter in VX1, usually WEFT, and when the time comes to resume, this would be done by using the VACT subroutine.

Once a TS initiation routine has called VTOP, the tape switch routine proceeds independently of the functional programs and their tables until it is time to actually attach a tape to a tape program.

The Tape Switch Routines

The TS routines (Figure 80) proper are a single set of routines which serve both VTSA and VTSB. Therefore, rather than discuss VTSA and VTSB separately, the routines of which they are both composed will be discussed.

As the tape units proceed through the successive stages of tape switch certain information must be carried over between stages. For this purpose a single full word is set aside for each IOD number. These are arranged in a symbolic unit reference table (VSURT), so that they may be directly accessed with the IOD number. For the tape IOD's the format of an entry is as follows:

Format

XW, VF, CF, RF, Flags

Content

Symbol	Position	Use
'vf'	VF	Used for maintaining sequence information for VTSA.
'cf'	CF	Contains file count for use by RT.
'rf'	RF	Contains TS routine table origin while a $REW or a $UNLD is in progress.
'rf'	RF	Contains the destination of this unit after $REW or $UNLD is completed (i.e., RT, WT, or ST.)
VOPB	Flag Bit .25	Distinguishes $REW and $UNLD.
VRDY	Flag Bit .26	Used to indicate unit ready.

The two main routines in TS are VTOP, which initiates a $REW or $UNLD; and VTDSP, which is entered when a channel signal interrupt is released.

The Tape Operation Routine (VTOP): The operation of VTOP is governed by the appropriate table, VTSA or VTSB, the origin of which is provided in index register VU by the TS initiating routine which called VTOP. The IOD reference number is obtained from VX1 (VTSX)* locating an entry in VSURT. The VOP (VTSX) bit determines whether a $REW or $UNLD operation is given and this setting is placed in VOPB (VSURT) where it is the basis of the operation to be performed. The VSQNC (VTSX) bit is tested and if 1, information relating to the order of rewinding must be preserved. This is done by using a linking mechanism in which each VSURT entry involved points to the next VSURT entry in the queue. The 'vf' portion of VSURT is used for this purpose and the relative location of the queue is in VTQ(VTSA).

Next, the VTMV subroutine is entered, which sets the TS origin in VU into 'rf' (VSURT), sets bit VRDY (VSURT) to zero indicating unit not ready, and on the basis of VOPB (VSURT) issues either a $REW or $UNLD operation.

Control now returns to VTOP which directly branches to VACT in order to attach a new unit if one is available. The mechanism involved in this action is similar to that in which the VEOP routine, as one of its last acts, branches to VACT. VACT ends by branching to VIO (VTSX) if it finds another parameter in the queue, and since VIO (VTSX) contains B, VTA, the tape attachment procedure is initiated.

The VTDSP Routine: VTDSP (Figure 80) is entered from the CS slot in the table of exits with the IOD number set into index register VTS. First the VRDY (VSURT) bit is set to 1 and then using the 'rf' entry in VSURT the applicable TS table is located. The 'rf's (VSURT) slot is set equal to VDS (VTSX) so that the VSURT contains now the ultimate destination program of the unit. It is significant to the transition routine VONLIN that the destination is set at unit ready time (CS) rather than at unready time.

The VSQNC (VTSX) bit is then tested to see whether sequence of actuation is to have a bearing on the sequencing of forwarding to the other TS routine. Should it not, then this is the phase 4 to phase 1 interface and a special compatibility test, VTDP1, must be made before forwarding can occur. Otherwise, the linked queue of rewinding tapes is scanned, using VTQ (VTSA) to locate the front of the queue, until a unit ready bit (VRDY) is found to be zero or the end of the queue is reached. As the queue is scanned all the tapes with VRDY (VSURT) equal to 1 are removed from the queue and forwarded to the successor through the standard interface.

* The VTSX indicates the preceding symbol refers to a table entry for VTSA or VTSB.

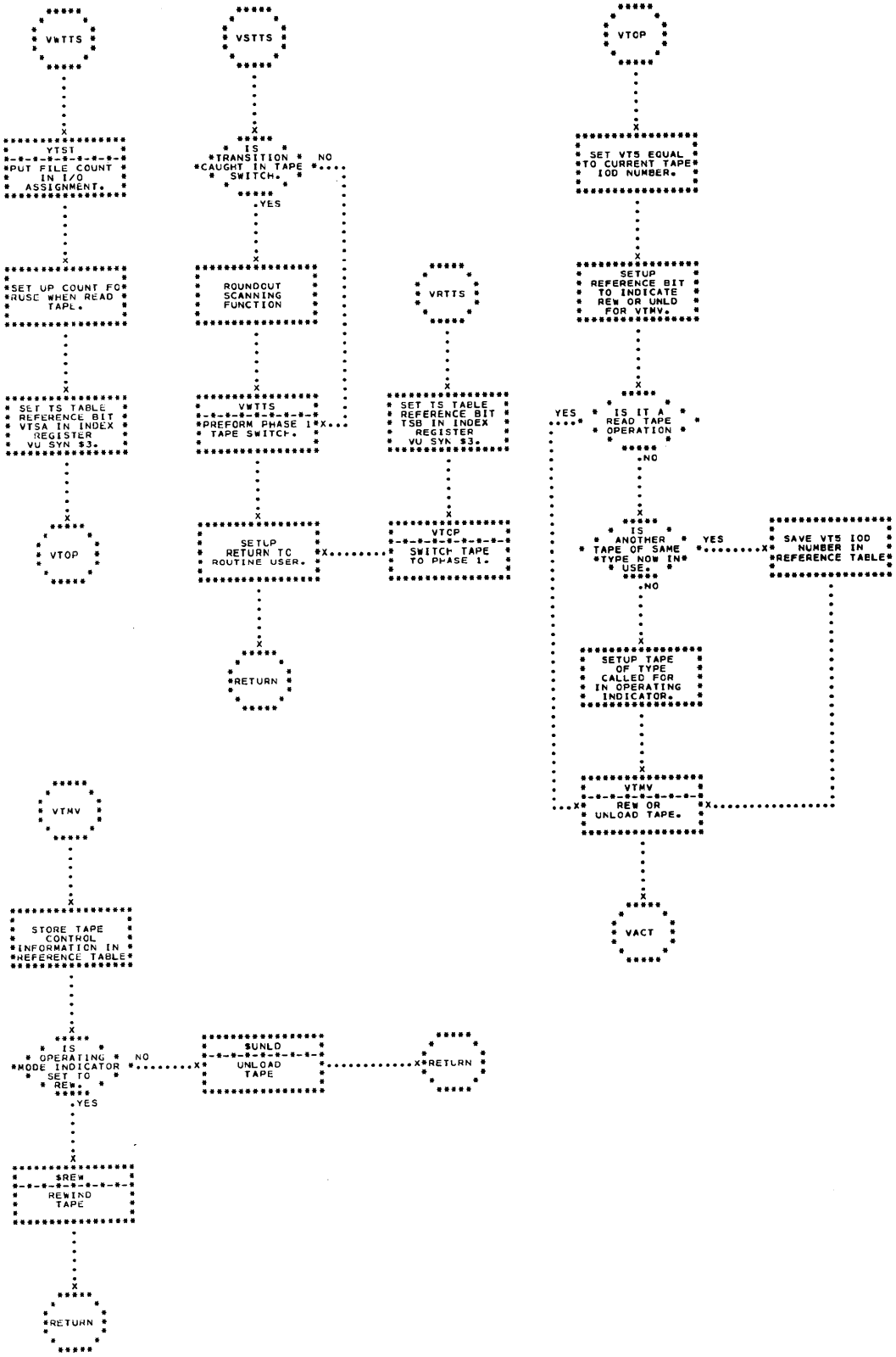


Figure 79. Input Chart 10 - Tape Switch 1

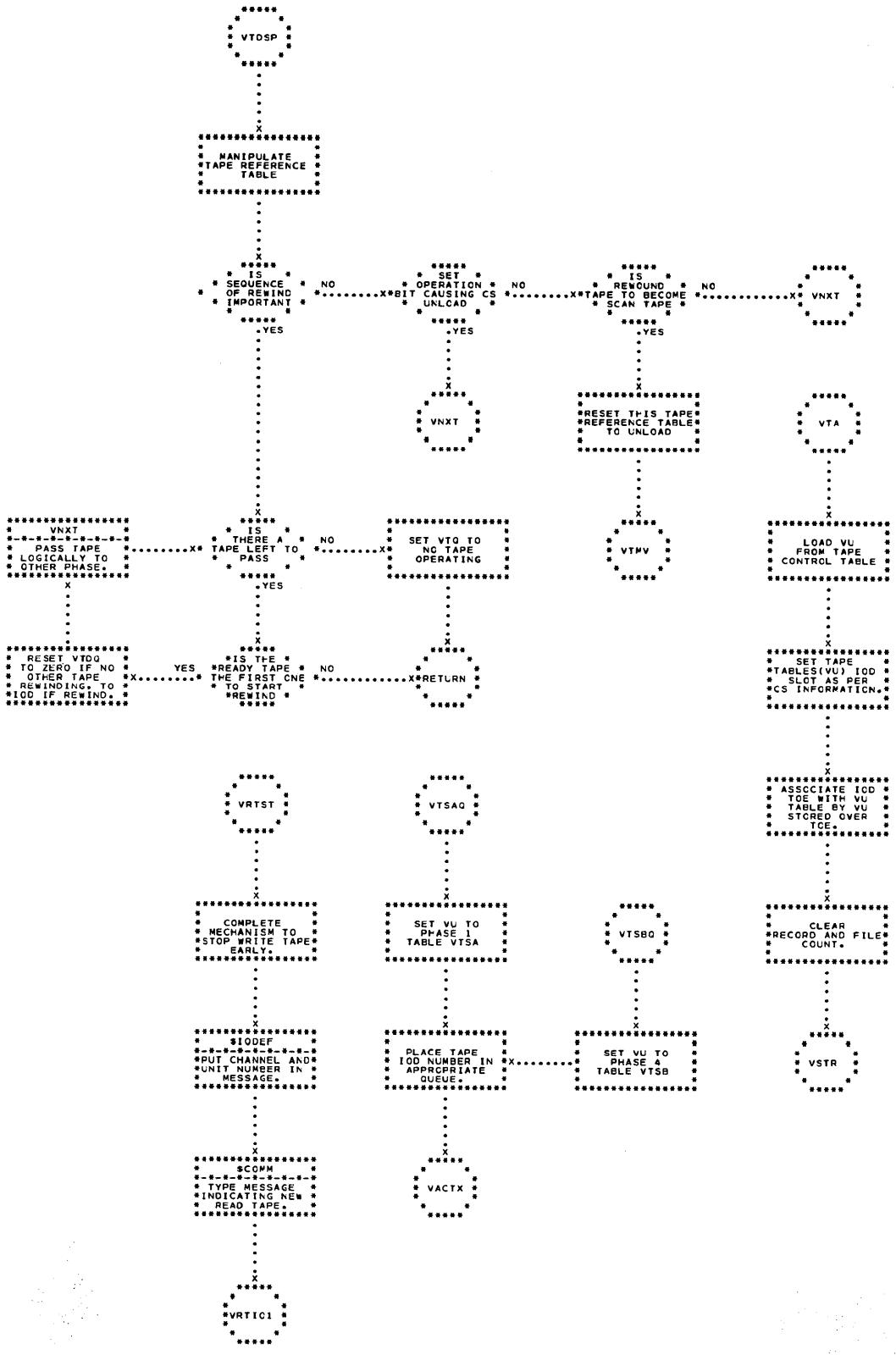


Figure 80. Input Chart 11 - Tape Switch 2

VTDP1 makes a test to see if tapes about to be forwarded to the phase 1 program, ST, have come ready as the result of an $UNLD and not a $REW. Obviously ST does not want to scan an old ready tape, which could occur if an VOFFLN transition is received after a tape is started rewinding. Should the test fail, the VOPB (VSURT) bit is set equal to 1 and the VTMV routine will be entered causing the $UNLD command to be issued. Tapes which pass compatibility are forwarded in the usual way.

VTA Routine: The tape attachment routine (Figure 80) is entered through the VIO slot in the TS tables which in turn is entered from VACT. Attaching a unit is very simple:

1. Obtain the destination program from the 'rf' slot in VSURT.
2. Store the IOD number in the VSU slot in the table of the destination program.
3. Store the program table origin above the table of exits where it will be used to control the destination of interrupt. (Refer to the code to see how this works.)
4. Set VRCNT and VFCNT equal to zero.
5. Branch to the VSTR slot in the program table to restart the program.

The Transition Routines

The six transition routines are initially entered from the input command dispatcher when the appropriate pseudo-op is given by the command package. If the requested transition cannot be immediately effected, controls are set for subsequent re-entry when the command can be effected.

Online from Offline -- VONLIN

The VONLIN routine (Figure 81) is entered when the ONLINE command is given through the console. The routine occurs at some time after the command is given because the implementation of the command must await a convenient break point in the offline mode operation. The time the command occurs is significant because all tapes subsequently readied at TSB will be online mode tapes. This is accomplished by changing VDS (VTSB) to WT immediately. Also, a negative parameter, VCCR (commence card reading) is put into the TSA queue.

After this point, tapes becoming ready at TSB will be given WT destinations and be forwarded to the queue where they will fall in behind VCCR in the TSA queue. When VTSA picks up -VCCR the online mode is actually started. This consists of 3 steps:

1. Perform bookkeeping which connects CR to SC to WT. The write tape VX is set equal to the card reader VX because an intervening bypass mode may have destroyed the phase equality of these two programs.
2. Restart the card reader by branching to a portion of the CR channel signal routine (so that the unit ready bit of the control word is not tested).
3. Re-enter VSETX to attempt to attach a tape through VTSA to WT. Phase 4 operation is not affected at all by this transition.

Online from Bypass -- VONLIB

The VONLIB routine (Figure 81) is entered as the result of an ONLINE command card having been received in response to an SCR4 command in the bypass mode. The VONLIB routine reactivates phase 4 and reroutes the cards following to WT, initiating phase 1. By the simple act of letting VREQ process subsequent requests from the Q tables, the phase 4 operation is reinstated. The rest of the transition is concerned with getting the phase 1 operation underway.

The SC routine is connected to WT, SC is modified for online operation, and the phase of WT is set equal to that of CR. At the time the VONLIB routine is entered, the E program most likely has some buffer area assigned to it, since phase 1 must obviously proceed on the assumption of no transition until the command card is actually delivered by the request processor to the package which recognizes it. It is therefore necessary to recover the remaining buffer area assigned to E and return it to SC for rescanning in the online mode. The recovery is accomplished by the VSWEEP subroutine (Figure 81) which repeatedly applies VSETX until a zero parameter is returned, totaling up all positive parameters returned and ignoring negative parameters. The total is returned to VONLIB which forwards it to SC and, most importantly, backs up to the SC phase parameter, VTX, to compensate.

The VOP (VTSB) bit is next tested to see if the previous overlapped command was online or offline. Recall that VOP (VTSB) determines whether phase 4 was unloading or rewinding tapes and that in the online mode, phase 4 would be rewinding tapes. If the previous overlapped mode was online, then the transition is complete because the tape switch mechanism was unaltered by the VBYPAN transition and is free to supply WT with a tape if, indeed, it has not already done so.

If the previous overlapped mode was different, that is, offline, the transition is deemed compound and further action is required. The VBYPAF routine has left VTSA blocked and therefore some tapes may be queued up in ST destinations. Therefore, the VTSA

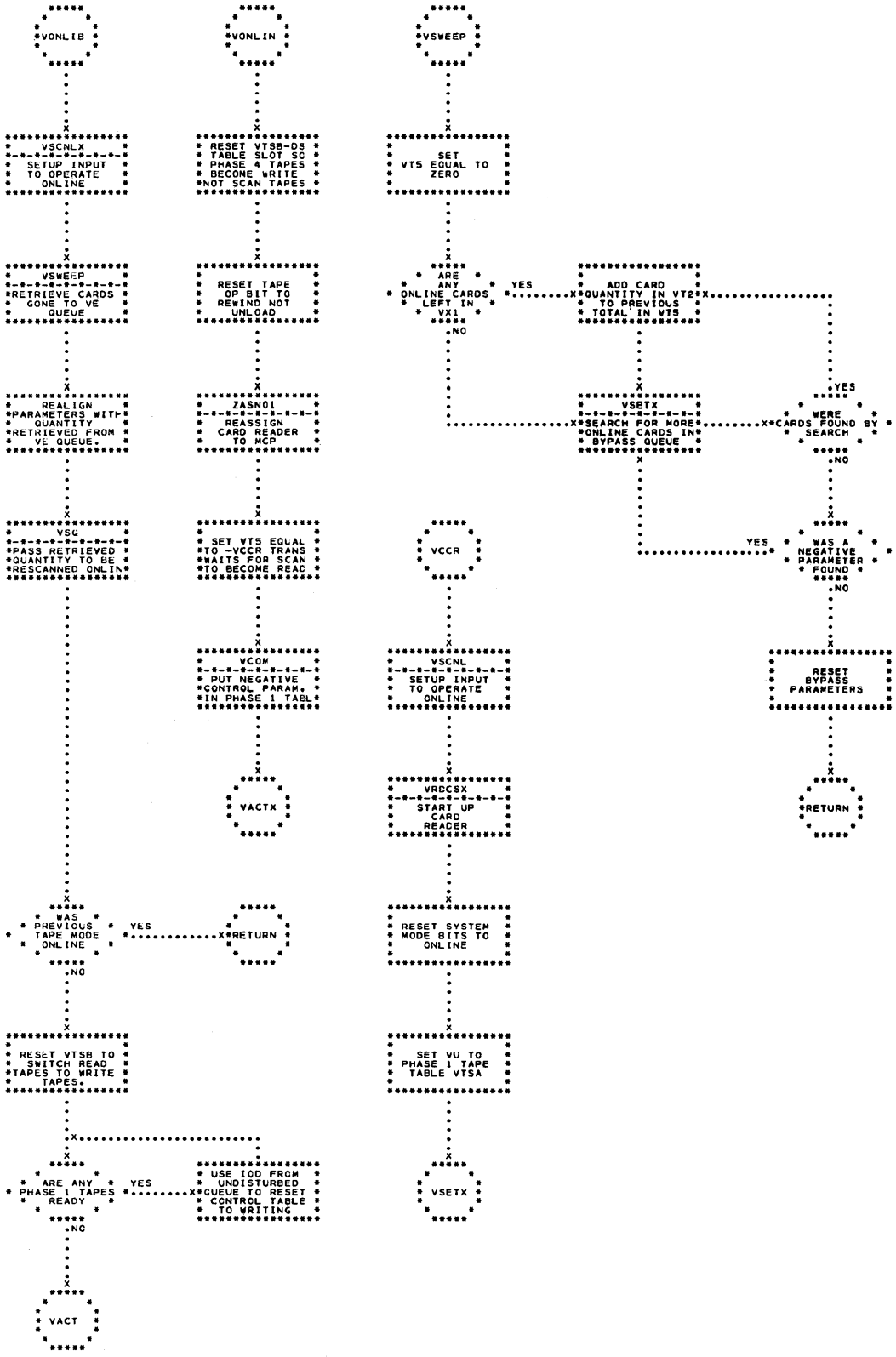


Figure 81. Input Chart 12 - Transitio

queue is scanned and the 'rf' (VSURT) entry is set to WT for every IOD number found in the TSA queue. The scanning operation does not affect the TSA queue. Then the VACT subroutine is applied to VTSA to unblock it and resume parameter flow. This completes the compound transition. However, there is one additional step, namely, VACTX is used in order to start the card reader. This step is normally redundant since in the bypass mode which had just been operating, the card reader is active and nothing has been done in the VONLIB transition to upset this. But this step is included for the benefit of the initial starting procedure, a subject covered in a later section.

Bypass from Online -- VBYPAN

The VBYPAN routine (Figure 82) receives control from the command package when a BYPASS command card is detected by JC1. The transition entails three steps:

1. Forward a negative parameter to WT to terminate the write tape.
2. Modify SC for bypass operation, connect it to E, and resume scanning as if for a SCAN command.
3. Direct the attention of the VREQ routine to the E table instead of the Q table.

The parameter -VBREW is forwarded to WT. In addition to starting the tape termination procedure, VBREW sets a bit, VBYON to 1. This setting of VBYON will cause the SCR4 routine to modify VREQ to process requests from E when the next SCR4 command is received. Thus, step 3 can only occur after all of VBF1 has been written onto tape.

After -VBREW has been forwarded, step 2 is performed by the VSCBY routine. Then the BYPASS linkage is complete and ready to process but may not do so yet because VBF1 may still have some cards which are being written or are waiting to be written on tape. Notice that if requests were processed from E, it is possible that CR would read over cards not completely written on tape or, to put it another way, E could act as a "short circuit", recycling parameters back to CR before WT could send its parameters, which logically precede those of E, back to CR. The interlock mentioned above prevents this. Once the VBYON bit has been set, the next SCR4 will set VREQ so that it processes requests under control of the E tables.

Except for this, the phase 4 operation is unaffected; parameters will line up at the Q program, which is now dormant. In this situation it is very simple to resume phase 4 operation later by directing the VREQ routine to Q.

The tape switch mechanism operates normally. It is possible for the read tape to switch over and attach to WT, but it will not actuate because by this time WT will be devoid of assigned buffer area. In this situation, the write tape just terminated would attach to RT, fill up VBF2 and then stop because of the blockage of the Q program. Observe that in this situation the resumption of the online mode would be rather straightforward.

Bypass from Offline -- VBYPAF

Like the ONLINE command, the BYPASS command is received through the console and command packages. The time it occurs is significant in that tapes coming ready at VTSA will not be scanned; their ultimate disposition depends on what overlapped mode is resumed later. Tapes already in line at TSA are scanned in order to free VBF1 for BYPASS operation.

Since the bypass mode may not be instated until VBF1 is free, no action is taken upon entering the VBYPAF routine (Figure 82) except to insert the negative parameter, -VCBR, into the VTSA queue. VCBR, being entered only after all tapes in the VTSA queue have been scanned, then proceeds with the transition.

VCBR (Figure 82) first sets VX1 (VTSA) non-zero thereby blocking any tapes coming ready at VTSA from attaching to ST. Such tapes will merely line up in the VTSA queue where they will be available when an overlapped mode is resumed. The phase of SC is set equal to that of CR by restoring the phase parameter which existed when in the last previous card reader mode. The CR phase will have been unaffected by the offline mode. Concurrently, the present phase of SC is saved for later use in restoring the SC, ST phase relation whenever the offline mode is restored.

Next, SC is connected to E, SC is modified for bypass operation, and the phase of E is set equal to that of SC. The sequence is the same as for the VBYPAN routine and the subroutine VSCBY does this for both transitions. In addition, for the VBYPAF routine, SC must be initialized to scan through B cards since it is left scanning for a B card by the offline scanning operation just completed.

Finally, the VBYON bit is set to 1 and control goes to the VRDCSX routine. The VRDCSX routine starts up the card reader just as it did for the ONLINE command and the setting of VBYON will interrupt phase 4 and complete the transition to bypass just as it did for VBYPAN.

The tape switch mechanism is left in almost the same state as in the VBYPAN command, except that tape attachment is blocked at VTSA. If a tape attachment to ST were allowed to occur, ST would read

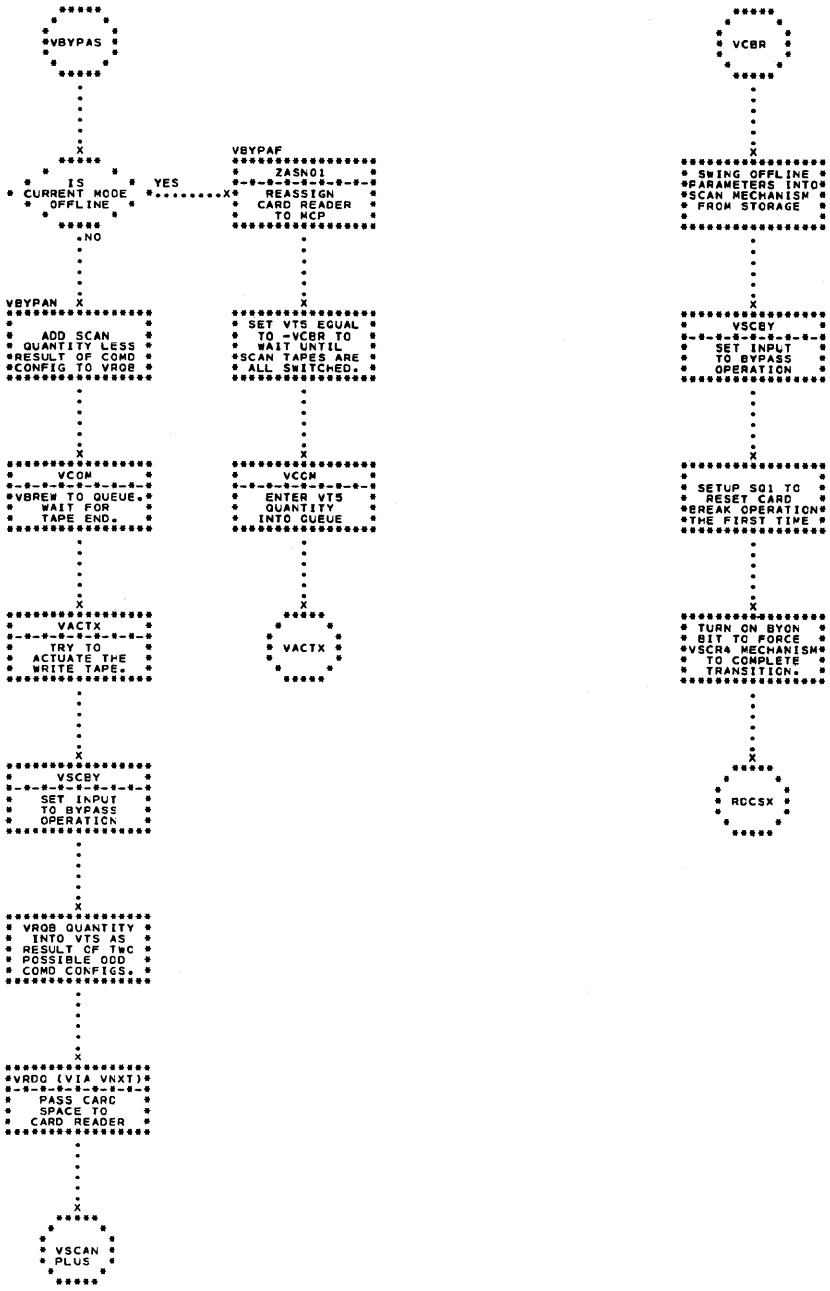


Figure 82. Input Chart 13 - Transition to Bypass

into VBF1 because, unlike WT, ST would have a full complement of buffer area assigned to it. When a mode of operation is suspended, it is generally true that the buffer parameters will collect at the functional program which is the source of data. Thus, in the online mode the CR is the source of data, and the parameters collect here when CR is blocked out by an out of material condition. Likewise, in the offline mode, the parameters collect at the data source, ST, when ST is blocked by not having a tape to read. Since the TS switch mechanism would normally provide such a tape, the mechanism must be prevented from doing so.

The philosophy of the BYPASS command is that it is primarily a temporary interruption of the existing overlapped mode. Therefore, the transition routines to bypass alter the prior overlapped mechanism as little as possible. The compound case, resumption of overlapped operation in a mode different from the previous overlapped mode is handled by the two resumption commands on an exception basis.

Offline from Online -- VOFFLN

The VOFFLN routine (Figure 83) is entered when JC1 identifies an OFFLINE command card. The VOFFLN routine first determines whether the OFFLINE card is the last one of the input file from the card reader. If it is not, it gives an end return to the command package which causes the OFFLINE command to be ignored and processing of the input file continues in the online mode. Otherwise, VOFFLN alters the card reader CS fixup to ignore CS thereby preventing the card reader's possibly reading into the phase 1 buffer.

In order to effect the transition to offline the following steps must be taken:

1. Modify VTSB so that old phase 4 tapes are unloaded and so that tapes coming ready at VTSB are given an ST destination.
2. Modify VSC and connect it up with ST.

Changing the destination of old read tapes can only be done when it is definite that all of WT assigned buffer area has been written on tape. A negative parameter given to WT at this time will be picked up only after all prior assigned buffer area has been processed. Therefore, the parameter, -VFREW, is forwarded to WT.

When -VFREW is removed from WT queue, VFREW proceeds with the offline transition. The phase of SC is set equal to that of ST by simply restoring to SC whatever position pointer it last had in the offline mode. ST, used only in the offline mode, will still have the same position. At the same time, the current position pointer of SC is saved so that it may be used in restoring the phase identity of SC and CR when leaving the offline mode (a SWAPI instruction accomplished both functions). SC is then modified for offline operation: specifically, it will not forward the parameter, -VWEFT, at job boundaries to ST. At this point, the phase 1 operation is set up and ready to operate; however, it is not started because provision for supplying tapes must first be made.

In order that future tapes coming ready from VTSB arrive at ST, the VDS (VTSB) slot must be set to ST. Furthermore, expended read tapes should be unloaded in the offline mode, not rewound, and so VOP (VTSB) is set to 1. However, it is possible that some tapes have already come ready and been put in the VTSA queue with WT destinations. The subroutine VMTTSA removes these tapes from the queue and recycles them through VTMV to be unloaded, whence they will come ready with ST destinations because of the new setting of VDS (VTSB). When the first tape does come ready it will initiate phase 1 operations through the normal tape switch mechanism, the transition procedure being completed by then.

VFREW must do one more thing, depending on how it was removed from the WT queue; whether in the normal continuation of WT processing or in reinitiating WT processing upon receipt of a new tape from tape switch. When -VFREW is delivered to WT by the VOFFLN routine, WT may terminate the tape just prior to picking up -VFREW from its queue. In this case parameter flow would halt at WT until a new tape is delivered to WT, at which time VFREW would be picked up. On the other hand, VFREW may be picked up in normal continuation just after the VWEFT procedure has been completed, and VFREW would use the write tape termination procedure to send the completed write tape to phase 4. VFREW can distinguish between the two cases by testing the file and record counts, both of which would be zero in the tape attachment case. In this case VFREW passes back the tape just being attached as it did for the tapes found in the VTSA queue. In normal continuation, VFREW goes to the WT tape termination procedure.

This completes the OFFLINE transition. It is well at this point to review the status of the old phase 1 configuration. The WT program is now in a position to receive a new tape, although it will not get any because VDS (VTSB) is set to ST. Also, WT is in a position to receive cards from SC, but will not get any because SC is now feeding ST. All of the phase 1 buffer parameters have by this time been moved to CR where they are held up because an out of material condition exists at CR. This will not be inadvertantly removed by readying the card reader, since the CS fixup has been altered.

Offline from Bypass -- VOFFLB

The VOFFLB routine (Figure 83) is entered under circumstances similar to the VONLIB transition. The card originating the command must be the last card in the file. This condition is first tested by VOFFLB and if it is not satisfied an "end" return is given to the command package and the remainder of the file is processed in the bypass mode. Otherwise, the future inactivity of the card reader is ensured by altering the CS fixup to ignore channel signals.

The phase 4 operation is resumed immediately by altering VREQ so that is thereafter processes requests from the Q tables and VBF2. The SC is modified for offline mode operation, connected with ST, and the SC-ST phase integrity is restored just as in the VOFFLN routine. At this point phase 1 operation is ready to be started, but the method of starting depends on whether this transition is simple or compound.

If simple, then the tape switch mechanism is already set up as for offline operation, and VTSA, which was blocked by the VBYPAF routine, need only be restarted by using the VACT subroutine with the VTSA queue.

If the previous overlapped mode were online, there may be a tape attached to WT and also some tapes in the VTSA queue with WT destinations. Using the subroutine VMTTSA will remove the tapes in the VTSA queue and put them through an unload cycle, just as for the VOFFLN routine. If a tape is attached to WT as indicated by a zero record and file count, it is removed by the VRJTP routine (Figure 84), which is that part of the VFREW routine performing this function for both the VOFFLB and the VOFFLN routines. Just prior to cleaning out VTSA, VOP (VTSB) and VDS (VTSB) were set for offline operation, so tapes coming ready at VTSB will be forwarded to the proper destination, ST. The first such tape will start phase 1 automatically, thus completing this transition.

The Input Command Dispatcher

All input commands except SSCR and SSCR4 enter through a single tentacle table, VSKM. This specifies as its entry point the command dispatcher routine, VSSKOM, which dispatches control to the appropriate command routine. The first parameter in the SKOM calling sequence generating the command specifies the command desired. This parameter is placed in the VVSKM tentacle where it is available to the VSSKOM routine.

VSSKOM uses this parameter in the sense of a relative location within one of two branch tables which contains branches to the appropriate routines. The particular branch table used is set externally to VSSKOM by some of the transition routines. The command routines addressed by the tables versus the relative location parameters used to address the particular entry are as tabulated below:

SKOM Name	Parameter Value	Overlapped Branch Table	Bypass Branch Table	Request For
SONL	0.	VONLIN	VONLIB*	ONLINE transition
SOFFL	.32	VOFFLN	VOFFLB*	OFFLINE transition
SEOF	1.0	VEOFN	VEOFB	System input EOF
SEJSCN	1.32	VOEOJ	VBEOJ	End of job scan
SBYP	2.0	VBYPAS*	VBSTRT	BYPASS transition
SREW	2.32	VSREW	---	System input rewind
SCR1	3.0	VSCR1	---	Phase 1 B card request
SCAN	3.32	VSCAN	---	Phase 1 scan to job boundary

The transition routines with an asterisk set the VSSKOM routine to use the other branch table. The various routines return to the command dispatcher with the return set as end or normal. The command dispatcher stores the return address in the tentacle table and issues $RET.

The REWIND Command

The system REWIND command allows the operator to control termination of the write tapes in the online mode of operation. The REWIND card is placed in the input file just before the job which the operator wants to place on a new write tape.

Control is given to VSREW for a REWIND command as soon as the card is recognized by JC1. Since control is received in phase with SC but implies action in a subsequent program (WT), a negative parameter, -VSSREW, is forwarded to WT. When VSSREW is entered as the result of the corresponding parameter's

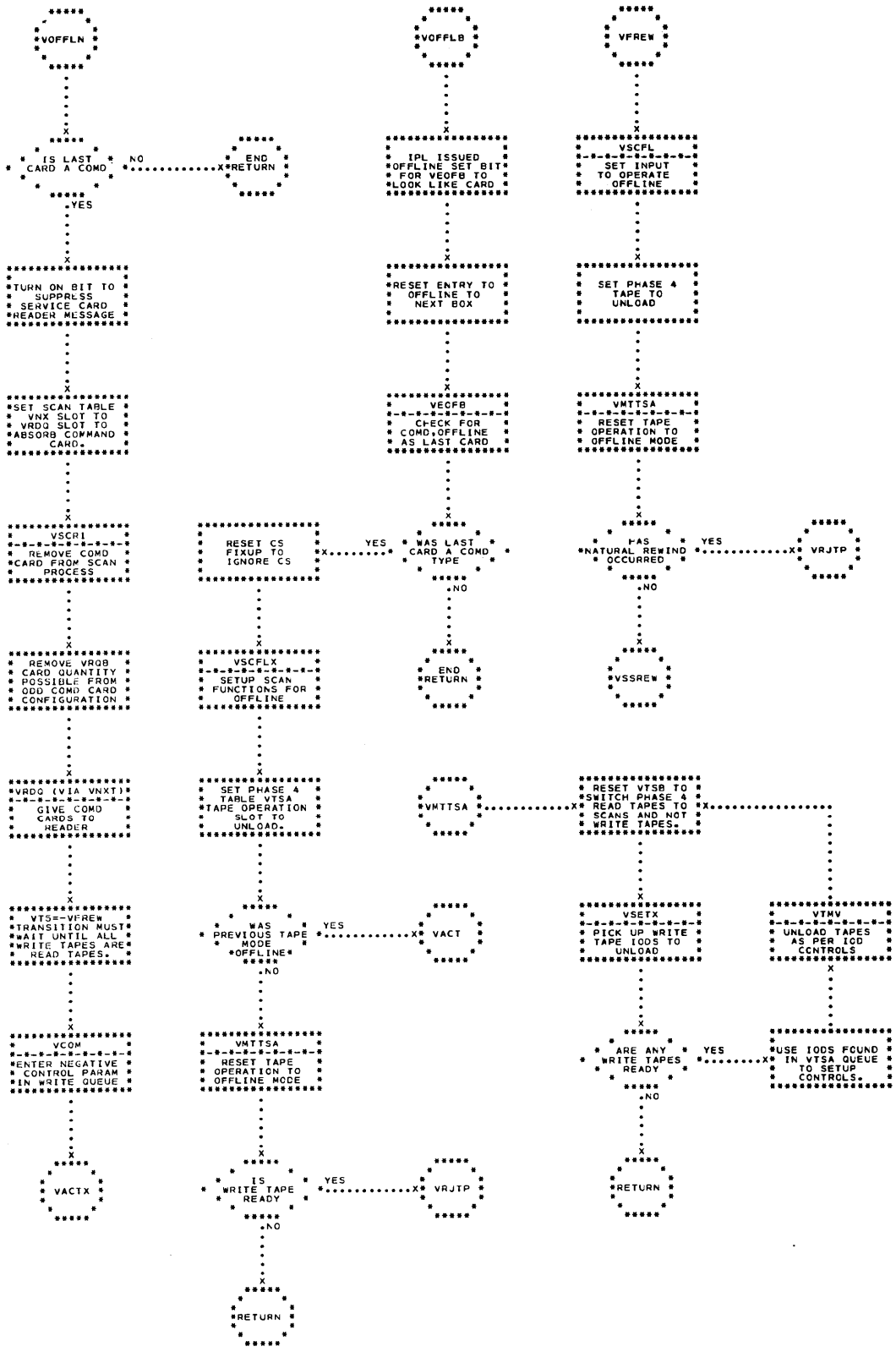


Figure 83. Input Chart 14 - Transition to Offline

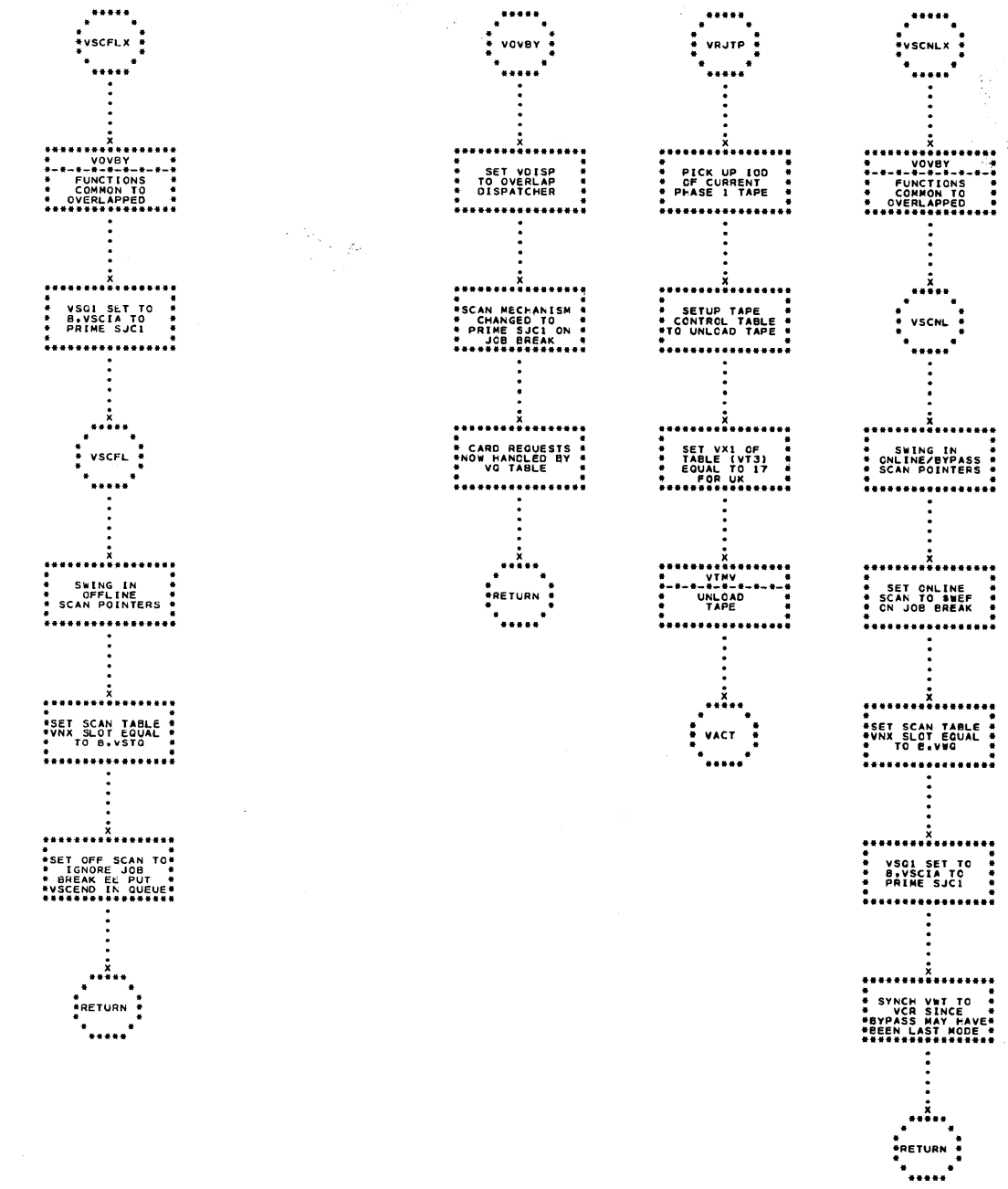


Figure 84. Input Chart 15 - Transition Subroutines

removal from the WT queue, it forces the phase 1 tape termination procedure, provided that the write tape has not, by itself, just applied this procedure. Should this be the case, VSSREW just returns to VSETX to get the next parameter from the WT queue. VSSREW is able to make the distinction by testing the record and file counts, both of which would be zero if -VSSREW were picked up when a new tape was being attached.

The System Input EOF Command

The EOF command is received in either of the two card reader modes from a card which should be the last card in the file. A file terminated in this way will not cause the message, "Service the card reader," to be printed, as would a card reader file terminated by EE alone. It will set a bit, VEOFX, to 1, to indicate that the card reader will be inactive for an extended period. The procedure for restarting the card reader is unaffected, however.

Internally, the EOF command works in conjunction with -VEES parameter which is forwarded from CR when an EE is received on the card reader. This parameter propagates itself to the ultimate destination of the cards, which would be WT in the online mode, and E in the bypass mode. When it is picked up at the final destination, it tests VEOFX and puts out the service message through the commentator only if the bit is zero. By thus delaying the message, the EOF routine gets a chance to suppress the printing of the message.

When the VEOF routine gets control, it first tests to see if it was the last card in the file. If it was not, the command is ignored, otherwise the VEOFX bit is set. The VEOFX bit is also set by the VOFFLB and VOFFLN routines. After setting the VEOFX, the VEOFB routine, (corresponding to EOF in bypass mode), just gives a normal return to commands. The VEOFN, corresponding to EOF in online mode, then gives control to the VSREW routine as if it had been a rewind command. The VBSTRT routine, corresponding to a BYPASS command received in the bypass mode, is a special routine used in starting the system.

The Initial Start of the Input Program

The input program starts when the command package issues one of the transition commands to it. The command sets the mode of operation. The program is initially set up so that some of the normal transition routines can start the system. Specifically, it is made to appear as if the current mode were bypass and the previous mode were offline, that is:

1. The command dispatcher is set as if in the bypass mode.
2. Phase 1 is in a bypass configuration; namely, CR to SC to E and back. The VREQ routine is set to service requests from E.
3. The VTSA queue contains two tape IOD numbers, VRTP and VWTP, and the tapes have ST destinations. VTSA is blocked with a non-zero VX1.
4. All functions programs are devoid of assigned buffer areas except CR, ST, RT. These are ultimate sources of information for the buffers and have a full complement of 34 cards assigned.

If the command dispatcher is set as if in the bypass mode, one of the three routines, VONLIB, VOFFLB, and VBSTRT will be addressed by the initial transition command entry. The VBSTRT routine is used when a bypass command is received in the bypass mode. This routine starts up the card reader by using the VACT routine for CR. As a safeguard it also modified the particular entry in the command dispatcher which addressed it, so that VBSTRT will never be addressed again. Since the program is all set up for bypass operation, no other action is required.

The VONLIB and VOFFLB routines are able to initiate their respective modes as they normally would, except that the VOFFLB routine must set a bit, VCSAW, to 1 to satisfy the test for the last card in the card reader file despite the fact that the command did not originate from a card.

The ST destinations of the tapes in the VTSA queue is significant because either overlapped mode will accept such tapes. However, if the destinations were WT, the offline mode would reject the tapes, putting them through an unload cycle. This would be a nuisance to the operator in starting up the system in the offline mode. Phase 4 operation is started in the normal way by the tape switch mechanism when and if the first tape is attached to RT.

The Input Program Fixup Routines

The input program normally uses three I-O tables of exits, and uses certain others during error recovery. EOP, EE, and CS interrupts relate directly to the functional programs, and no special fixup routines are provided. The input program fixup routines are confined to UK and EPGK error recovery routines for the various pseudo-ops.

The basic purpose of the input program UK procedure is to complete successfully any I-O pseudo-ops initiated by the input program. There are two possible results of the UK procedure:

1. A successful retry of the pseudo-op. This has no net effect on the input program.

2. An unsuccessful retry of the pseudo-op, and subsequent rejection of the jobs involved in order to keep the system running.

Thus, the UK procedure consists of two distinct phases for each error source: the retry of the pseudo-op, and the internal adjustment if it is unsuccessful.

This approach is used separately for each of the following:

1. UK/EPGK on tape write
2. UK/EPGK on card read
3. UK/EPGK on tape read
4. UK/EPGK on WEF
5. UK/EPGK on other pseudo-ops

The Input I-O Tables of Exits

Three I-O tables of exits are associated with the input functional programs:

IODNAME	TOE	Program
VCRD	VCREXT	Card reader
VWTP	VWTEXT	Write tape
VRTP	VRTEXT	Read tape

The tables of exits are permanently associated with the IOD names specified; however, the two tape tables alternate between the read and write (scan) tape functions.

Each tape table has the following form:

```
A       VF, (Functional program table name)
(I-O    DRZ, (BU, 64), 2
TOE     SIC, RKSBRC; B, RKSUK              'UK, EPGK
name)   LV, VU, A; B, -.32(VU)             'EE
        LVI, VT5, (IOD name); B, VTDSP     'CS
        LV, VU, A; B, -1. (VU)             'EOP
```

The card reader table differs only in that the CS position contains a branch to VRDCSS.

As previously mentioned, it is the function of the tape switch routines to enter the proper IOD name in the VSU position of the tape program tables, and to put the corresponding table name in the VF at A ahead of the TOE. Thus, when either I-O TOE is entered with EOP or EE, control is given to the proper functional program table at the slot corresponding to the interrupt. When CS occurs, control is returned to the tape switch routines, and when an error interrupt occurs, the fixup routine is entered at RKSUK.

Generalized Dispatcher

When an error interrupt occurs during an input program, the following steps are taken (Figure 85):

1. A test is made for EPGK or UK.
2. The pseudo-op is identified as $W or $RD. If $W, go to RGWUK (Figure 86); if $RD, go to RDUK (Figure 91).
3. If a UK-no EOP condition occurs on a control operation, it is an uncorrectable situation. MCP halts on a TYPE 76 error.
4. The pseudo-op is tested to see if $WEF. If not, go to RPUKLI (Figure 93). If it is, enter write end of file procedure.

Notice that the test for EOP is made after checking for a $W or $RD pseudo-op. If a UK- no EOP condition appears with $W, it may be a data error in a chained op on a write tape, which can be fixed by RGWUK. On the other hand, if a UK- no EOP condition appears with a $RD, it could be a card reader jam, which is taken care of by RDNOTP. A $WEF or pseudo-op lack of EOP is a unit down type condition.

EPGK Condition

The input program may cause a legitimate EPGK in the following situation (Figure 85): If the input program was in the offline mode, any scan tape to be saved would be file protected. If caught in a transition to online, this tape would be used as a write tape. When the $W attempts to write on this tape, an EPGK would be given. The tape must be unloaded and a scratch tape mounted before $W can be reissued.

UK on Write or Write End-of-File

A retry of both the $W or $WEF pseudo-ops will follow the same general procedure (Figure 86). The tape will be repeatedly backspaced to the preceding record gap and the write pseudo-op retried. When an arbitrary number of retries has been made, an erase gap SEOP (ERGS) is performed whereupon the backspace-write retry count is reset and the retry series begins again further along the tape. The ERG procedure is also limited by an arbitrary count, and when the count is exhausted the situation is considered a total UK failure.

Setting Up a Retry of $WEF: UK is possible on any of the following WEF cases:

1. There were two consecutive WEF's and the tape did not move.
2. There were two consecutive WEF's and the tape did move.
3. There was one WEF and the tape did not move.
4. There was one WEF and the tape did move.

The first step in handling a UK in any of these cases is to backspace the tape, after which a $RET is given. On the next interrupt another $WEF is attempted, after which the program is restored.

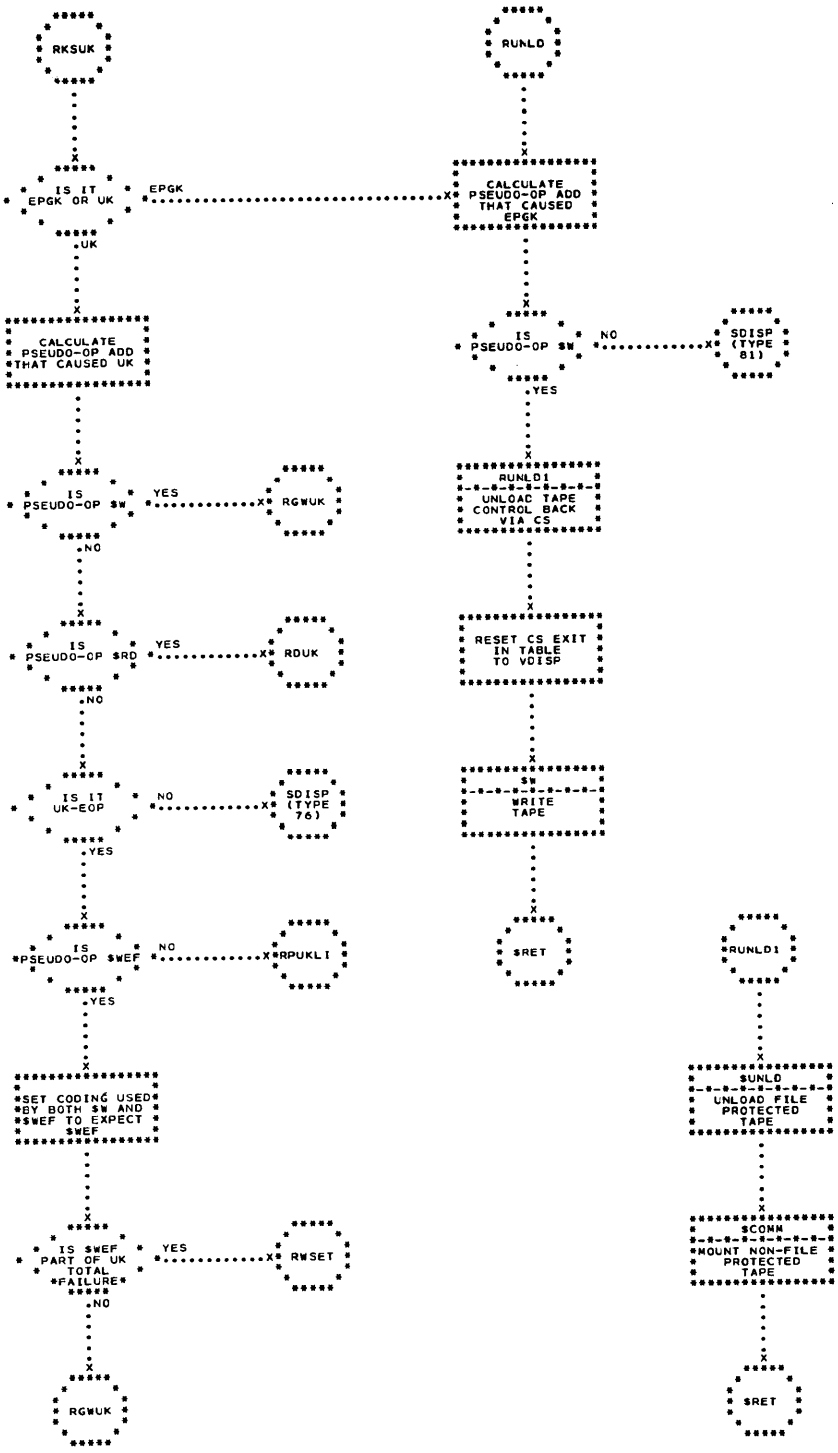


Figure 85. Input Chart 16 - UK Fixup Entry

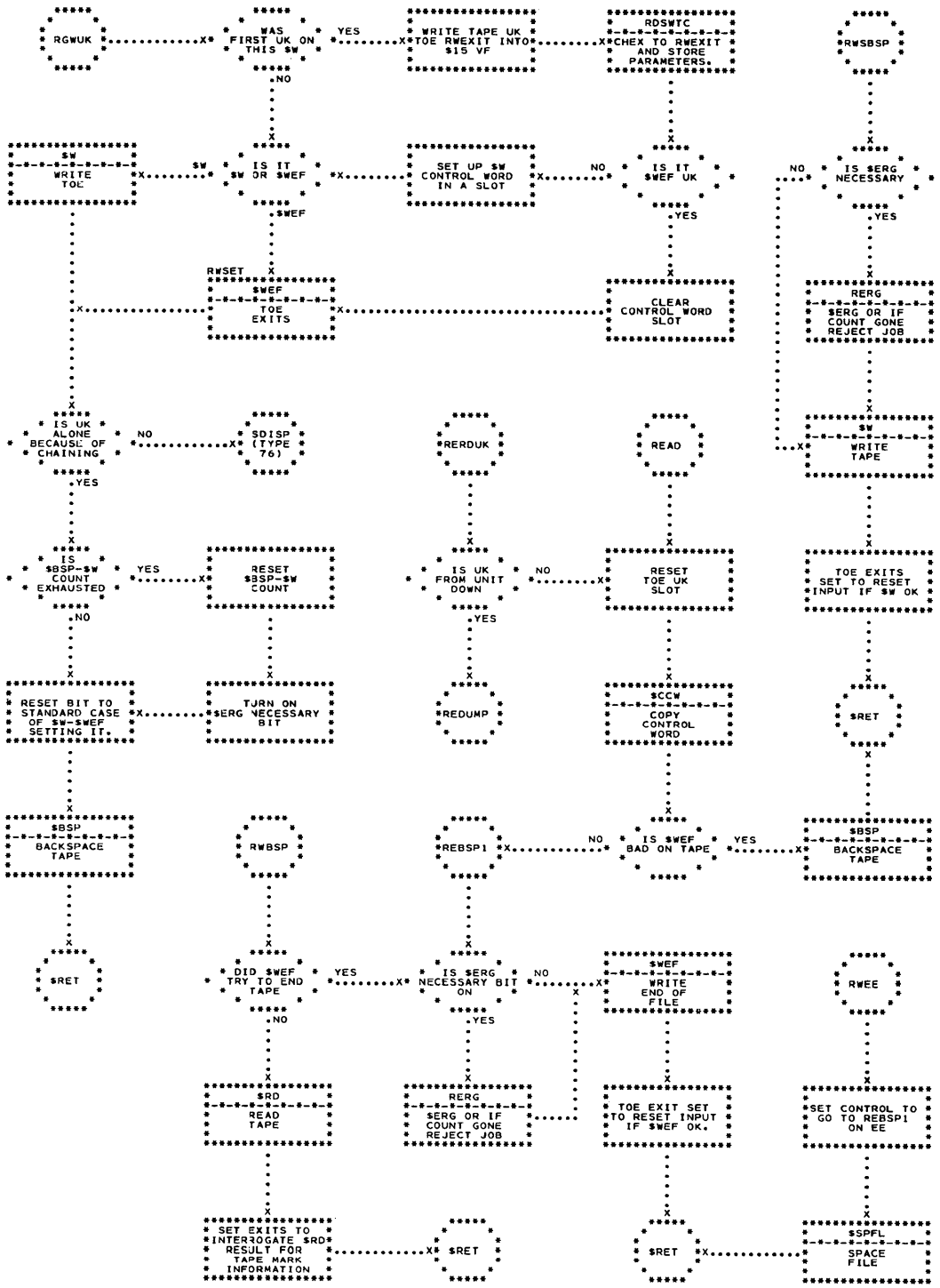


Figure 86. Input Chart 17 - Write Tape UK

If an EE appears after a backspace, the implication is that case 1 above was attempted and the tape did not move on the second WEF. After a $SPFL, the second $WEF may be reinitiated.

If an EOP appears after a backspace, any of the other three WEF cases are possible. The record count VRCNT must now be tested (Figure 87). If greater than zero, the UK occurred in case 2 above. The tape did move but the $BSP has put the tape in position for $WEF to be reinitiated. If, however, the record count is equal to zero, only single WEF, cases 3 and 4, need be considered. To determine whether or not the tape has moved, a one card read is initiated. If the $RD gives UK-EOP or EOP alone then $CCW is given and the word count is tested. If zero, the tape did not move and the WEF may be retried; if not zero, the tape moved and a bad tape mark was written on tape. A backspace must be given before reissuing $WEF. If the $RD gave a UK alone, the tape is rejected.

Retry of $WEF or $W: When a UK appears on a $W, only a backspace need be given before retrying the $W. The retry operation for $W or $WEF is very much alike. The tape is now in position and the $W or $WEF is given. The EOP slot in TOE is changed to RESTOR, and if on the interrupt an EOP is received, TOE is restored and control is given back to the input program. If, however, a UK is received, the entire fixup routine is repeated again. The tape is again backspaced and the type of operation, W or WEF, again established. A $W or $WEF is rewritten and the program restored. If a UK is received again, the fixup routine is begun once more. A count of the number of tape backspaces is maintained, and when an arbitrary limit is reached, an erase gap ($ERG) pseudo-op is performed. The backspace count is reset and the $W or $WEF again retried. An ERG count is also maintained, and when its arbitrary limit is reached, a total UK failure is assumed.

Total UK Failure on $W or $WEF: When a total UK failure occurs, the fixup program must eliminate the offending job both from the tape and parameter queue as well as terminate the input tape with two consecutive tape marks (Figure 87). Notice that the input program rather than the fixup program puts on the second tape mark. This is done because if the fixup program put the $WEF on tape and a UK should occur, the fixup program would consider this as a very special case, necessitating a great deal of extra coding. By giving the input program this chore, a UK here can be handled by the general $WEF fixup.

The fixup program must now dispose of all traces of the job from the input program tables, queues, etc. (Figure 88). If the operation that failed was a $WEF, the job is completely on tape and a $BSFL will eliminate it. If it was $W, the buffer position pointer (VX) must be moved beyond the offending job's cards as each parameter is removed from the write tape parameter queue. As the cards are skipped over, their space is made available to the card reader to refill. The number of cards being skipped over is contained in VT5, which is added to VX. Each parameter is examined as it is passed to the WT queue until -VWEFT is found. This is done by RESRCH and its associated routines.

After the preceding backspace, a $SPFL is given and the terminating tape end of file mark is written. If the original failure was a tape terminating WEF which did not move the tape, a REDOBL entry is made (Figure 87). The program cannot $SPFL and $WEF as in other cases, but must $BSFL over a completed job, $SPFL and then $WEF. Since a good job has been eliminated, the file count must be reduced by one.

In any case, should the second attempt to write end of file fail to terminate the tape, the entire tape is rejected via REDUMP and the appropriate job reject information given to job control. The entire tape will also be rejected immediately after the ERG count is exhausted if there was a previous total failure on the same tape or if the failure occurred on the first file (to prevent backspacing into the label).

UK on Card Reader

In general, there are three types of card reader failures: a mechanical feed failure, a read failure, or a mixture of both. The card reader fixup program can handle card jams, misfeeds, and misreads mixed in any order. Five UK cases are possible:

1. A single UK following a B card.
2. A consecutive total UK failure.
3. Two or more total UK failures with cards between on the same input read.
4. A single UK followed by cards.
5. A single UK on the last card of the read.

Card Reader Feed Failure: The fixup program determines whether a UK or a UK-EOP occurred on the card reader. If UK, a feed failure is assumed. The operator is notified (Figure 89) and he pushes the ready key if he can clear the jam, or the channel signal (CS) key if he cannot. The card reader is in the not-ready status when the CS is generated. The fixup program takes this as a signal to go into offline operation. If the card jam has been cleared and the cards reloaded, the operator pushes the ready key, causing another $RD to be issued.

Card Reader Read Failure: If a UK-EOP was received, a read failure is possible. The operator is notified to service the card reader. After reloading the cards, the operator pushes the ready button and tries again. The operator is expected to retry the read at the request of the input fixups, then if the attempt is not successful, the job may be abandoned.

Card Reader Total Failure: In case of a total UK failure where none of the job can be saved, the fixup program must reject the job. It first checks to see whether the card reader is on-line or not (Figure 90). If not, it looks to see whether: (1) the UK came when the system was in transit from bypass to online, in which case it gives control back to the input transition routine, or (2) the job was finished, in which case there is no reason to reject it and control can be given up normally, or (3) MCP asked for the card, in which case a bit is set and control is given back to the input program. If the card reader is not online and none of these conditions hold, the fixup program puts a TYPE 21 job rejected message on the output tape, primes EOJ and gives $RET.

If the card reader is online, the fixup program makes the read failure look like a write tape total failure to simplify Job Control communication. The WT is given a $BSFL, the RESRCH routine is entered, and parameters are stripped from the WT queue. If the card reader total failure was passed to the write tape so as to occur on the first job on tape, a $BSFL would result in EPGK. Control therefore goes to REDUMP to reject the tape.

UK Card Dropped or Retained: The fixup program will only reject jobs in which there is no doubt that a total UK failure occurred. In some cases a UK card is left in the buffer so that no more jobs than necessary are rejected. The UK card will either be read over in core or will be left in the buffer to be tested by the input program according to the following situations:

1. The UK card will be read over in core if:
   a. A total UK occurs but the UK card is preceded by a B card. The current job is rejected and the UK card is immediately read over.
   b. The UK card is preceded by another UK card. The first UK card will determine which job is rejected and the second UK card is read over.
   c. There are two or more UK cards with good cards in between on the same input read. Treated as in (b) above.
2. The UK card is left in the buffer to be tested by the input program if:
   a. There is a single UK card followed by good cards.
   b. A single UK appears on the last card of the read.

UK on Read Tape/Scan Tape

If a failure occurs on either a scan or read tape, the same fixup procedure takes place. A $BSP-$RD series is initiated and continued until an arbitrary count is exhausted (Figure 91). If the UK is still present, it might be caused by a dust particle. The tape is now backspaced three times and then spaced to its original position in an attempt to dispose of any dust particles. The $BSP-$RD series is reinitiated and a triple backspace again takes place. The triple backspace procedure is also controlled by an arbitrary count. A total UK failure is assumed when the triple backspace count is exhausted.

If the UK had occurred sufficiently close to the beginning of the tape, backspacing might cross the label file mark, giving an EPGK. This resets the space count and causes entry into the space procedure.

RT/ST Total Failure: A total failure on a scan tape might have happened at an inopportune time for the input scanning and I-O preassignment in Job Control whereas a total failure on a read tape has no problem in this respect. In any event, a $SPFL is given.

If the failure occurred on a scan tape, a $BSFL is given (Figure 92), then a negative parameter (-RKSCNN) is put in the SC parameter queue and control is given to the input program. The input program must always be given a first B card to be able to terminate a previous job, while preassignment must be given the B- non-B break to round up preassignment correctly within a job. A test is made to see whether JC1 has gotten the first B card. If not, a blank card except for a B in column one is given to JC1, followed by 16 blank cards. If it has gotten the first B card, then 17 blank cards are given to JC1. Control is returned to the input program.

If the failure occurred on a read tape, a check is made to see whether an EOJ has been issued. If it has, the fixup program will not reject the job but will instead return control to the input program. If an EOJ has not been issued, a -RTDOWN is put in the Q parameter queue causing the correct job to be rejected, and the input job boundary is set up. Control is then given back to the input program.

UK on Pseudo-Op

A failure on a pseudo-op calls the pseudo-op UK routine into play (Figure 93). Not only does this routine try to reissue correctly all non-data transfer pseudo-ops given by the input program, but it also tries to reissue all fixup program pseudo-ops. The

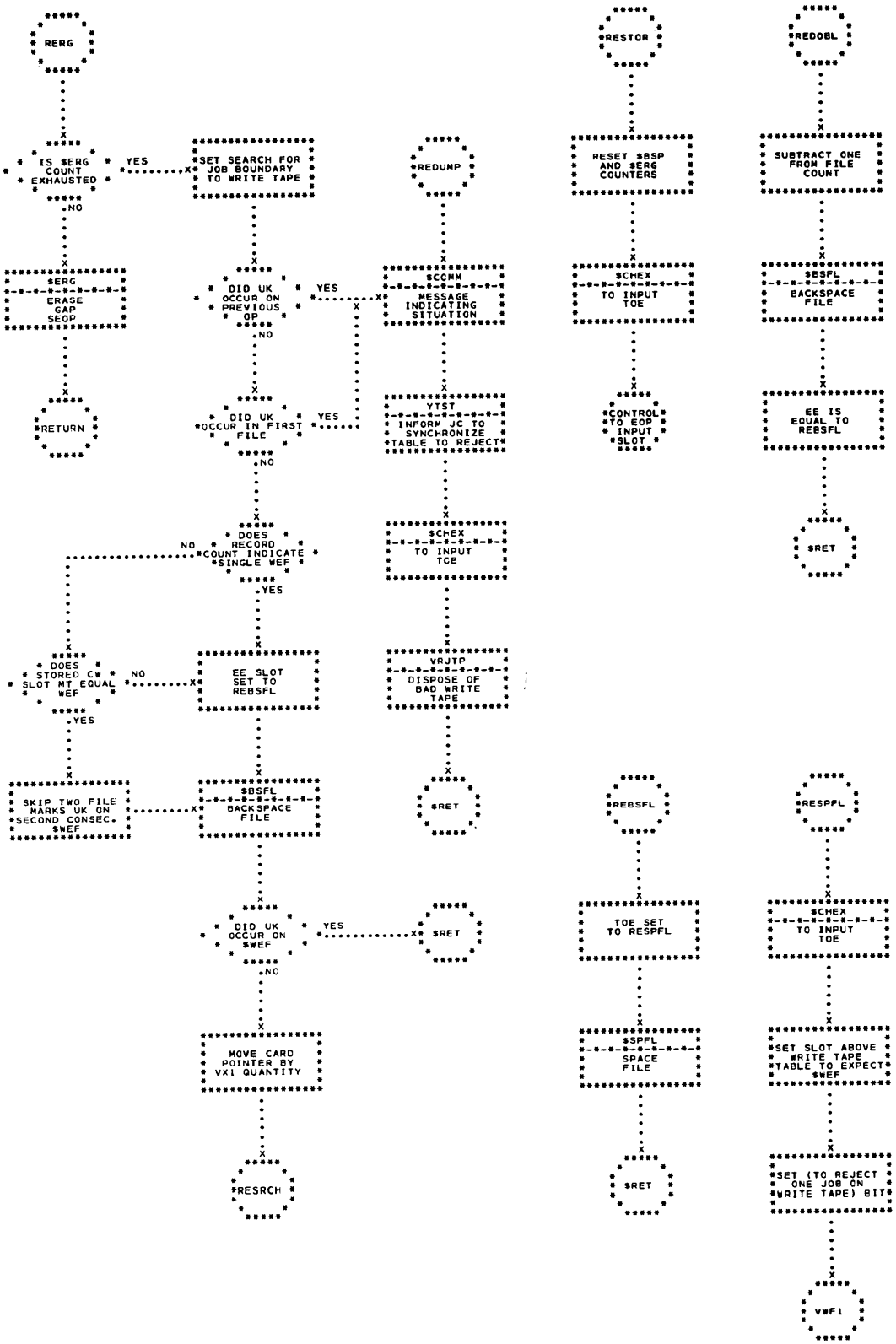


Figure 87. Input Chart 18 - Erase Gap UK Procedure 1

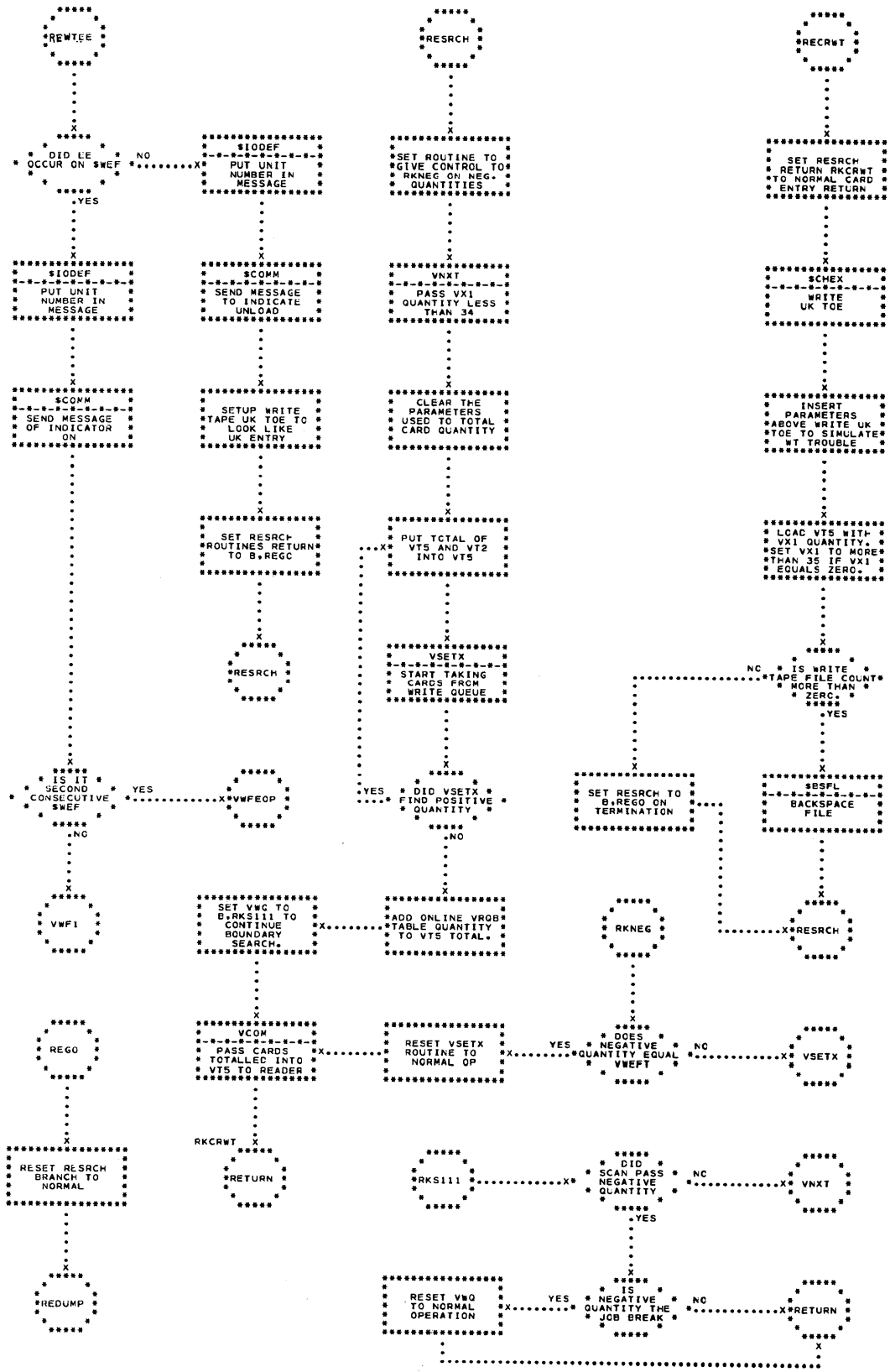


Figure 88. Input Chart 19 - Erase Gap UK Procedure 2

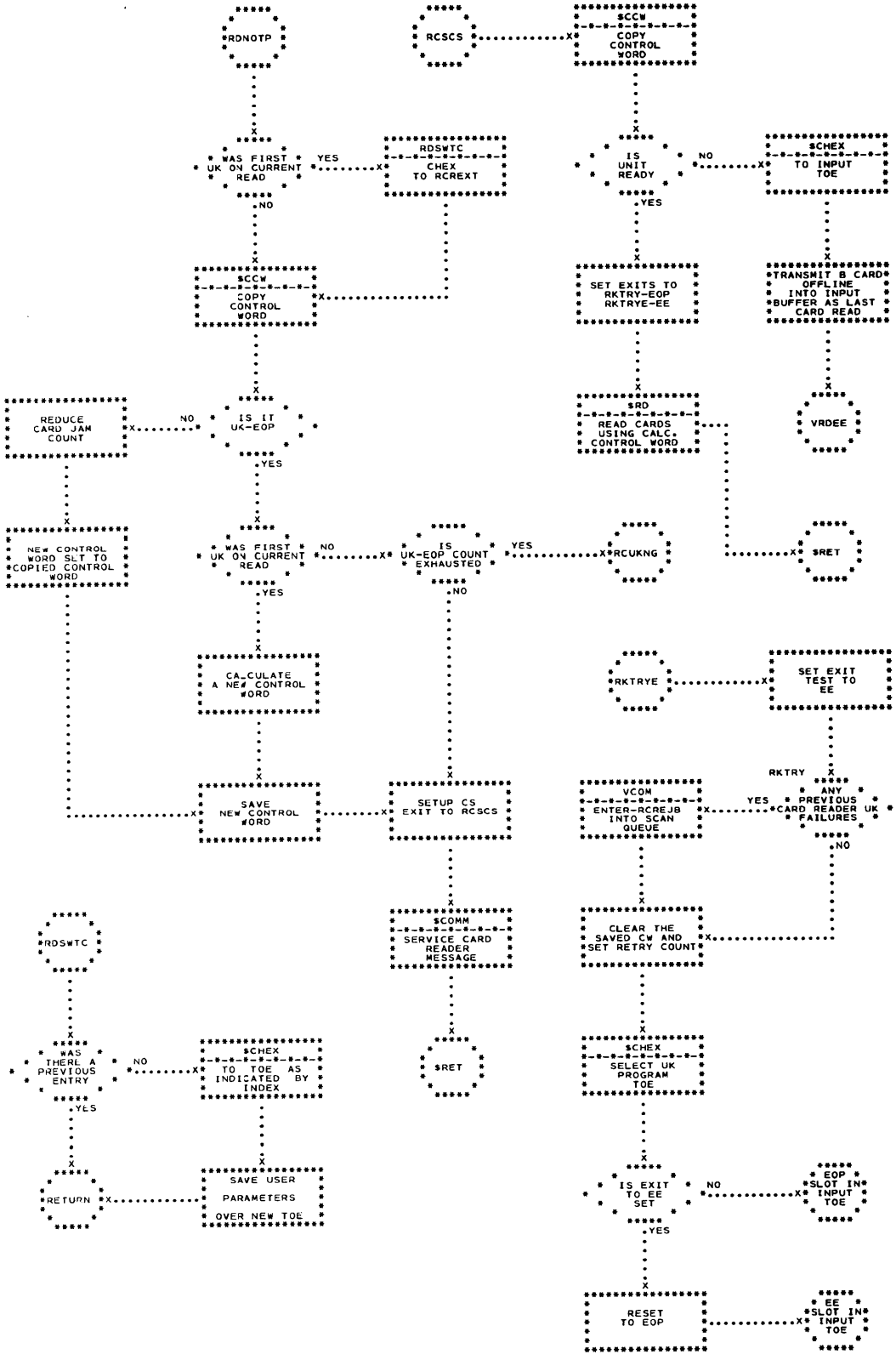


Figure 89. Input Chart 20 - Card Reader UK 1

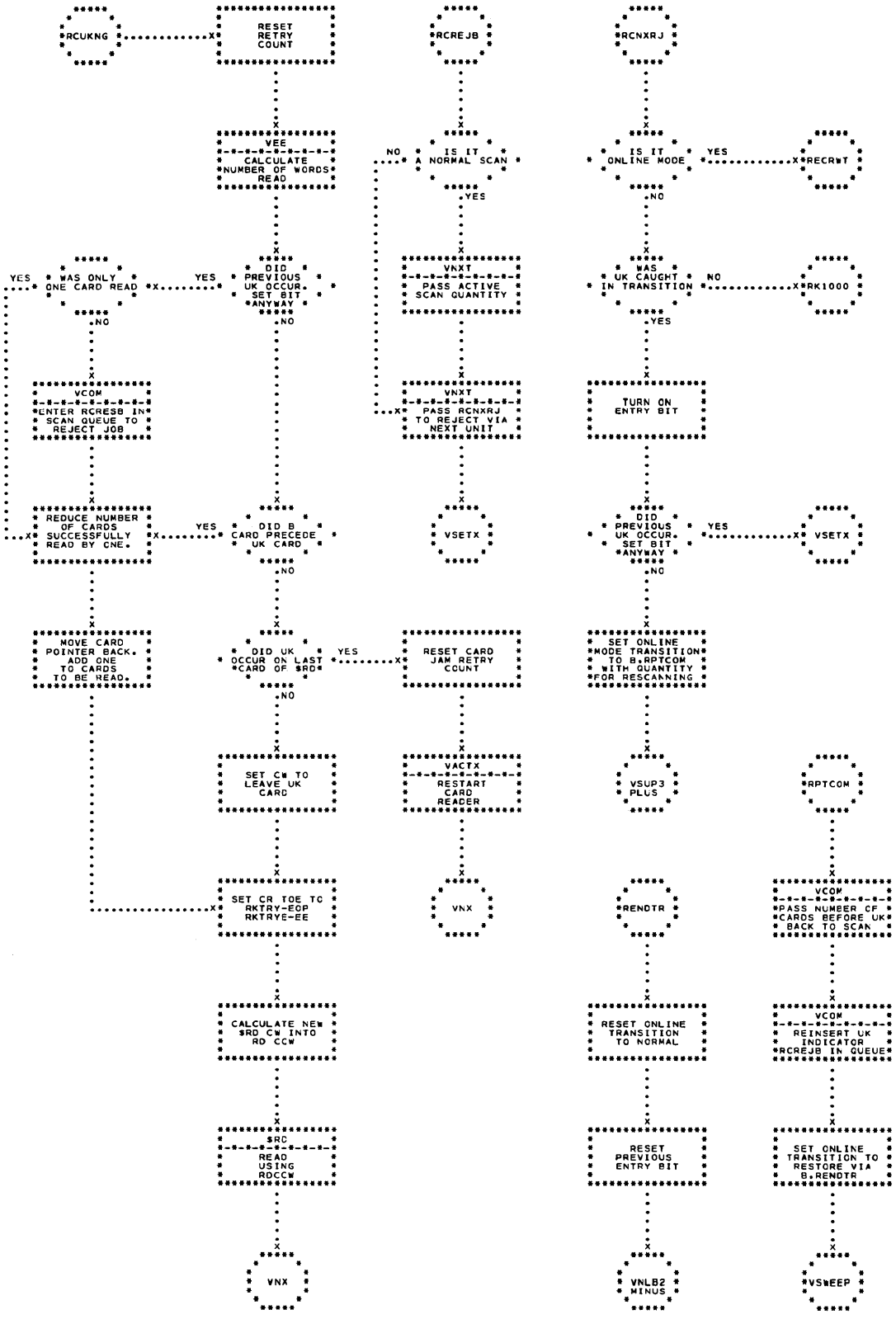


Figure 90. Input Chart 21 - Card Reader UK 2

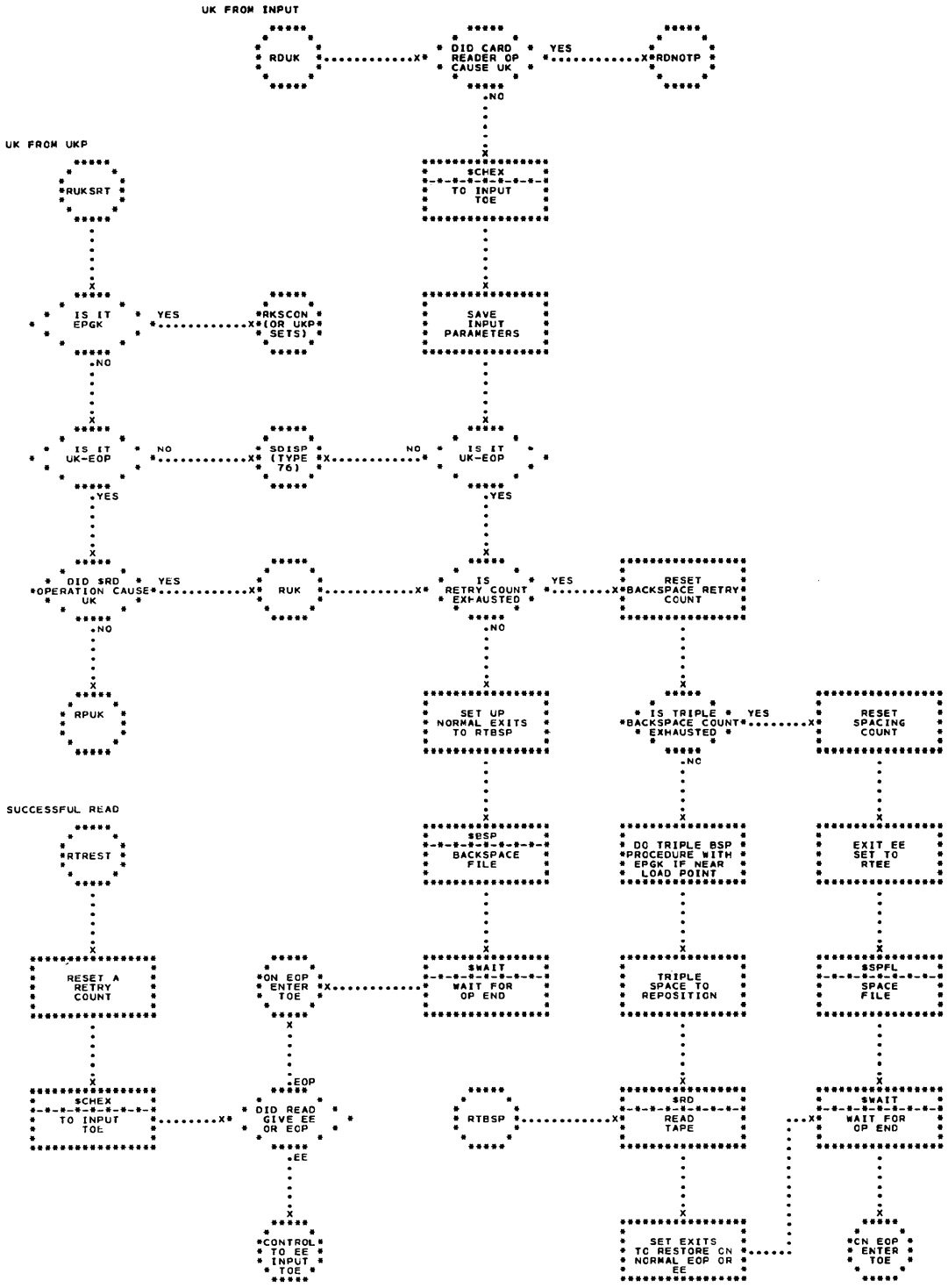


Figure 91. Input Chart 22 - Read/Scan Tape UK 1

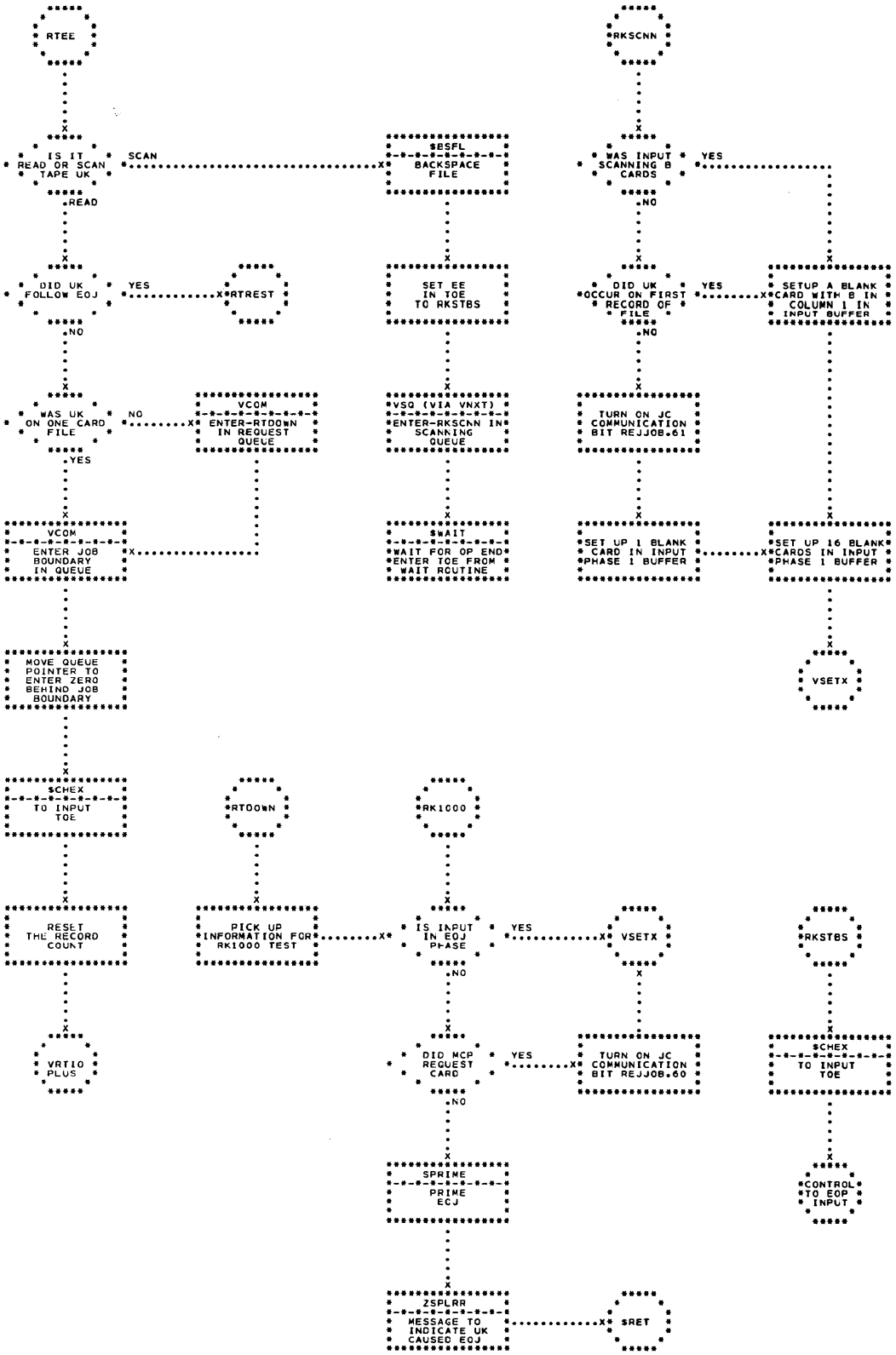


Figure 92. Input Chart 23 - Read/Scan Tape UK 2

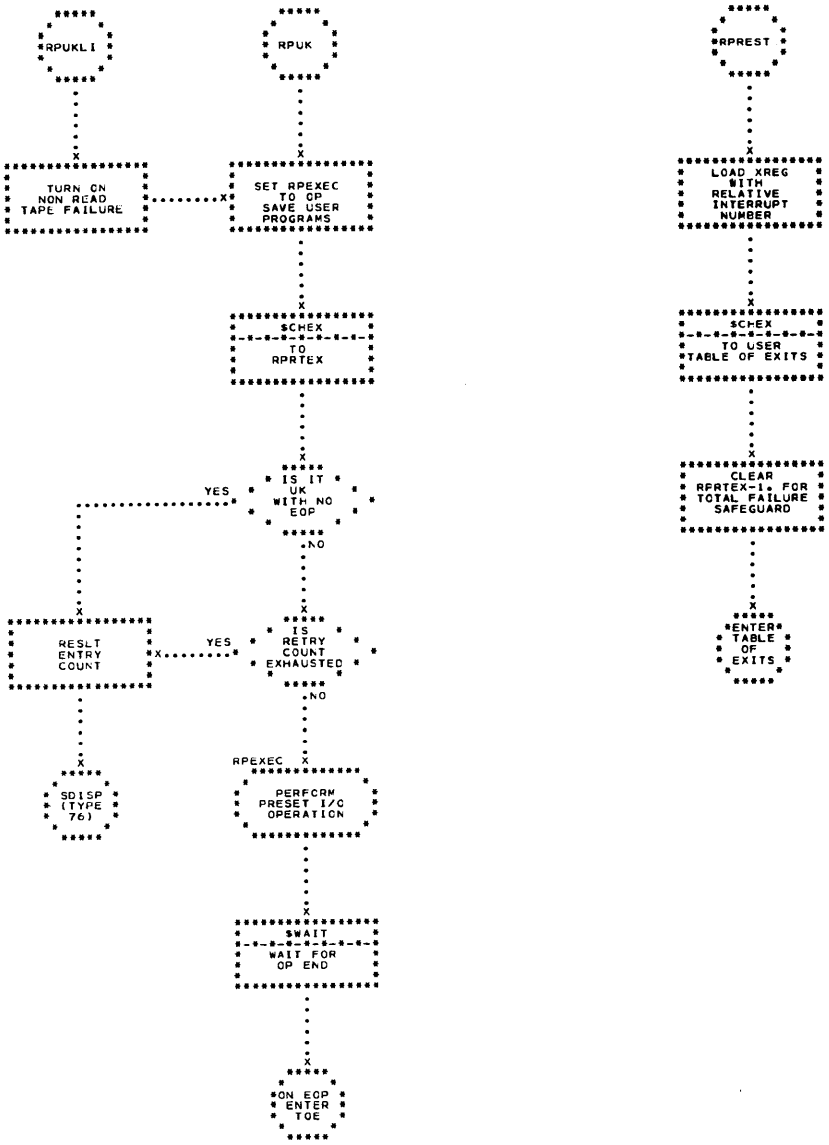


Figure 93. Input Chart 24 - Control OP UK Failure

type of pseudo-op is identified and is retried an arbitrary number of times, after which MCP halts on a TYPE 76 error.

Fixup Program Significant Counts and Bits

I. JC Communication
1. WT
REJJOB=1., JC rejects one job
REJJOB=2., JC rejects tape
2. RT
REJJOB.60 on, JC rejects current job
3. ST
REJJOB.61 on, JC rejects current job
4. CR
REJJOB.60 on, JC rejects current job

II. Internal Fixup Program
1. REJJOB.17 on, indicates a UK total failure on a $W or $WEF.
2. REJJOB.63 on, an ERG is to be given upon completion of a backspace op.
3. REJJOB.62 on, a UK total failure has occurred.
4. REJJOB.59 on, a UK total failure has occurred in the middle of an ONLIB transition.

III. Retry Count Fields
1. RWTRY/ - (WT) limits the number of $BSP - $OP procedures to RKD SYN --
2. RETRY/ - (WT) limits the number of $ERG following RWTRY failures to RKC SYN --
3. RTTRY/ - (ST & RT) limits the number of $BSP - $RD procedures to RKB SYN --
4. RTSP3X/ - (ST & RT) limits the number of triple spacing procedure to RKH SYN --
5. RBSP3X/ - (ST & RT) controls spacing ops to give 4 $BSP and 3 $SP to reposition tape to give $RD
6. RCOUNT/ - (CR) limits the number of retries done by the operator to RKF SYN --
7. RCJAM/ - (CR) an infinite count of card feed operations used by fixup program to control changing of TOE
8. RPTRY/ - (WT, ST or RT) limits the number of pseudo-op retries to RKA SYN --

SYSTEM OUTPUT

The system output program performs the writing required by the system pseudo-ops $SPR and $SPU. In addition, it provides logical control of the information on the tape at EOJ, and responds to the system command: COMD,OUTPUT.

The program consists of four major package (print, punch, EOJ, output command) and a common subroutine to accomplish tape switching when required. In addition, fixup routines are provided to handle interrupts from the output tape.

The Output Tape

The tape prepared by the output program is intended to be processed by the system peripheral output program on the IBM 1401. It consists of a mixture of printer and punch records. Each physical printer record contains twelve full lines, and each physical punch record contains ten card images.

A record is written at EOJ and whenever the specified number of units of output has accumulated. Jobs are separated by tape marks, the end of output on the tape being indicated by a double tape mark. This is followed by a trailer record indicating whether or not the tape was terminated at the same time as the job.

In order to expedite physical tape switching, two output tapes (CT11 and CT21), are defined by IOD cards as distinct units on the same channel. The I-O tables of exits (TOE) for these two units are AXITA and AXITB, respectively. Since the output tape may be referenced by any of the output programs and by error control routines, five parallel TOE's are used to direct control to the various fixup routines. Additional TOE's are used by the error control routines. Any time the output tape is referenced by one of the output programs, the location of the applicable TOE is stored in AOP. The basic TOE's, AXITA and AXITB, pass all interrupts except CS to the table specified in AOP, along with the two control words. On receipt of a CS interrupt, control is given to the fixup routines ATPACS and ATPBCS respectively (Figure 94). If a tape switch is in process, these routines give control to the tape switch routine to complete its function. Otherwise they set a bit (ATARI, ATBRI) to indicate that the tape is ready, and they issue a $RET.

The Print and Punch Programs

The print and punch programs are logically identical, except for titling. (See Figures 95, 96.) Since printer and punch output cannot be predicted, they are written as separate programs, and are to be buffered and controlled separately.

Pseudo-ops $SPR and $SPU, respectively, cause the print and punch programs to receive control from the IF analyzer with tentacle table linkage set up. The print program begins at CSTART, the punch program at ASPUO. The respective tentacle tables are

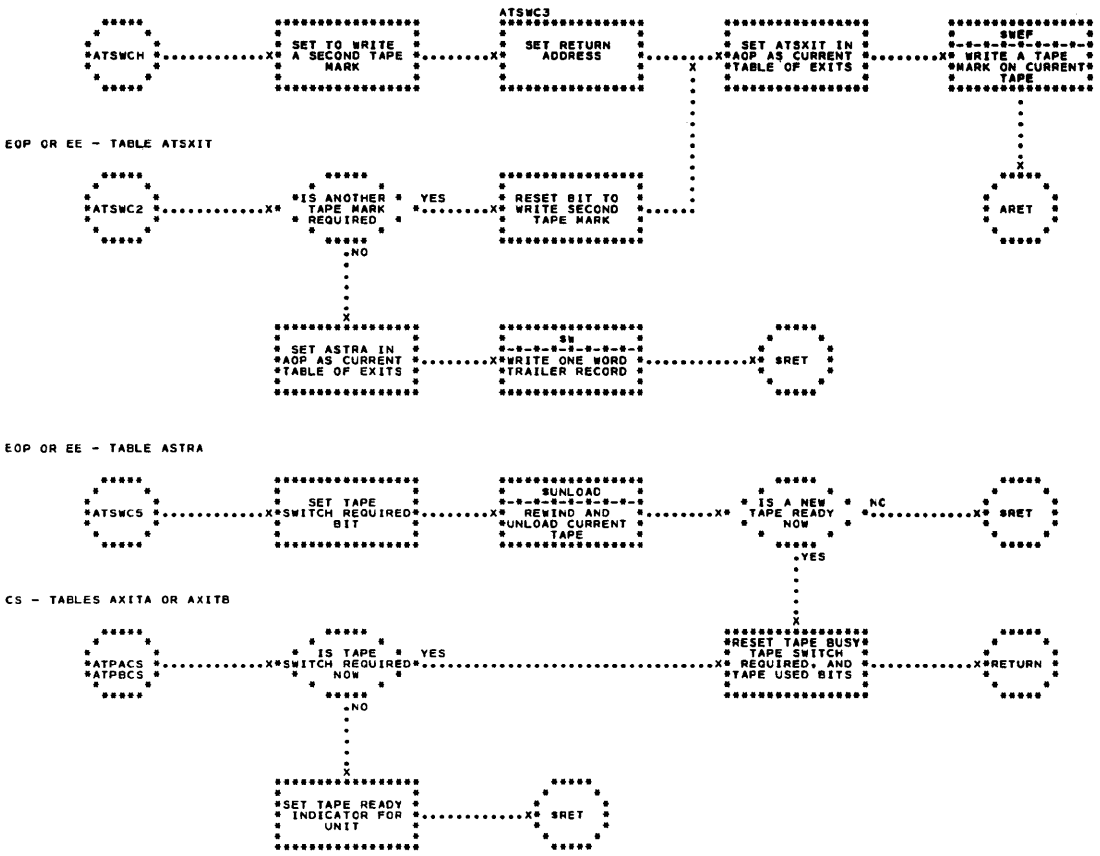


Figure 94. Output Tape Switch Routine

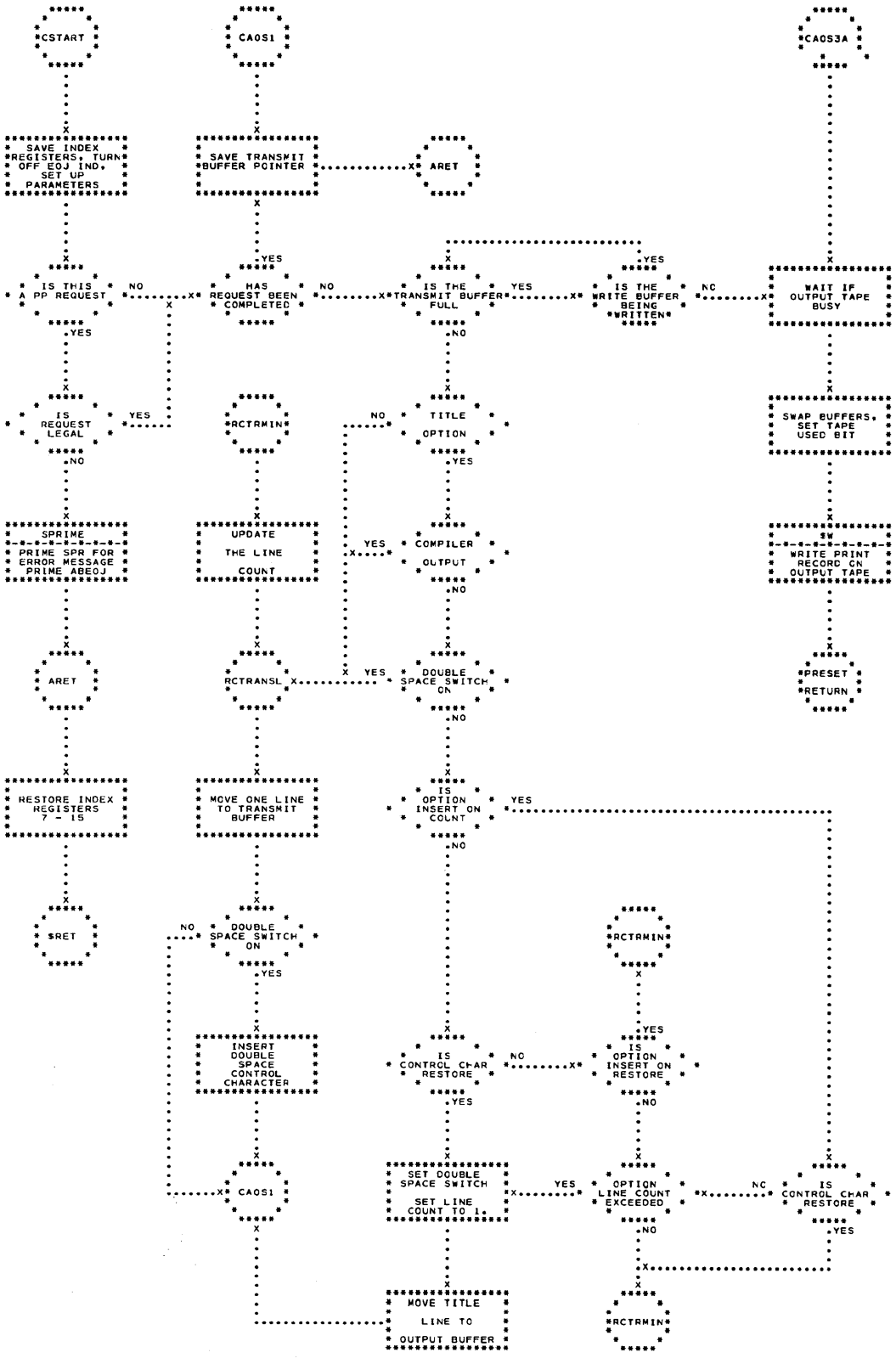


Figure 95. System Print Routine

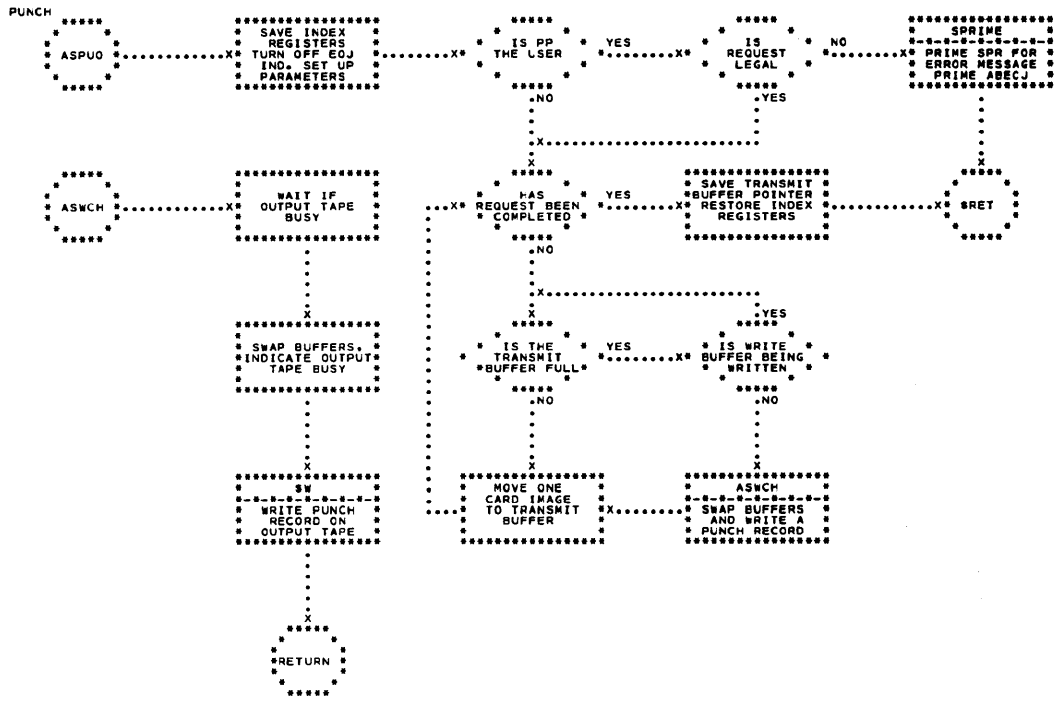


Figure 96. System Punch Routine

SCSPRT and SASPUT, each four words in length. Both programs operate enabled, and use parameters specifying the FWA of the output and the number of lines (or card images) of output. Both programs save and restore index registers 7 to 15, and both exit with $RET, with ABEOJ primed if a PP parameter specification exceeded PP boundaries.

The method of both programs is as follows: Upon entry, AEOJI, a special control indicator used at end of job (see output EOJ), is reset. The index registers are saved and, if PP is the user, subroutine ACHK is used to check the parameters with the PP boundaries. If the FWA exceeds PP boundaries, SPR and ABEOJ are primed (the former to provide an error message), and $RET executed immediately. If the FWA is within PP boundaries but the calculated last word is not, SPR and ABEOJ are primed and $RET executed immediately. If the parameters are in bounds, the program enters a loop, moving one unit of output at a time to the print or punch transmit buffer until the output request has been completed or the buffer has been filled. When the output is completed, buffer control is saved, index registers are restored, and $RET is executed. Whenever the transmit buffer is filled, the program uses a subroutine (CAOS3A for print and AWSCH for punch) to wait for the output tape to become available, to logically swap the transmit and write buffers by interchanging control words, and to write the full buffer on the output tape. In addition, the respective subroutines set a control word (AOP) so that the resulting interrupt will be taken by the proper table of exits (for print, CEXN; for punch, AWEXIT).

Figure 97 presents the control symbols used in the two programs.

The print and punch routines use parallel I-O tables of exits CEXN and AWEXIT, respectively. These tables cannot be entered at the channel signal return, since a CS is taken by the first level table of exits. The EOP and EE fixup routines for these tables are given in Figure 98, and the error return fixup is described with the error routines.

$SPR provides an additional feature. If the programmer wishes to use the TYPE card TITLE option, the output print program does the title line insertion. The print program does not insert titles for compiler output. The output program inserts titles in three different situations.

1. Insertion on line per page count basis.
2. Insertion if carriage control character is a restore character.
3. Insertion on line per page count basis when carriage control character is a restore character.

If insert on count is specified (these parameters are interpreted and placed in output locations by the JC4 functions) and a restore carriage control character is encountered, output resets the current line count to 1 and does not insert a title. If a title line is to be inserted, a double space control character is placed in the output buffer for the line that initiated the insertion.

Symbol			
Print	Punch	Meaning	Control
CBTA	ABTA	If set to 1, the logical transmit buffer is full.	Set when buffer is full, reset by buffer switch subroutine.
CBWA	ABWA	If set to 1, the logical write buffer is being written.	Set by the buffer switch subroutine, reset by the EOP fixup.
CATUB	CATUB	If set to 1, the output tape is busy.	Set by both buffer switch subroutines, reset by the EOP fixup. (Also used elsewhere in output.)
CCX.17	AACX.17	Flip-flop for control word selection.	Initially 1. Inverted by buffer switch subroutine.

Figure 97. Print and Punch Control Symbols

Table of Exits	EE Interrupt	EOP Interrupt
CEXN (Print)	If EOJ is in process (AEOJI set), EE is treated as EOP. Otherwise, the Trailer record is set to denote physical end of tape, and tape switch (ATSWCH) is entered with return set to the EOP fixup.	The output tape busy bit (CATUB) and the write buffer in use bit (CBWA) are reset. If the transmit buffer is full (CBTA=1), control is given to CAOS3A to initiate another write, and $RET is issued. Otherwise, $RET is given immediately.
AWEXIT (Punch)	Same as for CEXN.	Same logic as CEXN, using punch bits ABWA, ABTA, and the punch write routine, ASWCH.

Figure 98. EE, EOP Fixups for Print and Punch

Output EOJ

At end of job, the output program is given control to complete output for the job being terminated and to write a separating tape mark. Output EOJ is a major package with an associated tentacle table (SAEOJT). It is called by the sequence:

```
B, $MCP
 , SSPEOJ
```

during processing of $EOJ or $ABEOJ by Job Control.

The program starts at AEOJ (Figure 99) and operates enabled. It saves $7 to $15, and determines whether or not there is information in the punch transmit buffer. If there is, it is written on the output tape and control given to CEOJ. If not, control is given directly to CEOJ, where the process is repeated for the print transmit buffer, in this case filling the unused portion with blanks. Both programs test control bit CATUB to wait for an output in process to finish. At the conclusion of the print output, a tape mark is written. The flag, AOPTRI, is tested to determine whether or not an output command has been received for execution at EOJ. If not (AOPTRI zero), $7 to $15 are restored and $RET executed. If an output command has been received, control is given to the output command program at CDRY1.

The subsequent interrupt is taken by the I-O Table of Exits CEXEF. On EE or EOP, the tape busy bit (CATUB) is reset, and $RET issued.

Output Command

The COMD, OUTPUT is used to request termination and unloading of the current output tape at the next (or present) EOJ. The output command program receives control at CDRY (Figure 99) via its tentacle table, SCDRYT. It operates enabled.

Upon entry, index registers 7 to 15 are saved and the output end of job indicator, AEOJI, is tested. If the system is not at EOJ (AEOJI zero), the output command request indicator, AOPTRI, is set, the index registers are restored, and $RET is executed.

Control is given to CDRY1 if the system is at EOJ when the command is given, or by the output EOJ program when EOJ occurs. After waiting, if necessary, for an output in process to finish, a test is made to determine whether or not the current tape has been used. If not, a tape switch is not required, and the index registers are restored and $RET is executed. Otherwise, the output tape busy indicator (CATUB) is set, the trailer record is set to indicate physical and logical EOT, and control is given to the tape switch program to write one tape mark, the trailer record, and switch tapes. Return is set to $RET.

Tape Switch

The tape switch subroutine (ATSWCH) is used by all four output major packages: print, punch, EOJ, and command. It is entered by print, punch, and EOJ from the respective fixup routines on occurrence of an EE interrupt. It is entered by the output command program at EOJ.

There are two calling sequences used to enter the tape switch routine. To write two tape marks and switch tapes, the following calling sequence is used by the EE fixup routines:

```
LVI, $15, (return)
B, ATSWCH
```

The following calling sequence is used by the output command program to write one tape mark and switch tapes, since one tape mark has already been written when control is given to the output command program:

```
CM0000, ATMI
LVI, $15, (return)
B, ATSWC3
```

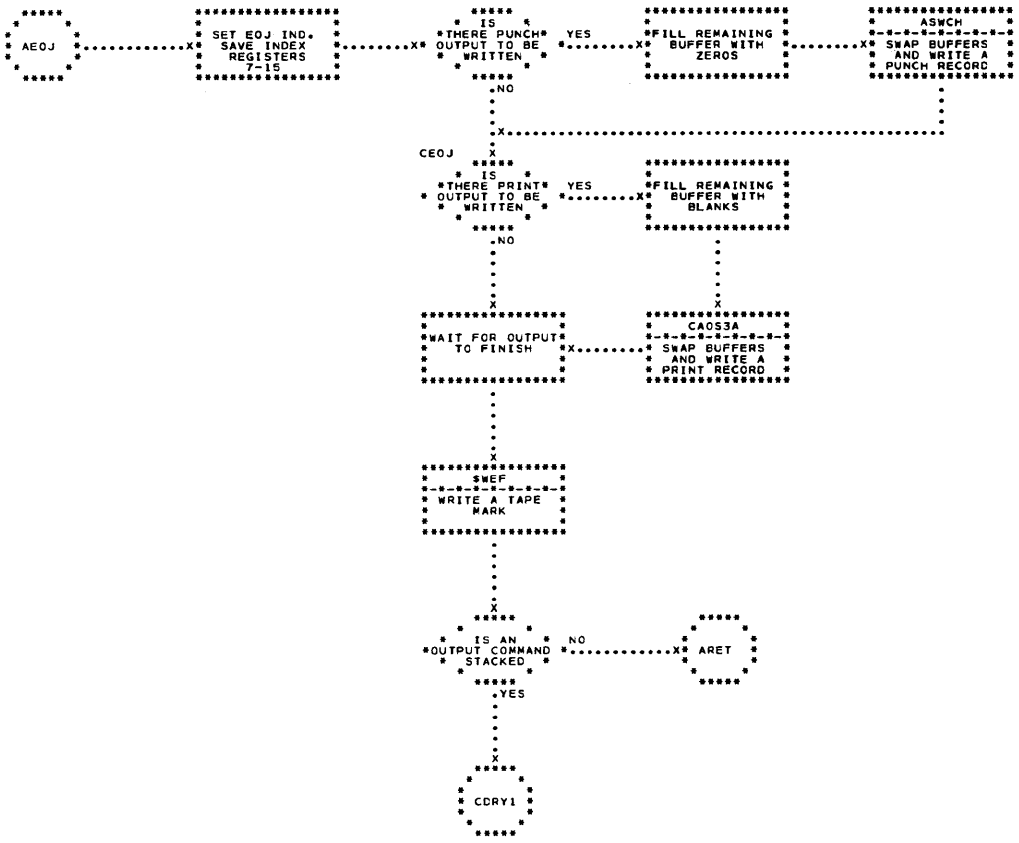
Bit 5 of the one word trailer record (CTRAIL.+.5) indicates whether or not the physical end of tape is also the logical end of tape (EOJ). On entry from the print and punch fixups, the bit will have been set to zero, indicating only physical EOT. On entry from output EOJ, and output commands, the bit will have been set to one, indicating logical EOT (EOJ).

After saving the return address, the routine sets ATSXIT in AOP as the current table of exits, writes a tape mark, restores index registers, and executes $RET. Depending on the entry conditions, this may or may not be in auto stack mode. The rest of the routine operates entirely in the auto stack mode.

Upon EOP, the control bit ATMI is reset and, if it had been set, another tape mark written. If it was not set, a new table of exits, ASTRA, is set into AOP, and the trailer record is written. (Note: For EPGK and UK fixups for tables ATSXIT and ASTRA, see description of the error routines.) $RET is then executed.

When the EOP occurs for the trailer record, the tape is unloaded. If another tape is ready, as evidenced by a CS having been received (ATARI or ATBRI=1), the corresponding IOD name (CT11 or CT21) is set into AMR as the current tape, the tape busy bit (CATUB) is reset, the new tape bit is reset (CXY+.19), and control returned to the return address. If a new tape is not ready, the tape switch request bit (ATSWCR) is set and $RET is executed. With the tape switch request bit set, the CS fixup routines (ATPACS, ATPBCS) will give control to the tape switch routine (ATSWC4 or ATSWC6) to reset the bit and complete the tape switch as previously described.

OUTPUT EOJ



OUTPUT COMMAND

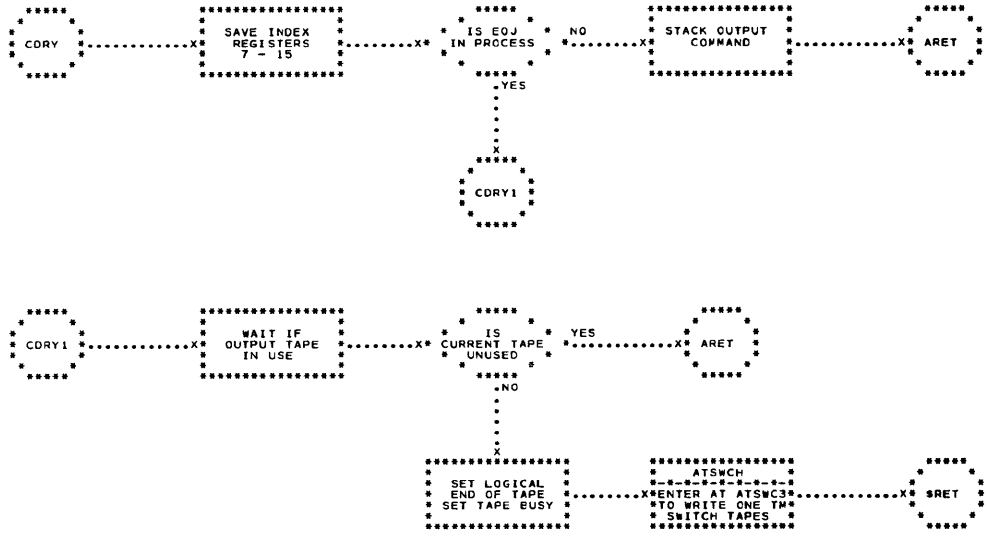


Figure 99. Output EOJ and Output Command Programs

The logic described for tape switch is dependent upon the following:

1. The tape ready bits, ATARI and ATBRI, are assembled as 0 and 1, respectively.
2. The initial tape selection is CT11, with I-O TOE AXITA.
3. Upon entry, and until completion of tape switch, the output tape busy bit (CATUB) is set. Thus, if a $SPU or $SPR is given, the corresponding programs will wait for a free output tape and will not destroy the contents of AOP, which specify a tape switch table of exits.

Error Control

A set of error control routines with their own I-O tables of exits is used when UK or EPGK interrupts occur in one of the five output program tables of exits previously described. These routines are:

Symbol	Occassion for Use	Table(s) of Exits
AEPGK	EPGK occurs on write	----
CKUK	UK occurs with EOP on write	CUKUK, CKWRIT
ALNUKC	UK occurs without EOP on write	----
AUKEF	UK occurs on WEF	ATSW1K
AERRUN	UK without EOP occurs on writing trailer record	----
ANFM	Announce unusual end of tape	----
AGFM	Announce unusual end of tape	----

The error routines are entered as follows for the various combinations of error interrupts and tables of exits:

	Interrupt		
Original TOE	EPGK	UK-not-EOP	UK- EOP
CEXN	AEPGK	ALNUKC	CKUK
AWEXIT	AEPGK	ALUNUKC	CKUK
ATSXIT	AUKEF	AUKEF	AUKEF
ASTRA	AERRUN	AERRUN	CKUK
CEXEF	AUKEF	AUKEF	AUKEF

AEPGK

The EPGK error routine, AEPGK, is entered with the actuated address in $2 when the interrupt is received by tables CEXN or AWEXIT. The EPGK is assumed to have been caused by a file protected tape being mounted on an output unit. A message is typed via the commentator, and the tape switch routine is entered at ATSWC5 to unload the tape and select a new one. Return is set to try the write operation again. In all other cases, EPGK is treated as UK-not EOP.

CKUK

The general UK routine is entered when UK and EOP (UK-EOP) occur on a write operation. It is entered with the address of the TOE in $1, and with $3 specifying the return to be used if retries are unsuccessful.

The routine tries the sequence $BSP, $ERGS, $W, using the original control word for the write. The $BSP is given with TOE CUKUK set in AOP, and the $ERGS and $W are given by the $BSP EOP fixup, which changes the TOE selected to CKWRIT. If this table receives an EOP or EE, it is passed to the corresponding entry in the original TOE. Otherwise, the $BSP is tried again. The sequence of retries will be attempted ten times. If the write is still not successful, control is given to the specified failure return. This return is ANFM if a print or punch record is being written, and C2FM if a trailer record is being written. These routines are discussed under error message routines.

AUKEF

This routine is entered if UK or EPGK occurs when attempting to write a tape mark. If EOP did not occur, control is given to AERRN. If EOP did occur, the routine will attempt the sequence $ERGS, $WEF, using TOE ATSW1K. On EOP or EE, control is returned to the corresponding entry in the original TOE. On ten consecutive failures, control is given to the failure return originally specified in $13. This return is A1FM if a WEF was being attempted by tape switch, and A0FM if the WEF was being attempted by EOJ.

Error Message Routines

In the event of failure to write, a message is printed via the commentator. The message specifies data or unit failure, the channel and unit, and the number of file marks on the end of the tape. Two routines may be used to print the message. Both print the message, and enter tape switch to unload the bad unit. However, they set different returns from tape switch.

ANFM.32: is used when a print or punch record could not be written. Tape switch return is set to retry the original write. It is entered via ALNUKC, and directly at ANFM.

AERRUN.32: is used when no further attempts will be made to accomplish the output operation. It sets the tape switch return to$RET. It is entered via A0FM, A1FM, C2FM, AERRN, and AERRUN.

THE DISK FETCH PROGRAM

The disk fetch program transmits named system material from the permanent read only storage area (PROSA) on the disk to a specified area of the user's core when the pseudo-op $FETCH is requested. The requested material does not have to start at an arc boundary and need not be an integral number of arcs.

The $FETCH pseudo-op requires the linkage:

```
B, $MCP
, $FETCH
(AX)DD(BU,48,6), TYPE AREA X
, RELFWA.(I)
, FWA.(J)
, N.(K)
, M
(Error Return)
(End Return)
(Normal Return)
```

where:

TYPE AREA is an eight-character name of a specific part of PROSA.

RELFWA.(I) specifies the first word within the type area (relative to the type area starting address) which is to be brought into core storage by $FETCH.

FWA.(J) specifies the first word (18 bit) address of core storage into which $FETCH will read the requested part of the type area.

N.(K) specifies the number of full words requested. If N.(K) is zero, $FETCH will read in all of the type area from RELFWA.(I) onward.

M will be set by $FETCH if and only if N.(K) words cannot be transmitted. M will then contain the number of words which were transmitted.

End Return is a half-word instruction to which $FETCH will return control with M set as above, when the type area is exhausted.

Error Return is a half-word instruction to which $FETCH will return with M set as above, when protected storage is violated or when a non-existent type area is called for.

PROSA Controls

The permanent read only storage area, PROSA, is written on the disk at IPL time from the master IPL tape prepared by the system update program. The first two type areas on the disk are the PROSA dictionary and index.

The dictionary type area consists of as many arcs as necessary to provide a two word entry for each type area (Figure 100).

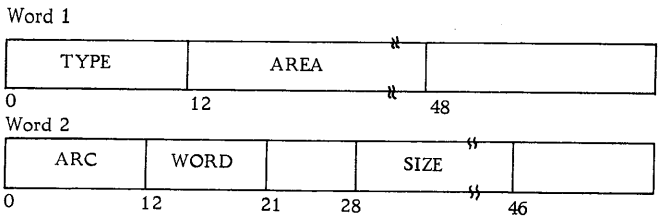


Figure 100. Typical Fetch Dictionary Entry

The first 48 bits of each dictionary entry are the type area name in A6 code. The second word of the dictionary entry holds the disk location (arc and word) of the first word of the type area, and the number of words in the type area. The dictionary is in alphanumeric order by type area.

The Disk Fetch Program

Performing a fetch from PROSA consists of the following:

1. Reading the PROSA dictionary into the MCP 512 word buffer in the overlay area.
2. Obtaining from the PROSA dictionary, the disk address of the type area.
3. Reading into and transmitting from the MCP buffer the initial non arc boundary portion of the requested type area.
4. Reading the remainder of the requested type area directly to core storage starting at the type area's first full arc boundary.

To improve the performance of $FETCH, the program does the following:

1. Attempts to find the type area in the special short dictionary to avoid reading the full dictionary from the disk.
2. Keeps a record of the contents of its 512 word buffer, transmitting from the buffer if the requested information is there from a previous read.
3. Reads directly into the user's core storage if possible, eliminating a subsequent transmit from the buffer.

The program is entered at CFETCH (Figure 101) from the IF analyzer with its tentacle (XTENB) set up. The program operates enabled and non-SIO. The short dictionary (to be discussed later) is set up by IPL. The following steps are executed for a $FETCH request.

1. Determine if the requested material is in the buffer from the last request. If it is, go to step 13.

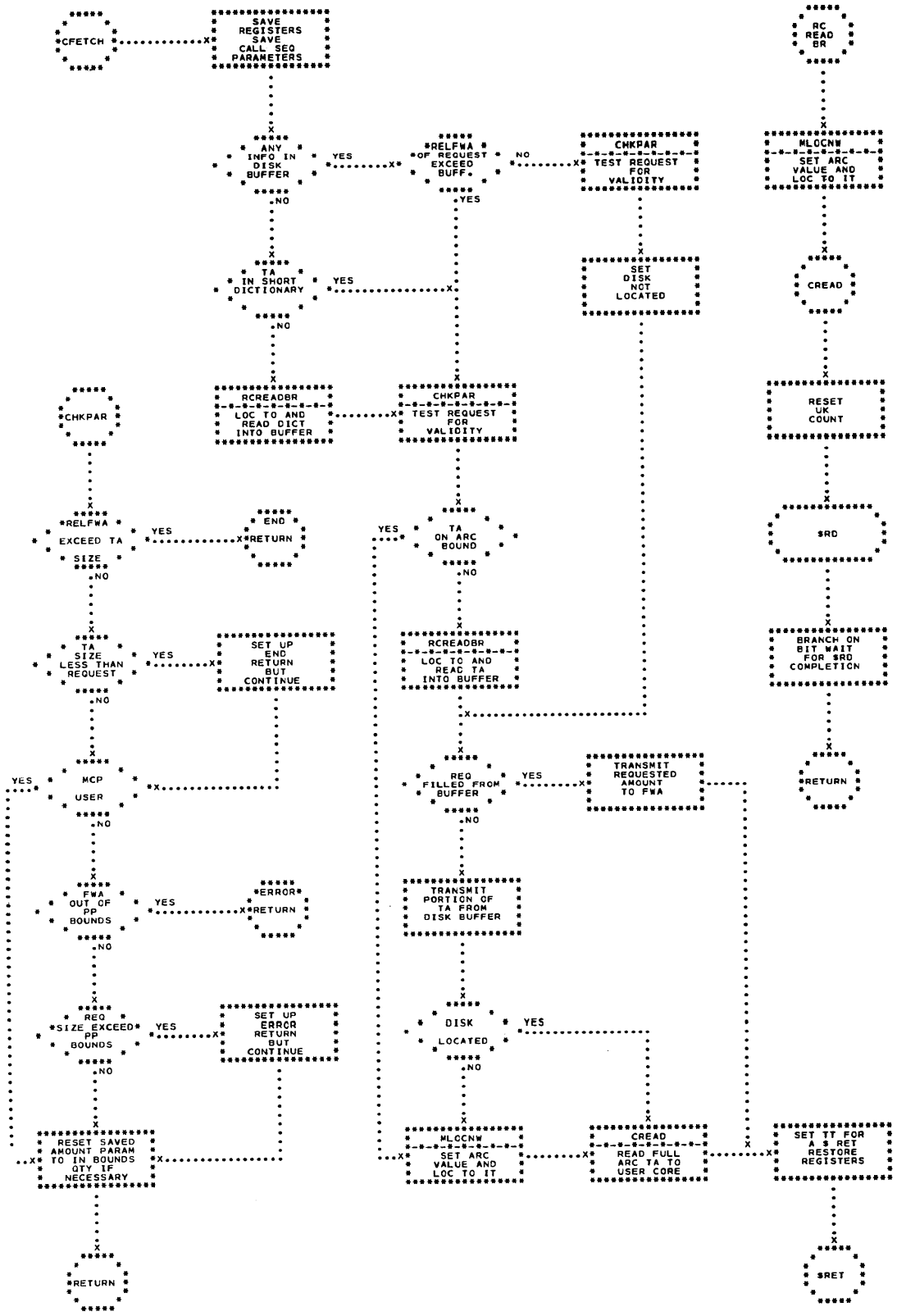


Figure 101. Disk Fetch Package

2. Scan the short dictionary for the requested type area entry. If it is there go to step 4. The short dictionary is filled at IPL time with the first 110 entries from the full dictionary. Generally, there are less than 110 entries, thus completely eliminating the $LOC and $RD time for each different type area's use of the full dictionary.

3. Read the full dictionary if the requested type area is not in the short dictionary.

4. Place the type area two word dictionary entry in CFOT and CFOT+1.0 for a possible later use by step 1.

5. The request parameters are checked for validity. There are five types of exits from the validity check routine (CHKPAR).

a. The RELFWA of the type area is greater than the type area -- immediate END RETURN, no transmission.

b. The FWA of the type area violates the problem program boundaries -- immediate ERROR RETURN, no transmission.

c. The requested count N is either 0 or greater than the type area size -- END RETURN is set up in the tentacle table and the parameters are changed so that the user gets the entire type area beyond RELFWA.

d. The requested count, N when added to the FWA violates the user's upper bound -- ERROR RETURN is set up in the tentacle table and the parameters are changed so that the user gets transmission up to the upper bound.

e. If the parameters are correct, the user gets exactly what is specified with NORMAL RETURN set up in the tentacle table.

6. Check for the requested RELFWA of the type area being on an arc boundary for direct read to core storage. If it is, go to step 17.

7. Read an arc into the disk buffer containing the first X words of the type area ($0 \prec X \leq 511$).

8. The X words read into the disk buffer may be sufficient to satisfy the $FETCH request, N. If it is, go to step 19.

9. A transmission to the user's core storage is done of the X words in the buffer.

10. If one of the $FETCH optimization attempts is partially successful as per steps 15 and 17, a $LOC must be done here.

11. The remainder (N-X) of the $FETCH request is honored by a $RD directly to the user's core storage.

12. The type of return to the $FETCH calling sequence has been set by CHKPAR (step 5 or 14). The registers are restored and $RET is issued.

13. $FETCH may be refetching the same type area but may not have the correct arc of the type area in its buffer. In this case, go to step 5.

14. The parameters for the $FETCH are checked via CHKPAR. (See step 5 for details.)

15. The material in the buffer may not be sufficient to cover the request. If the previous $FETCH required a $RD to core storage, the disk is improperly located. A bit is set for use in step 10. If the buffer can fulfill the request ($X \geq N$), control never gets to step 10.

16. Go to step 8.

17. Set parameters so that X effectively equals 0.

18. Go to step 10.

19. Transmit N words from the disk buffer to the user's core storage.

20. Go to step 12.

The service routines are those disabled MCP routines entered from the identifier which carry out service pseudo-op requests. They may be considered in seven groups:

1. Those used to actuate I-O operations.
2. Those used for symbolic I-O control.
3. Those used to impose mode controls on interrupts.
4. The commentator.
5. Time service operations.
6. Those used for transfer of control ($RET and $RAM).
7. The communication region transmission operations.

The routines in group six are functionally a part of the dispatcher, and are described with the dispatcher routines. The routines concerned with I-O activity use the set of I-O control tables previously described.

THE ACTUATOR I-O ROUTINES

The actuator routines provide I-O pseudo-operations corresponding to those provided by hardware instructions. They operate on logical units which are defined through IOD cards. The IOD cards set up the control tables as described in the preceding section.

When an I-O pseudo-op is requested, the actuator performs the requested operation and any setup operations required. The actuator always performs a locate tape setup operation since MCP does not provide the pseudo-op equivalent.

When the processing of a request must be delayed to await completion of a setup operation, control is usually returned via the service op return routine. However, the setup bit (SSETUP) is set in the UST, and SRETAD is preset in the UAT, to allow the processing of the request to continue when the interrupt occurs. Since these controls affect the unit, it is possible for several I-O requests for distinct units to be in various stages of processing at any given time.

The actuator I-O operations use a common group of subroutines to perform portions of request processing which are common to several pseudo-ops. All actuator programs operate disabled, and are entered from the identifier when the pseudo-op is requested, or from the receptor when an interrupt caused by a setup I-O operation occurs. For either entry, index registers 4, and 10 through 13 contains the following in the value field:

Index Register	VF Content
4	Equipment code
10	Address of channel status word
11	Address of unit status word
12	Address of unit area table
13	Address of file area table

When entry is from the identifier, index registers 0 through 3 contain the following:

Index Register	VF Content
0	Address of control word (if applicable)
1	Address of entry in I-O symbolic location table
2	Actuated address plus 2.0
3	Pseudo-op code

In addition, the identifier has stored the actuated address in the file area table (SACTAD), the actuated file address in the unit area table (SFIAAC), the SEOP bit in the unit status table (SSEOP), and has insured that the requested channel and unit are available. (See the Identifier Routine.)

The Read Routine

The read routine is entered at ZSTART (Figure 102). The control word is checked using the control word check subroutine (BCWCR), and SEVLMK and SEVLTB are set for subsequent entry to the status evaluation routine. The mask and table are designed to check unit select, label verification, and console signal. The channel operating bit (SCHOP) is turned on in the channel status table, and the routine branches depending on equipment type ($4), entering the status evaluation routine (ZSTEVL) for tape (if the verify buffer is idle, otherwise the PP is forced to loop until buffer becomes idle) and console equipment. For the card reader, the routine at KTESTM checks the mode and proceeds with the read instruction. For the disk, a special routine at ZDISK1 is entered. Any other equipment code is illegal, causing an EPGK interrupt to be faked by an appropriate entry to the receptor.

The routine at ZDISK1 is used by both the read and write routines. It tests the disk locate setup bit (SLOCSU) and the high-speed exchange bit (SHSECH). If either is on, the routine turns off the channel operating bit (SCHOP) and enters an enabled loop to the actuating address via KCBUSY. When both are off, the routine computes the arc address of the last

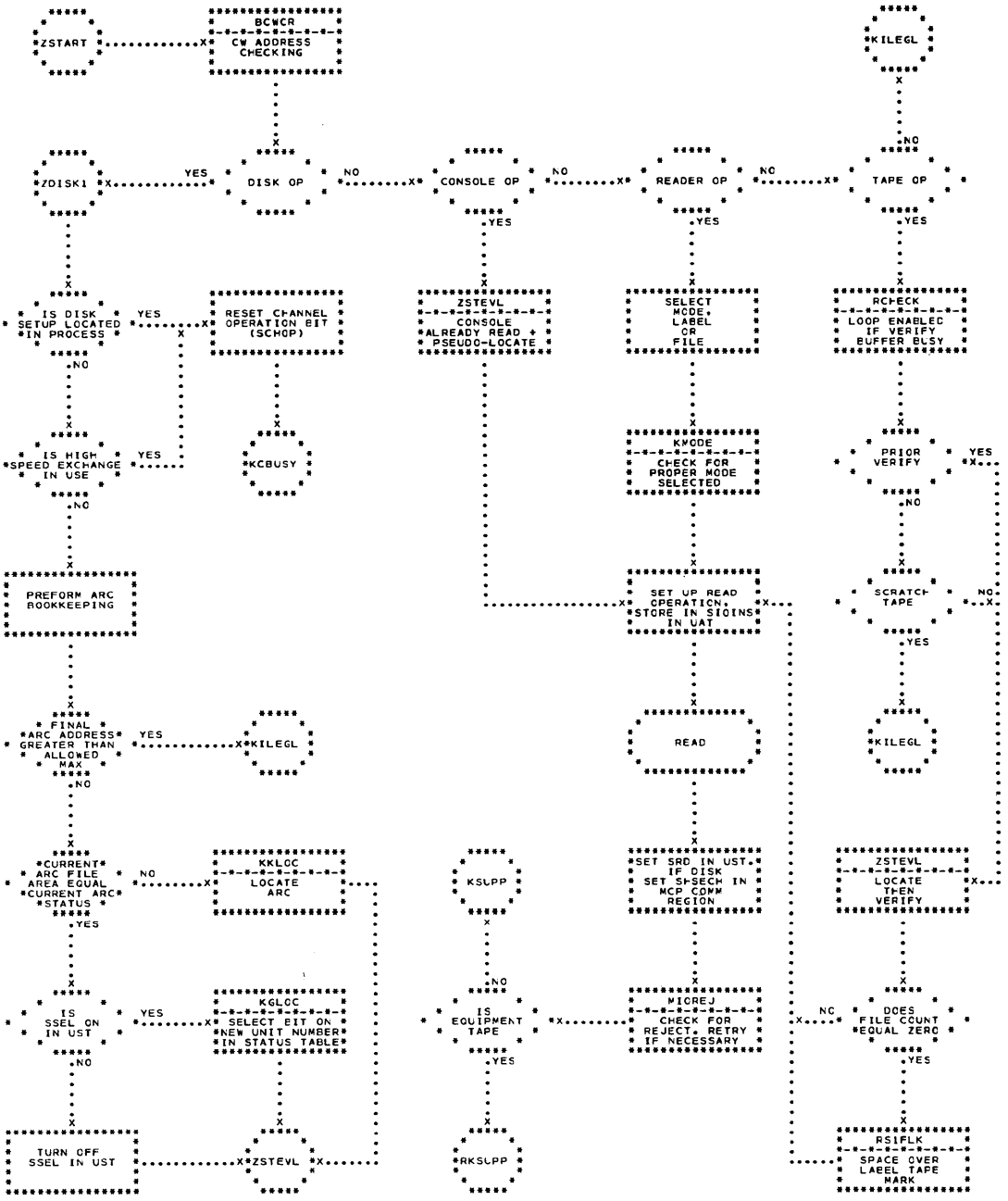


Figure 102. Read Routine

word to be transmitted to or from the disk, and if it exceeds the maximum assigned to the IOD, fakes an EPGK interrupt. If the disk arm is not properly positioned, the locate routine (KKLOC) performs a set up locate and ultimately returns to the status evaluation routine. Otherwise, the unit status table is updated if necessary, and control given to the status evaluation routine (ZSTEVL). The status evaluation routine will return to the instruction following the last branch in the branch table (SEVLTB). For tape equipment, after return from status evaluation, the tape label is passed if necessary. Then, for card and tape equipment, a check is made of the mode selected (KTESTM). The read is set up and performed, control bits adjusted, and control given to the I-O reject routine (MIOREJ) with return set to the service op return routine, KSUPP for non-tape, and RKSUPP for tape writes.

When a read request for the console is given in response to a console channel signal, a hardware read is not performed. The status evaluation routine finds the bit SCNSSG on in the unit status table and enters PCSRD, the routine specified by the branch table. The hardware read instruction was previously performed by MCP in order to identify the owner of the channel signal. The routine at PCSRD moves the information previously read according to the control word specified, turns off SCNSSG, and fakes an EOP interrupt by entering the receptor at KGATE. Since it does not return to status evaluation, control is not returned to the read routine, and a hardware read instruction is not performed. A console read request which is not in response to a channel signal will result in a hardware read instruction being performed (SCNSSG is off).

The Write Routine

The write routine is entered at EWR1 (Figure 103). The control word is checked using the control word check subroutine (BCWCR), and SEVLMK and SEVLTB are set in the unit area table for subsequent entry to the status evaluation routine. The mask and table for the write routine are designed to check unit select and label verification. The channel operating bit (SCHOP) is turned on in the channel status table, and the routine branches depending on equipment type ($4). For tape (if the verify buffer is idle) and console equipment, it enters the status evaluation routine and for disk equipment it enters the special routine, ZDISK1. (See the Read Routine.) If the equipment is the printer, control is given to EWR31, where the write is performed. If the equipment is the card punch, EMODE is entered to check the mode selected before giving control to EWR31 to perform the write.

When the control is returned from status evaluation, EWR31 is entered if the equipment is disk or console. If the equipment is tape, and the label specified protected, an EPGK interrupt is faked. If the tape is positioned at the label, the label is passed, and control given to EMODE. At EMODE, the file mode is checked and changed if necessary. For tape, an erase gap operation is performed, if one had been requested, before giving control to EWR31 to perform the write.

At EWR31, the write operation is set up and performed, and the control tables are updated. The I-O reject routine (MIOREJ) verifies acceptance of the write, and control is returned to the requestor via the service op return routine, KSUPP for non-tape writes and the label verify return, RKSUPP, for tape writes.

Copy Control Word

The copy control word routine is entered at BCCWR (Figure 104). If the requestor is PP, the specified address is checked to see whether it is a full word within PP bounds; if not, dispatcher error control is entered.

The control word is copied from the file area table under the following conditions:

1. The channel is not operating and the unit is not a card reader.
2. A non-disk channel is operating with a logical unit other than the requested one.
3. The requested unit is operating on a set up I-O instruction.

If the unit is a disk and the channel is operating, an enabled loop to the actuating address is set up via KCBUSY. If the unit is a card reader, or if it is the selected unit on an operating channel, a hardware copy control word instruction is executed. It is repeated twice in the event of exchange rejects. Control is returned to the requestor via the service op return routine, KSUPP.

Write Tape Mark

The write tape mark routine is entered at EWT1 (Figure 105). The routine adjusts the return address in STIC, and checks to see that the equipment is tape, faking an EPGK if it is not. An EPGK is also faked if the tape label has specified a file protected reel.

If the PP verify buffer is idle, the status evaluation routine is used to select the unit and verify the tape label, if necessary. A tape mark is written, first passing over the label file or erasing gap if required. Control is given to the I-O reject routine (MIOREJ) with return set to the label verify return, RKSUPP.

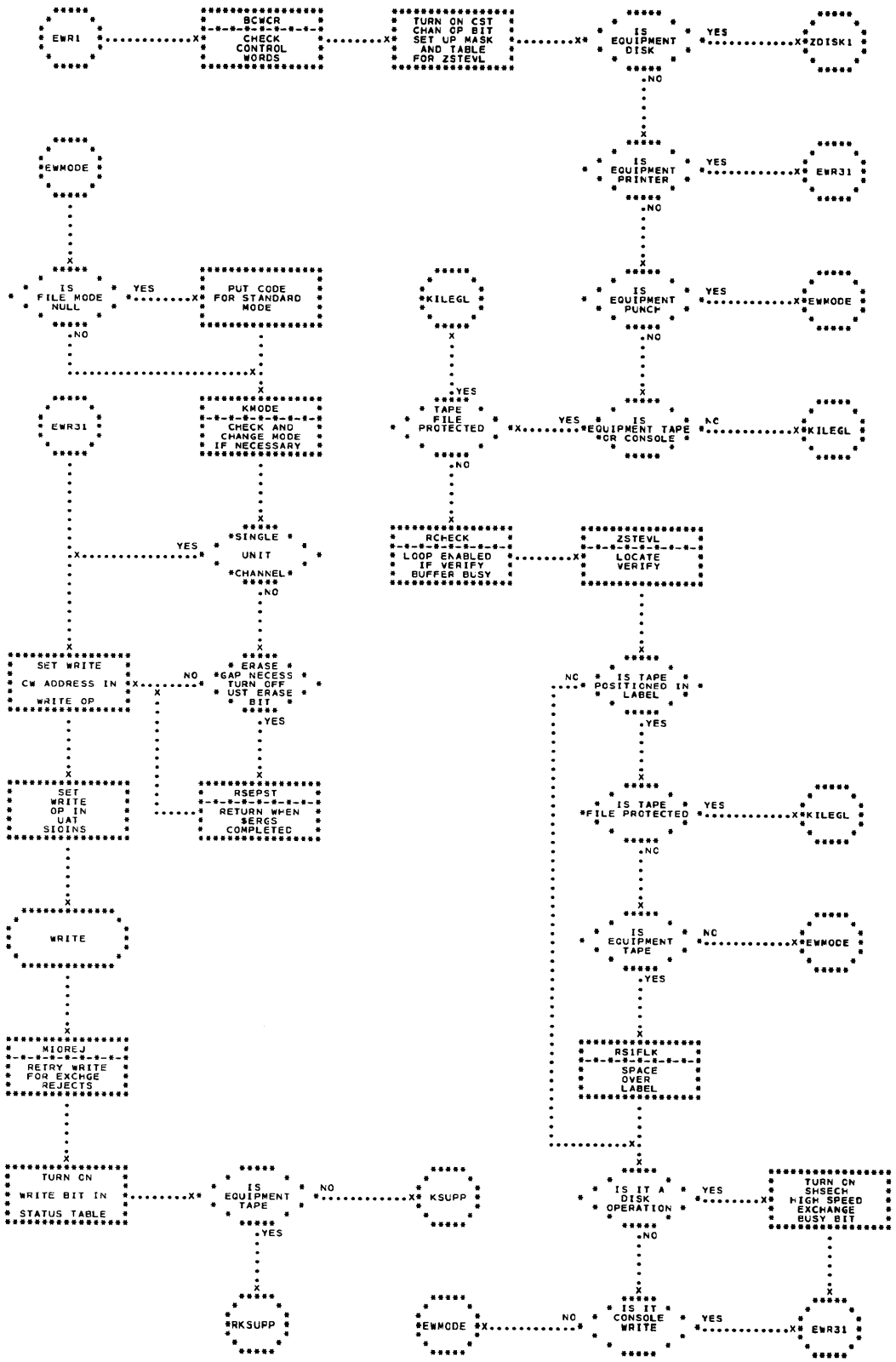


Figure 103. Write Routine

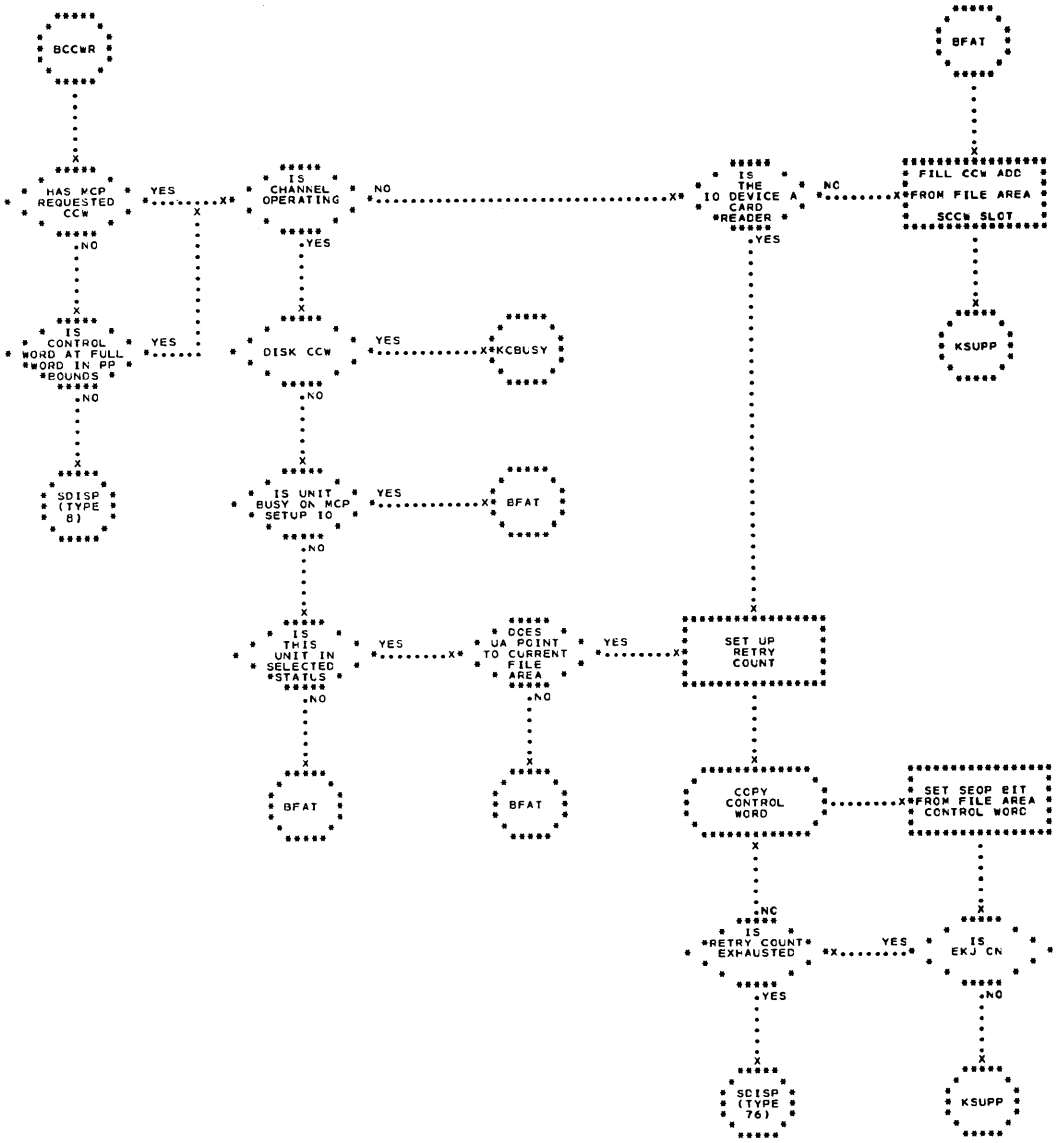


Figure 104. Copy Control Word Routine

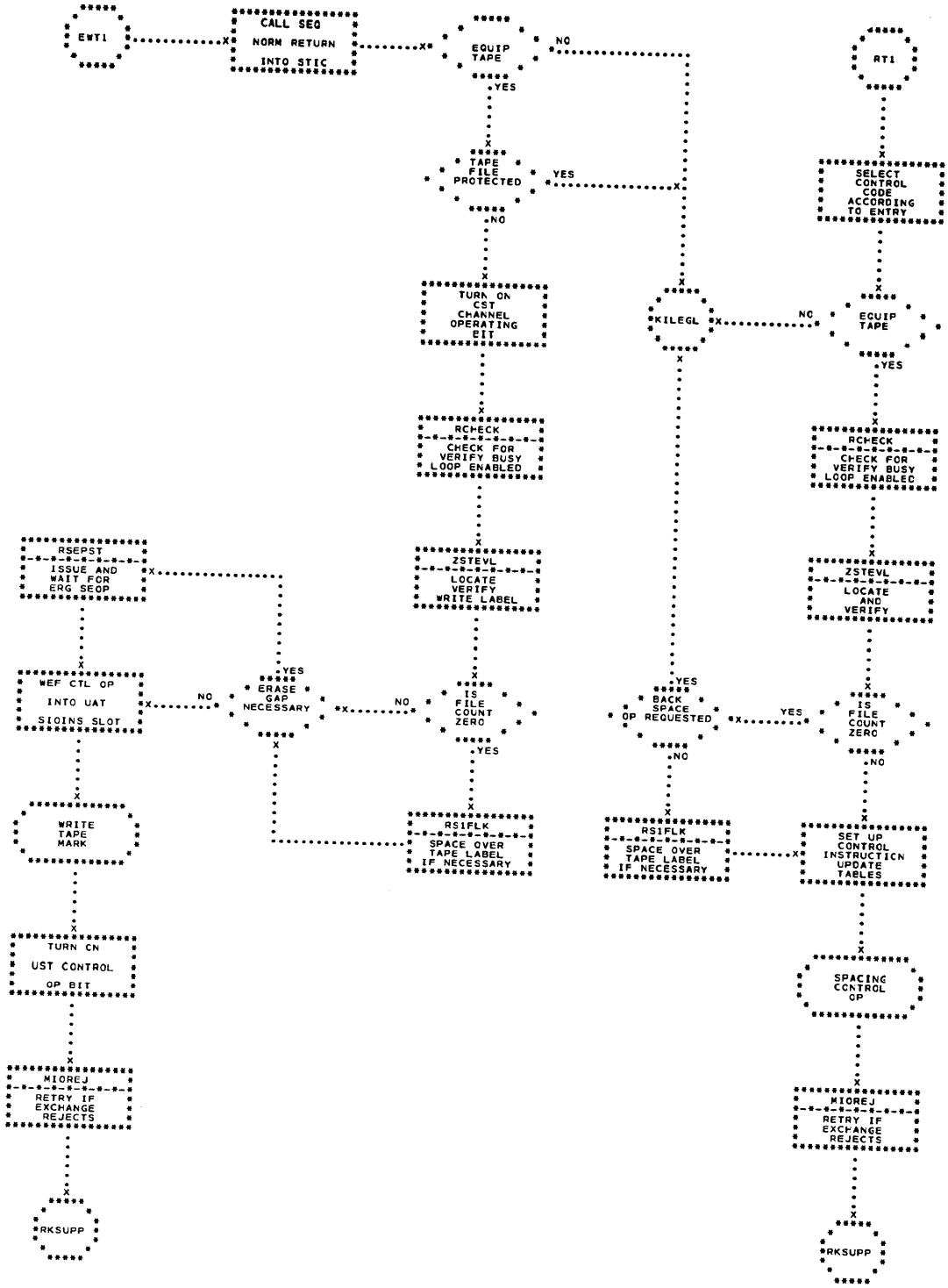


Figure 105. Write Tape Mark and Spacing Routines

Locate

The locate routine serves two distinct functions:

1. Locates the disk when requested by the locate pseudo-op.
2. Locates any unit required to perform another pseudo-op.

All locate operations for the disk are performed as non-SEOP set up I-O instructions. All locate operations for other equipment are performed as SEOP.

The Locate Subroutine, KLOCAT: The locate instruction is performed by the routine starting at KLOCAT (Figure 106). This routine performs a non-SEOP locate for disk equipment, turning on the set up I-O bit and presuming that SRETAD has been preset in the control tables, and finally exiting to KSUPP. For non-disk equipment, the KLOCAT routine performs a SEOP locate, waits for its completion in the SEOP test routine (KSEOPT), and exits to the preset return address.

The Locate Pseudo-Op: When the pseudo-op locate is requested, the routine is entered at KSLOC (Figure 106) from the identifier. This routine checks the legality of the request, faking an EPGK if the equipment is non-disk, or if the arc address is illegal. If it is disk and the arm is already positioned at the desired arc, it bypasses the locate and fakes a channel signal. Otherwise, it stores KMOON in SRETAD, and enters the KLOCAT routine.

When the interrupt from the locate occurs, the routine at KMOON is entered. It checks for a successful locate, attends to necessary bookkeeping, and fakes a channel signal by appropriate entry to the receptor.

Set Up Locate Instructions: The KLOCAT routine is used as a subroutine whenever a set up locate is required by another actuator routine. If the status evaluation routine is not being used, KLOCAT may be entered with the linkage SIC, KLOCTD; B, KLOCAT, with index registers set as usual for actuator routines and non-disk equipment. If the status evaluation routine is being used, the appropriate entry in the branch table should be B, KKLOC, where the routine at KKLOC sets the KLOCAT return to ZSTEVL, and sets SRETAD in the event the equipment is disk. If the equipment is not disk, KLOCAT will perform a SEOP locate, check it, and return to ZSTEVL. If the equipment is disk, KLOCAT will perform a set up non-SEOP locate, and return to KSUPP. The subsequent interrupt will return to KKLOC, where necessary disk bookkeeping will be performed prior to return to ZSTEVL.

Release

The release routine is entered at WREL (Figure 107). It adjusts the return address, and if the channel is operating with the specified unit in select, sets the release status bit, and returns via KSUPP. Otherwise, an EOP is faked for the release by appropriate entry to the receptor. The routine does not use a hardware release instruction. Release functions are performed by the receptor.

Feed Card

The feed card routine is entered at WFC (Figure 107). It adjusts the return address, and, if equipment other than punch is specified, fakes an EPGK by appropriate entry to the receptor. The routine updates the control tables, performs a feed card instruction, and returns via KSUPP after verifying that the exchange accepted the instruction.

Erase Long Gap

The erase long gap routine is entered at EELG (Figure 107). It adjusts the return address, and fakes an EPGK if equipment other than tape is specified. The routine sets the erase gap status bit, and exits to W23K in the release routine to fake an EOP. The erase gap instruction is not given until the next write is requested for the unit.

The Space Routine

The four spacing pseudo-ops are handled by one spacing routine. There are four entries to the routine, one at each full word starting at R1 (Figure 105). Each entry selects the proper control code for the pseudo-op requested, and enters the common routine (R2).

If the equipment is not tape, an EPGK interrupt is faked. If the PP verify buffer is idle, the status evaluation routine is used to locate the unit and verify the tape label, if necessary, and the tape position is examined. If the tape is positioned at the label and a backspace operation was requested, an EPGK interrupt is faked. Otherwise, the label is spaced over, and the spacing operation is performed. The control tables are updated, and control is returned via RKSUPP after verifying that the exchange accepted the command.

Unload

The unload routine is used to perform the unload pseudo-op, and is also used by the free and verify routines for unload set up operations. It is entered at ZUNLDB when the unload pseudo-op is requested;

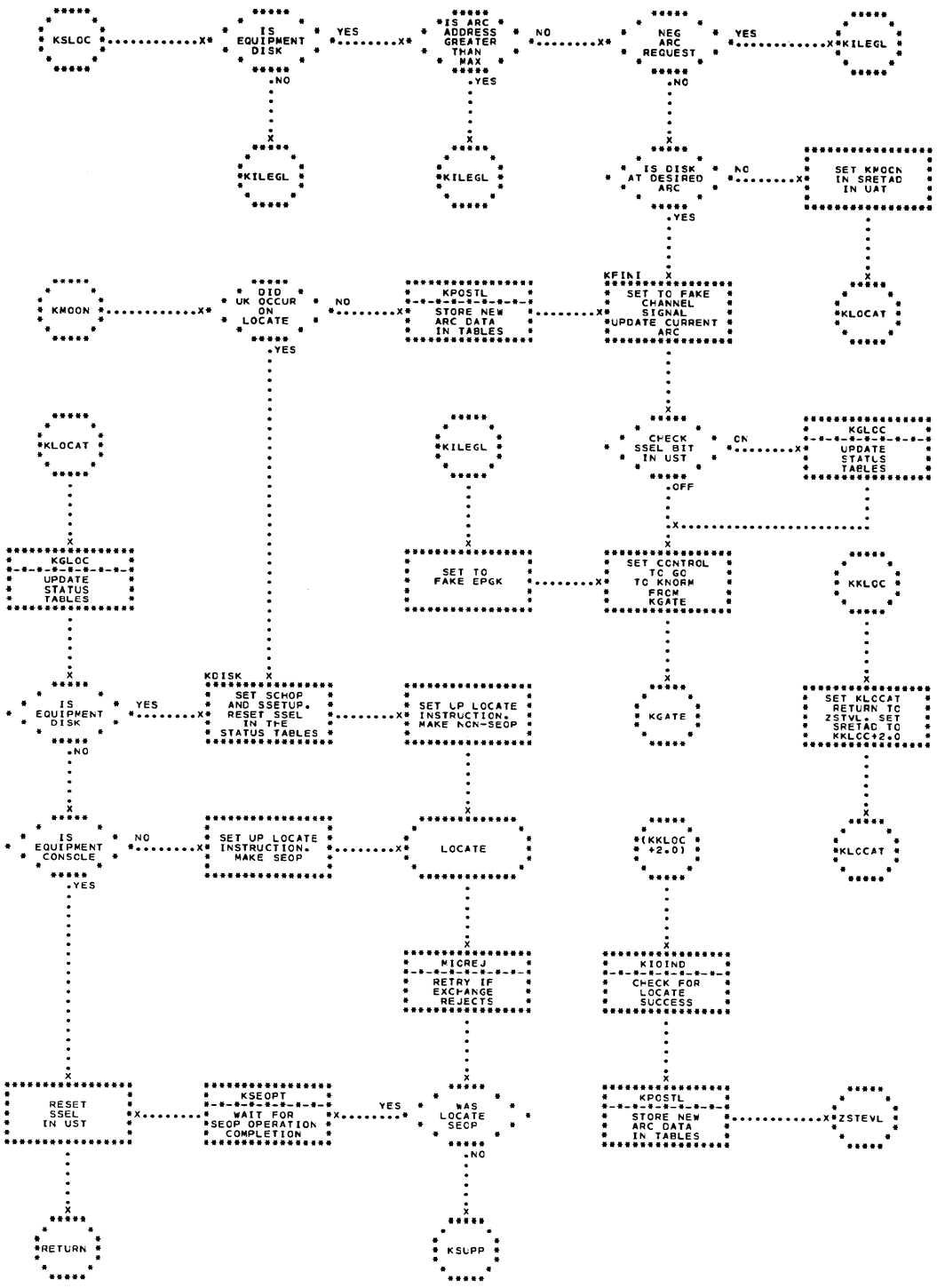


Figure 106. Locate Routine

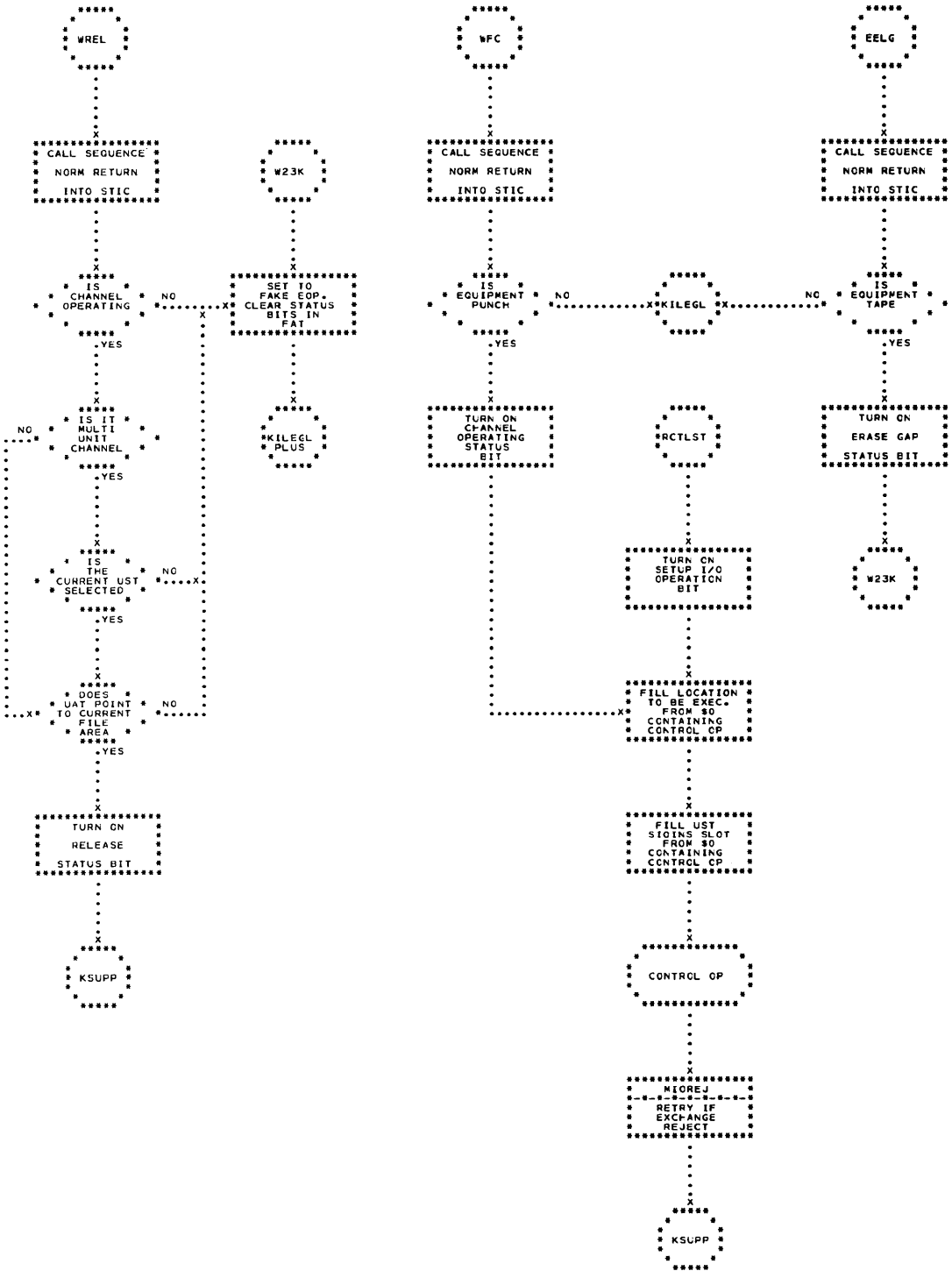


Figure 107. Release, Feed Card, Erase Gap Routine

at KRITET from the verify routine, and at KUNLD1 from the free routine (Figure 108).

The unload is always performed in the SEOP mode, since the EOP would normally occur when the exchange accepts the instruction. The channel signal occurring when a new tape is mounted is given to the user or the actuator, according to the SEOP and set up controls in effect.

When the unload pseudo-op is requested, the routine (ZUNLDB) uses the subroutine ZCOMRT to adjust the return address, check for tape equipment, adjust the control tables, and set up for and enter status evaluation. The SBDTA subroutine is entered to compute the location for this reel in the current reel on drive table. The SBCDP subroutine is used to determine the final disposition of the tape to be unloaded (by means of the specified disposition), the disposition setup in the current reel on drive table, the status of the job, and the status of the reel itself. The routine then sets up a mount message and updates the reel table. The message is modified (KRITET) to specify either the required reel number or a scratch tape. If the next reel request is for a scratch tape (SBSCRT), and the reel to be unloaded is a usable scratch (i.e., originally a scratch tape that is not saved), the reel is rewound via the rewind routine (ZREWST) and reused as the next scratch request instead of being unloaded.

If the new reel is to be file protected, the message is changed accordingly. The absolute channel and unit numbers are placed in the table (KUNLD2) and an unload (SEOP) performed. The I-O reject routine (MIOREJ) is used to perform retries if necessary, and the SEOP test routine (KSEOPT) entered to wait for EOP.

When the next reel request is established (SCBMNR), the current reel on drive table is updated to indicate the name and type of reel to be mounted on the specified channel and unit.

When the operation has been accepted, the mount message is typed and control returned via KSUPP. When the routine is entered at KRITET or KUNLD1 by verify or free, the setup bit and SRETAD must have been set, in order for control to return to these routines when the channel signal occurs.

The Alter Tape Disposition Routine

Entry is at SCABLT where STIC is updated. The check is made for a tape IOD, if not a tape IOD, error code 10 is given to the error control routine (SDISP). Otherwise, the new disposition is loaded and the three bit code is stored into the appropriate file area table. Return is made via KSUPP.

The Rewind Routine

The rewind routine is entered at ZREWST (Figure 108) when the pseudo-op is requested. After attending to bookkeeping, a rewind SEOP is performed. The I-O reject (MIOREJ) and SEOP test (KSEOPT) routines are used, and when the instruction is accepted, control is returned via KSUPP. When the rewind is complete, the channel signal will be given to the requestor.

Unit Lights

The unit lights routine is entered at separate points for each of the light pseudo-ops (Figure 109):

Pseudo-Op		Entry Point
$KLN	Check light on	BKLNR
$RLN	Reserve light on	BRLNR
$RLF	Reserve light off	BRLFR
$TIF	Tape indicator off	BTIFR

Each selects the appropriate control code and enters the common routine at BTFOK. The routine fakes an EPGK interrupt if tape equipment is not specified for the TIF operation, or if disk or tape equipment is specified for the other operations. The control tables are updated, the tape unit is selected if necessary, and the control operation is performed. Control is returned via KSUPP after using the I-O reject routine (MIOREJ) to verify acceptance of the instruction.

Gong

The gong routine is entered at WGONG (Figure 109). An EPGK interrupt is faked if equipment other than console is specified. The routine updates the control tables, performs the sound gong instruction, and uses the I-O reject routine (MIOREJ) to check acceptance. Control is returned via KSUPP.

Density Change

The density change pseudo-ops are allowed only on a tape at load point, otherwise, an EPGK is faked. If the tape is at load point, the density bits in the file area table and the verify bit in the unit status table are set, the PP return address is updated, and an EOP is faked.

Even and Odd Parity

These pseudo-ops are defined only for tape, but the tape need not be at load point. The file area mode bits

are set for later use, the PP return address is updated, and an EOP is faked.

NOECC Mode

The NOECC pseudo-op is defined only for reader or punch. A tape NOECC pseudo-op results in a faked EPGK. Either ODD or EVEN should be used for tape in this case. The mode bits on the file area table are set for later setup operations use, the PP return address is updated, and an EOP is faked.

ECC Mode

The ECC mode is defined for tape, reader and punch. Any other device causes a faked EPGK. The mode bits are set in the file area table, the PP return address is updated, and an EOP is faked.

Actuator Subroutines

Subroutines are used to perform functions common to several actuator routines. Most of the subroutines assume that index register 4 and 10 through 13 contain the same information as when control is received from the identifier.

Control Word Check

The read and write operations use control words, with the location of the first control word given in the linkage. These are hardware control words, and, for PP requests, must meet the following specifications:

1. They must be in PP storage when the operation is started.
2. They must not cause reading or writing out of PP bounds.
3. The chain bit in the last control word must be zero.

The control word check subroutine checks the control words specified, and makes the appropriate entries in the control tables. The control word check routine (Figure 110) is entered with the calling sequence SIC, B1RA; B, BCWCR, with the location of the first control word in the VF of $0. The routine performs the necessary checking for PP request, entering dispatcher error control with error code 7 in $14 if the control word(s) are not correct. At JPICK1, the control word information is placed in the control tables, and the routine returns to the preset address.

Status Evaluation

The status evaluation routine checks the status of the requested unit against a desired status provided by the requesting program. It is entered at ZSTEVL with the location of a status evaluation mask in SEVLMK in the unit area table, and the first location of a table of branch instructions in SEVLTB in the unit area table. The table of branch instructions consists of a half word branch instruction for each one in the mask. The status evaluation routine performs an AND operation with the half word mask and the second half word of the selected unit status table, and enters the branch table at a position corresponding to the first masked status bit which was on. Thus, the routine performs the specified status test, and if the masked status bits are not all off, enters the proper routine to correct the status. These routines perform the required function, turn off the status bit, and return to status evaluation. When all the masked status bits are off, the routine branches to the instruction following the branch table.

For example, the write routine requires that the unit be selected, and the tape label verified. It enters the status evaluation routine with EMK1A in SEVLMK and EWR18 in SEVLTB. These are defined as:

```
EMK1A    VF, (8)3000
EWR18    B, KKLOC
         B, RVFY
```

EMK1A has ones corresponding to the bits SSEL and SVER in the unit status table. Thus, if the unit is not selected, KKLOC will be entered to perform a locate, and if the label is not verified, RVFY will be entered, prior to performing the requested write. The status test for the various pseudo-ops is:

Pseudo-Op Requested	SSEL	SVER	SCNSSG
Read	KKLOC	RVFY	PCSRD
Write	KKLOC	RVFY	
WEF	KKLOC	RVFY	
Space	KKLOC	RVFY	
REW, Unload, Free	KKLOC		

Note that the control linkage used with the status evaluation routine permits returning control to the requestor after initiating a setup I-O operation. Since SRETAD is set in the unit status table on set up I-O operations, control will be returned to the proper unit by the receptor to continue processing the request. Furthermore, intermediate I-O requests on other units may use the status evaluation routine while a set up I-O is in progress.

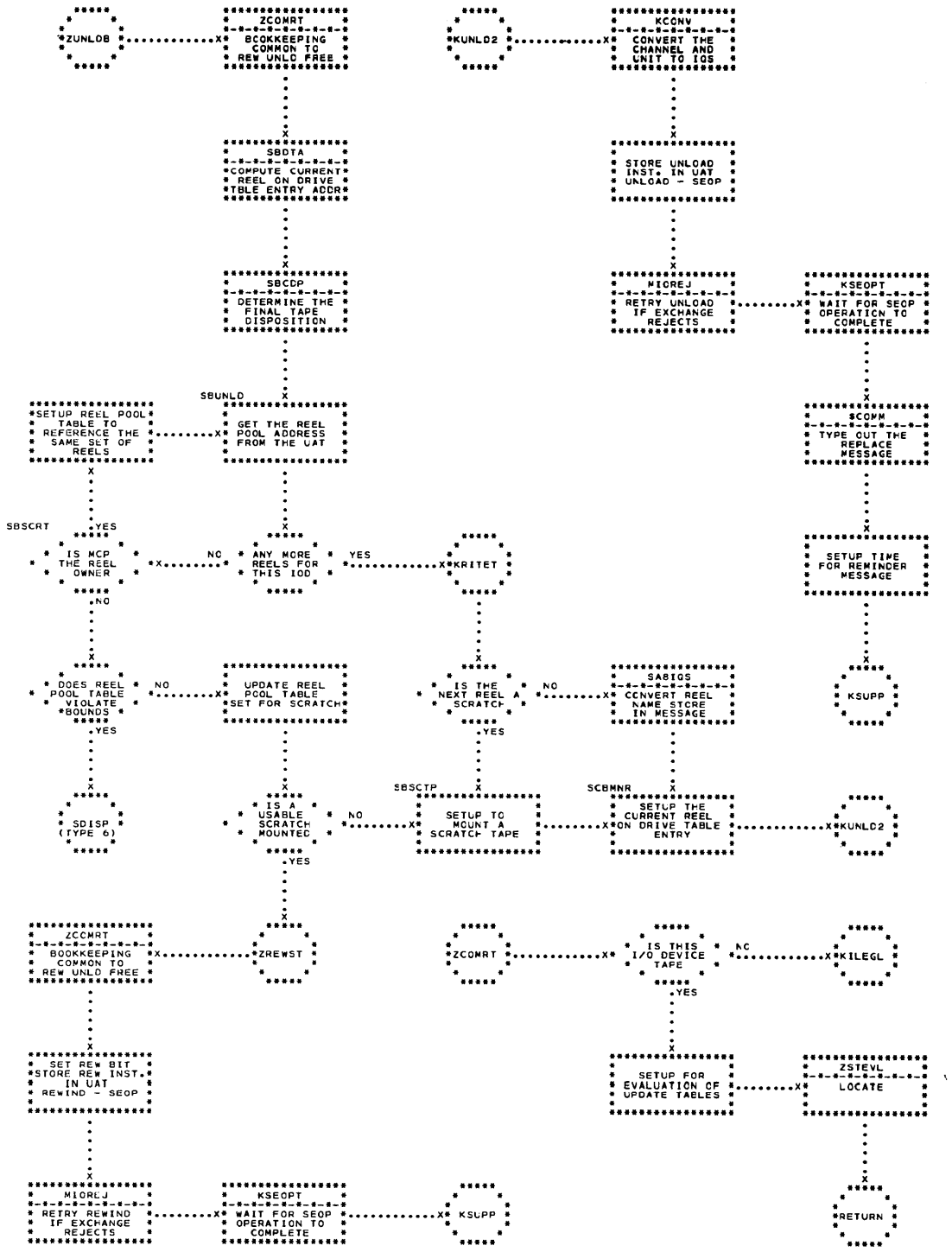


Figure 108. Unload and Rewind Routines

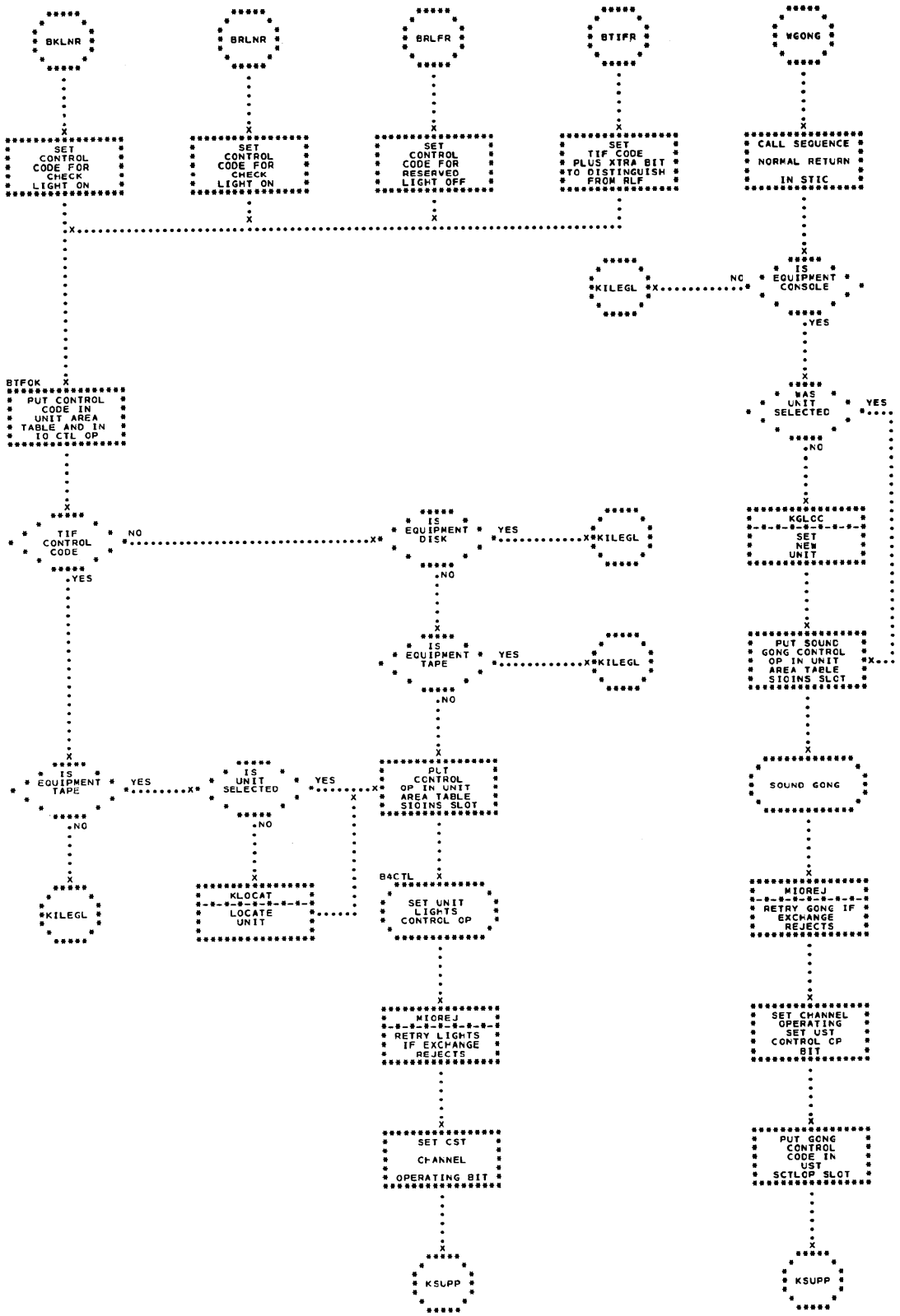


Figure 109. Unit Lights and Gong Routines

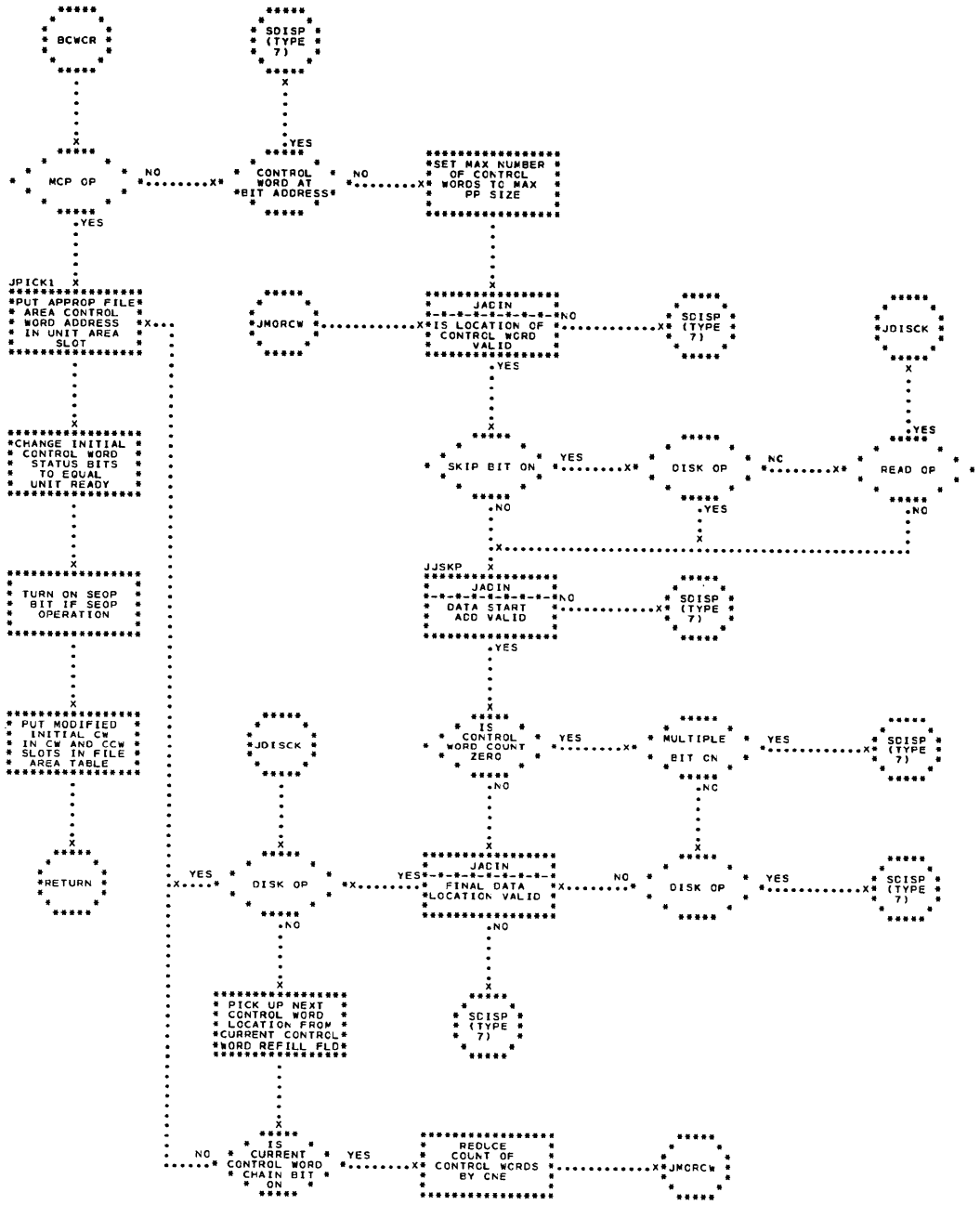


Figure 110. Control Word Check Routine

Verify

The verify routine is entered at RVFY (Figure 111) during status evaluation for read, write, write tape mark, and space pseudo-ops, when the verify bit in the unit status table is on. If the reel card specified an unlabeled tape, the label mode and density are set to the installation mode and density, the verify bit is reset, and control returned to status evaluation (ZSTEVL). The label mode and density must be set up even when there is no label, because they will be used in the event the IOD specification for mode and density is null.

If the tape is a labeled tape, a set-up read is performed and control is returned to the requestor via KSUPP. When the interrupt from the label read is received, the label is compared with the reel card specification for non-MCP units. If they do not compare (RV3SCR) and a scratch tape is being requested, the pseudo-op is examined. If it is write or write tape mark, the write label routine is entered (KWLABL). Otherwise, an EPGK is faked, since the first operation on a scratch tape must be a write. If a scratch tape is not being requested, the tape is unloaded and the operator requested to mount the correct tape. This will be done only once for a PP tape.

If the label compares, or if the tape is an MCP tape, the pseudo-op is examined. If it is write or write tape mark, control is given to KWLABL to write the label and ultimately return to status evaluation. If the mode specified by the label is null, the tape is unloaded and the operator is requested to mount another tape (KRELOD). If the label mode is not null, the unit area table is updated and control given to the space label at KSPTM with ZSTEVL set as the ultimate return in scratch pad memory (SSCRPM).

Write Label

The write label routine is entered at KWLABL (Figure 112). It is used to write a new label on a tape prior to the first write operation for that tape. It is assumed that the tape is positioned between the label and the following tape mark.

If the reel card specified a file protected tape, an EPGK is faked. Otherwise, the routine does three setup I-O operations: a backspace, a write label, and a write tape mark. It writes the label in its density and mode, and the tape mark in the density and mode specified for the file. It returns control, via KSUPP after each I-O operations, and when the tape mark has been written the routine returns to status evaluation (ZSTEVL).

Space Label

The space label routine is entered at RSIFLK (Figure 113). It is entered from the verify routine at KSPTM. The space label must be performed in two operations, one to space the label block, and one to space the label tape mark, since the tape mark is written in the file density which may differ from the label density. The sign of the file count is used to distinguish whether the tape is positioned at the beginning (file count minus zero) or end (file count plus zero) of the tape label. If the label must be passed, a set up I-O space block operation is performed in the density of the label, and control returned via KSUPP. When the interrupt occurs, the density is changed to the file density (KSPTM) and a set up I-O space file operation performed. When the interrupt for the space file occurs, control is returned to the location specified in scratch memory in the UAT, SSCRPM.

Set Density

The set density routine is entered at KDENST (Figure 112) with the desired density code in the VF of $1 and the return address in SRETAD. The routine returns immediately if the unit is in the desired density. Otherwise, a set up change density operation is performed and control returned via KSUPP. The density codes are:

1	Low
0	High

Change Mode

The change mode routine is entered at KMODE (Figure 112) with the code for the desired mode in the VF of $1 and the return address in KKMODE. If the unit is set to the desired mode, the routine returns immediately. Otherwise, a SEOP change mode instruction is set up and performed, and the SEOP test routine, KSEOPT, used to wait for its completion. Control is then returned. The mode codes are:

01	ECC odd
10	No ECC even
11	No ECC odd

I-O Reject Test

The I-O reject test routine is entered at MIOREJ (Figure 114) with the return address in $1 and the I-O instruction in SIOINS in the unit area table. The function of the routine is to test indicators EKJ, UNRJ and CBRJ after an I-O instruction to determine its acceptance, and to retry it if it is rejected. The

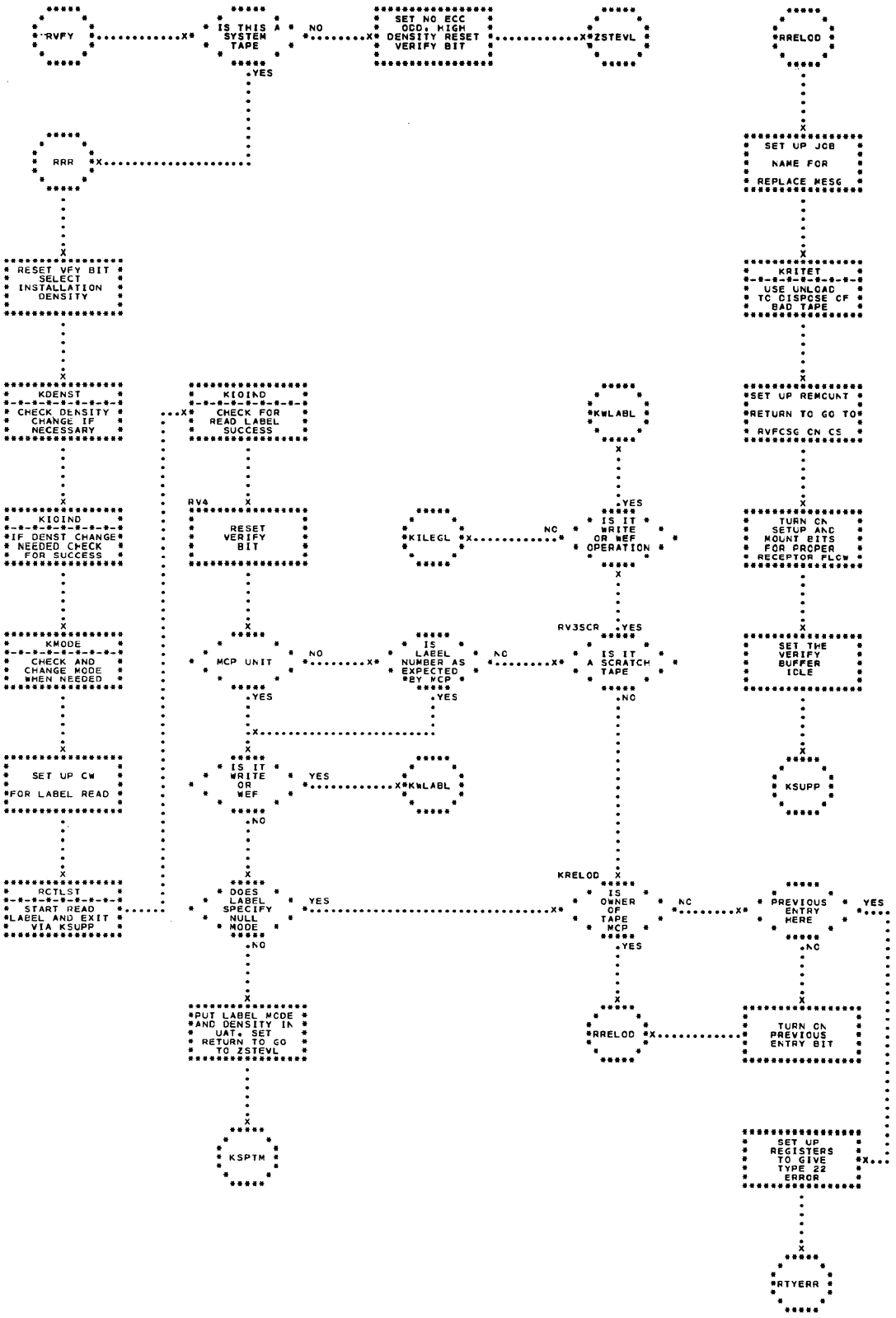


Figure 111. Verify Routine

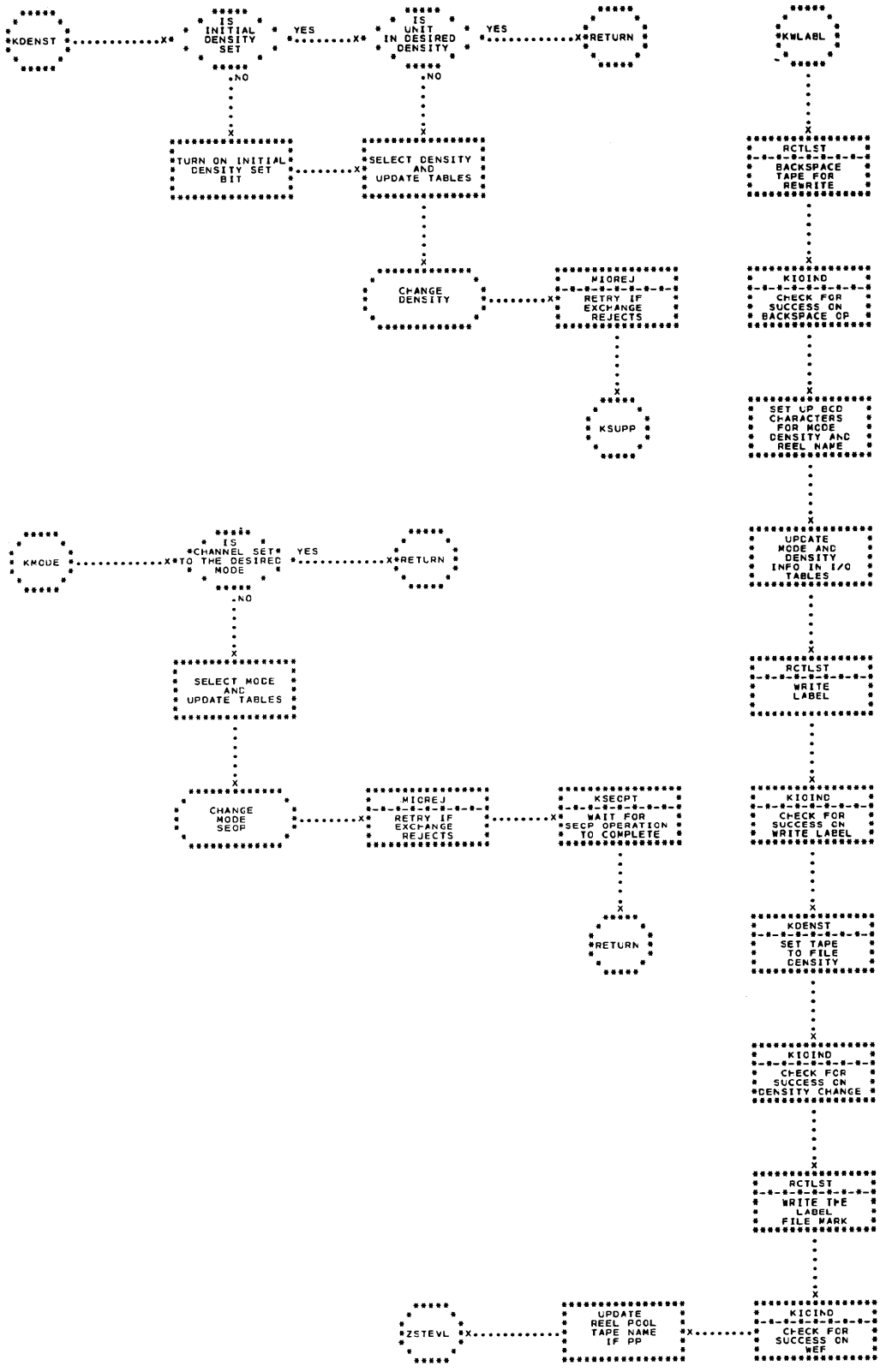


Figure 112. Check Density, Check Mode and Write Label Routines

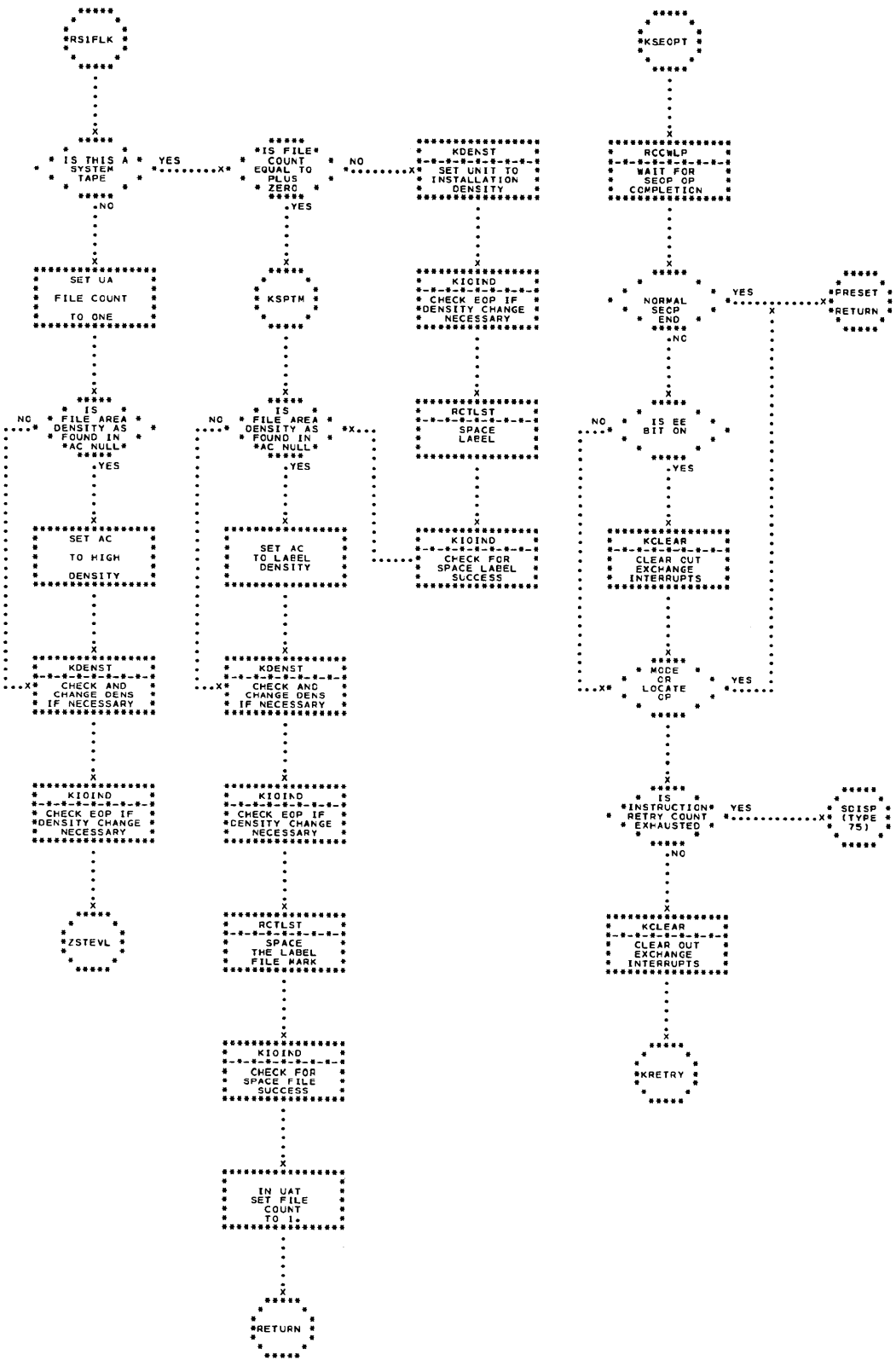


Figure 113. Space Label and SEOP Test Routines

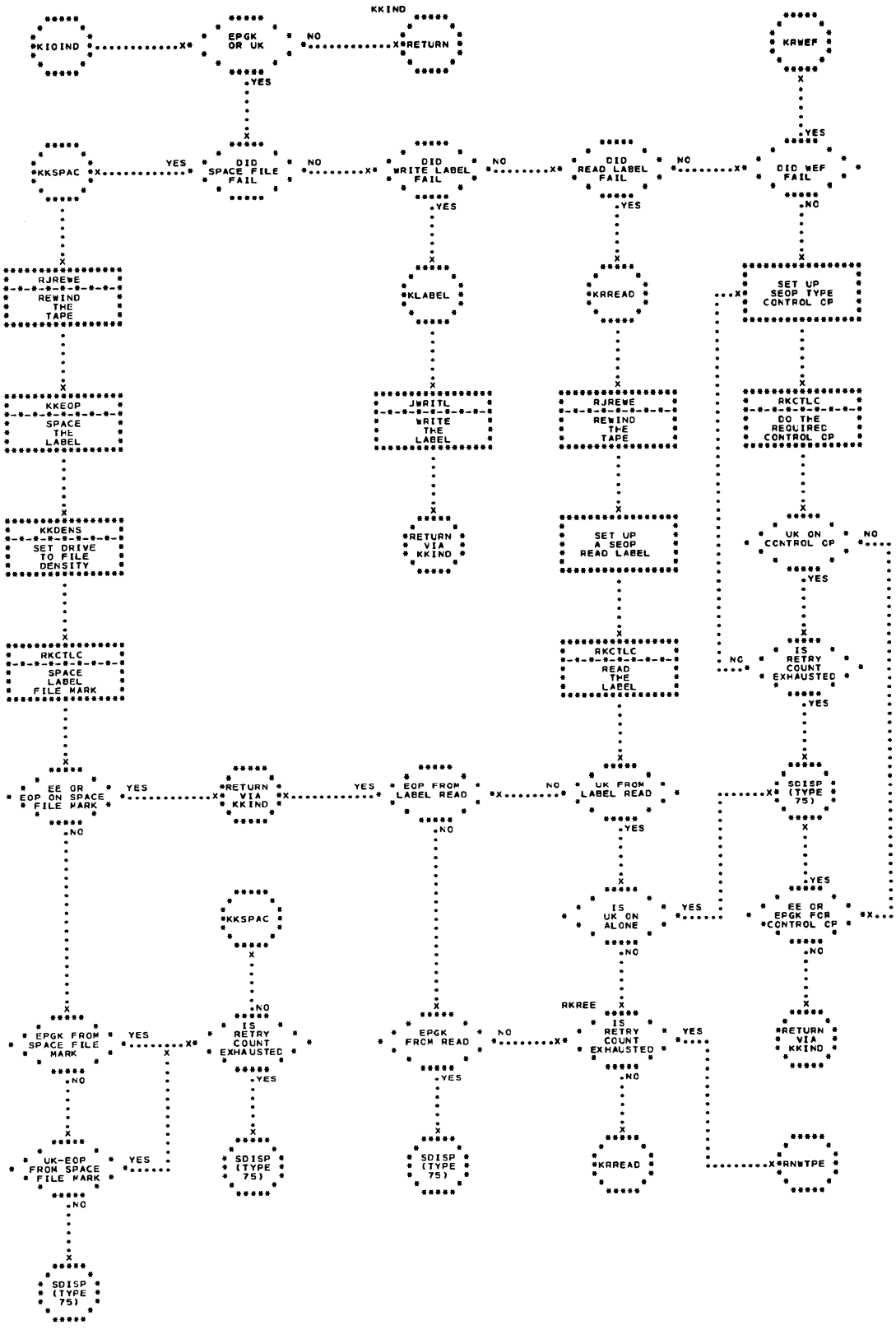


Figure 114. Disabled Service Routines

routine will return to the address specified if the instruction is accepted. If the instruction is not accepted after a number of retries, dispatcher error control is entered.

When the MIOREJ routine is used after set up I-O instruction, it is always followed by the SEOP test routine (KSEOPT) for SEOP I-O instruction, and by the I-O indicator check routine (KIOIND) for non-SEOP set up I-O instructions. The following code might appear following a SEOP set up I-O instruction:

```
LVI, 1, KSEOPT
SIC, KEOPR
B, MIOREJ
(return at EOP)
```

for non-SEOP setup I-O, this code might appear:

```
        LVI, 1, KSUPP
        SIC, SRETAD($12)
        B, MIOREJ
A       SIC, KKIND
        B, KIOIND
```

In this case, control is returned to KSUPP by MIOREJ and to A by the receptor when the interrupt from the set up I-O occurs.

EOP Test for SEOP I-O

The SEOP test routine (KSEOPT, Figure 113) is used to wait for completion of a SEOP set up I-O operation. The routine is entered with the linkage:

```
SIC, KEOPR
B, KSEOPT
```

or its equivalent (see the I-O reject routine, MIOREJ). The routine copies the control word repeatedly until the SEOP bit is off, returning when all indicators are off. If other indicators are on when SEOP is off, the channel is cleared and the operation repeated, unless the indicator is EE on a locate or a mode change operation. Since the EE indication results when these operations are performed while the tape indicator is on, the EE indication is ignored.

If it is necessary to repeat the operation, control is given to KRETRY with the VF of $1 still containing the return address just used by the I-O reject routine MIOREJ. Thus, the operation will be repeated and KSEOPT entered again. A count is set up by MIOREJ in $11 to limit retries. The counting is done at ZTSEOP.

I-O Indicator Check

After a non-SEOP set up I-O operation, the I-O indicator check routine (KIOIND, Figure 115) is used to test the EPGK and UK indicators. If both are off, it returns; if either is on, the routine determines the nature of the I-O operation and initiates a series of retries, ultimately returning or entering dispatcher error control. The five main paths (Figures 115, 116) through the routine are:

1. KKSPAC Space file failure
2. KLABWL Write label failure
3. KRREAD Read label failure
4. KRWEF Write tape mark failure
5. KREPET Locate or change density failure

Each path uses some of a set of specialized small routines to perform the retry. The routine at RNWTPE (Figure 117) is used if it is necessary to unload a tape and try a new one. The routine at JWRITL (Figure 117) is used when a write label operation fails, or when a rewrite of the label is required. The short subroutines have the following functions:

RJREWE	Rewind the tape over the label to the load point.
KIOREJ	Retry an I-O operation and verify its acceptance by the exchange and unit.
RSEOPT	Continually copy control word until the SEOP bit is off (all retries are SEOP); then check the indicators and return accordingly.
KKEOP	A common routine to retry control operations.
KKDENS	Verify that the unit is in the desired density, and change density if necessary.

In addition, the subroutines, KKLOOP, KUNR, and KCLEAR in the I-O reject routine (MIOREJ) are utilized.

In attempting retries due to an initial EPGK or UK, there are many occasions for additional errors to be detected. In general, error indications are tested throughout a retry attempt, and dispatcher error control is entered if the original or any subsequent operation cannot be successfully performed.

SYMBOLIC I-O CONTROL ROUTINES

Three service pseudo-ops are provided for flexibility in the use of symbolic I-O:

```
$IODEF -- I-O definition report
$CHEX -- Change table of exits
$FREE -- Free a tape unit for other assignment
```

All are entered in the disabled mode from the identifier with the index registers set as specified in table AM (The Identifier Routine - Interrupt Control).

I-O Definition Report

The I-O definition report routine (JZIOR, Figure 118) obtains the storage address from the linkage, converts the channel and unit numbers to IQS, and stores

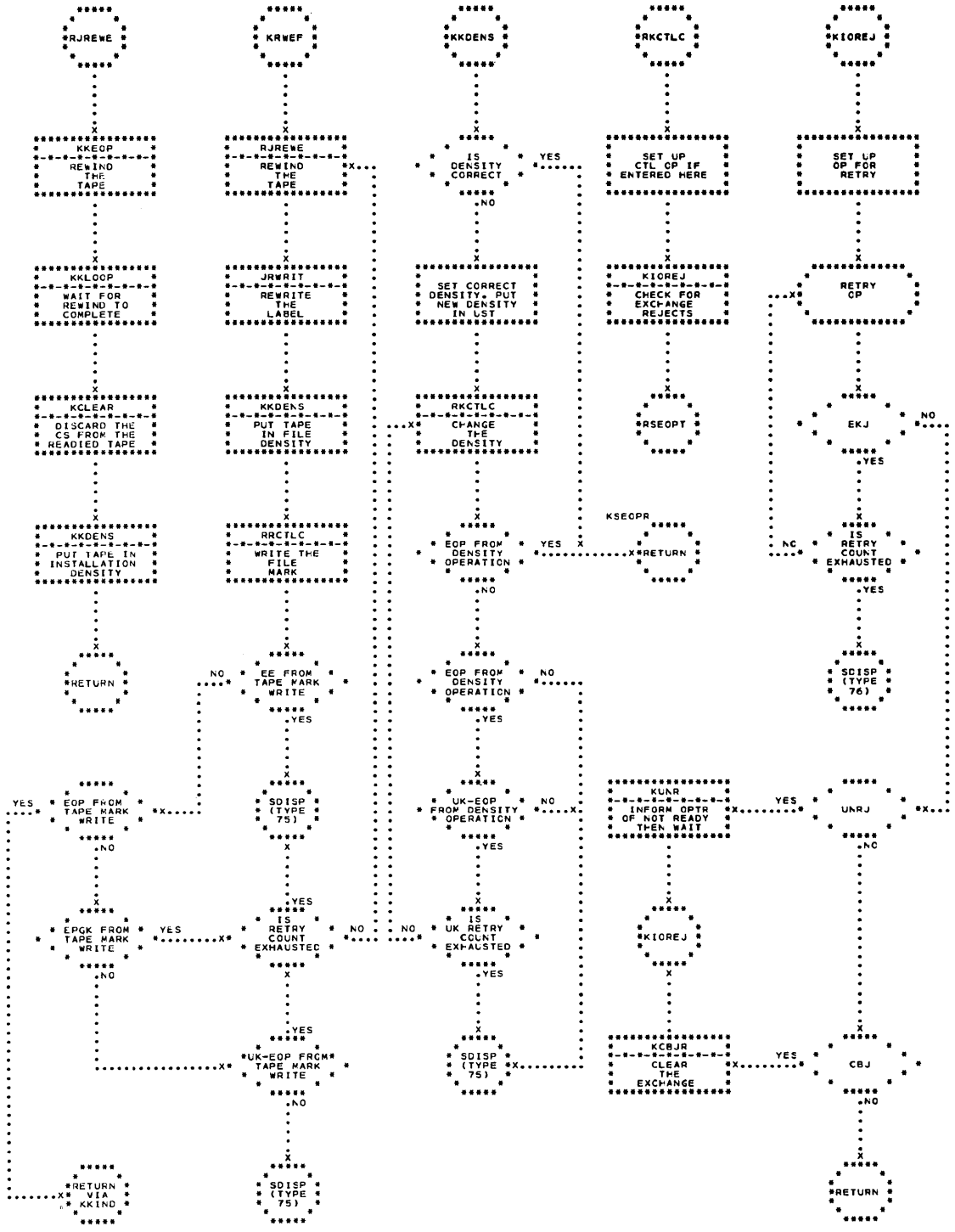


Figure 115. I-O Indicator Check Routine - Chart 1

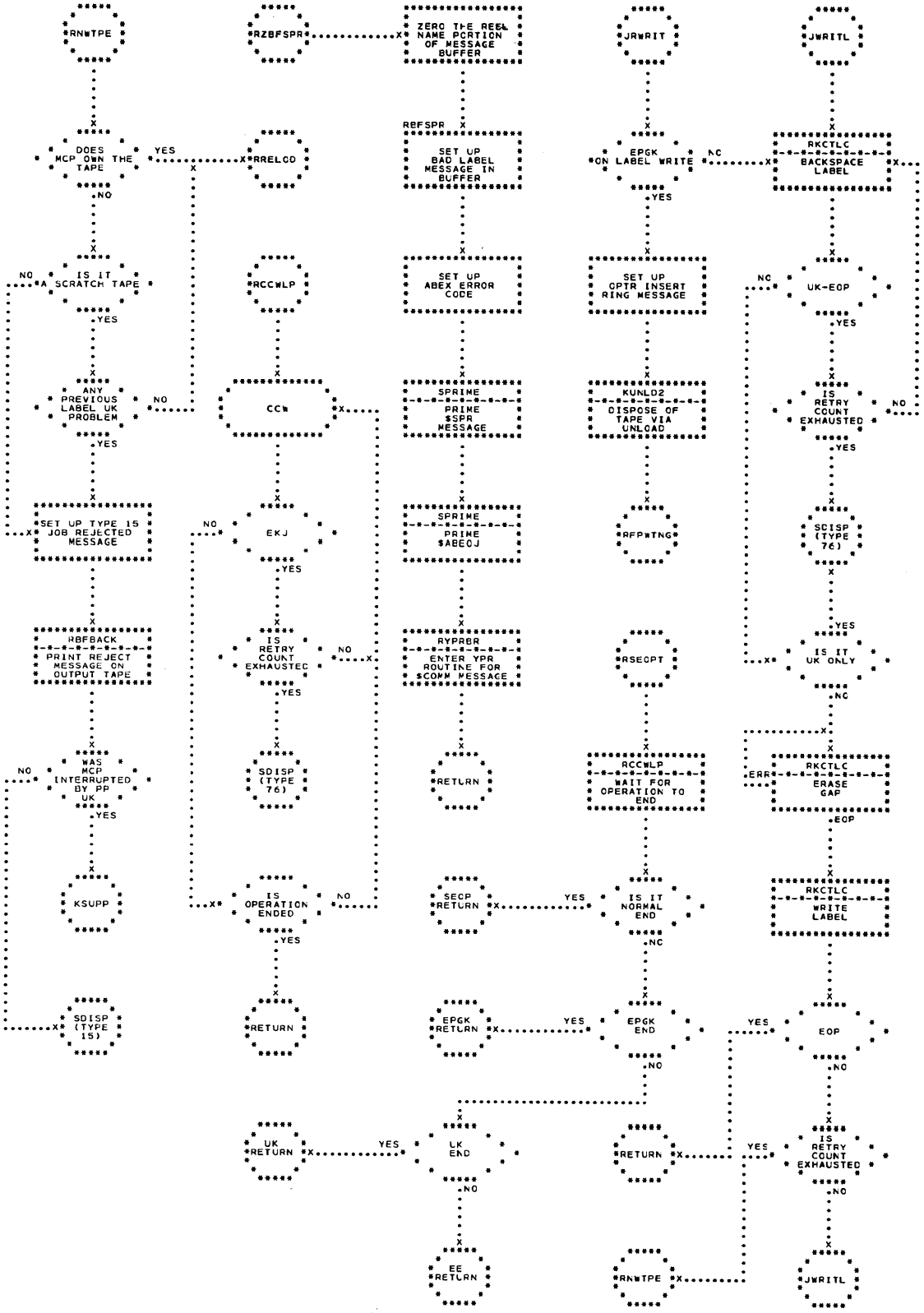


Figure 116. I-O Indicator Check Routine - Chart 2

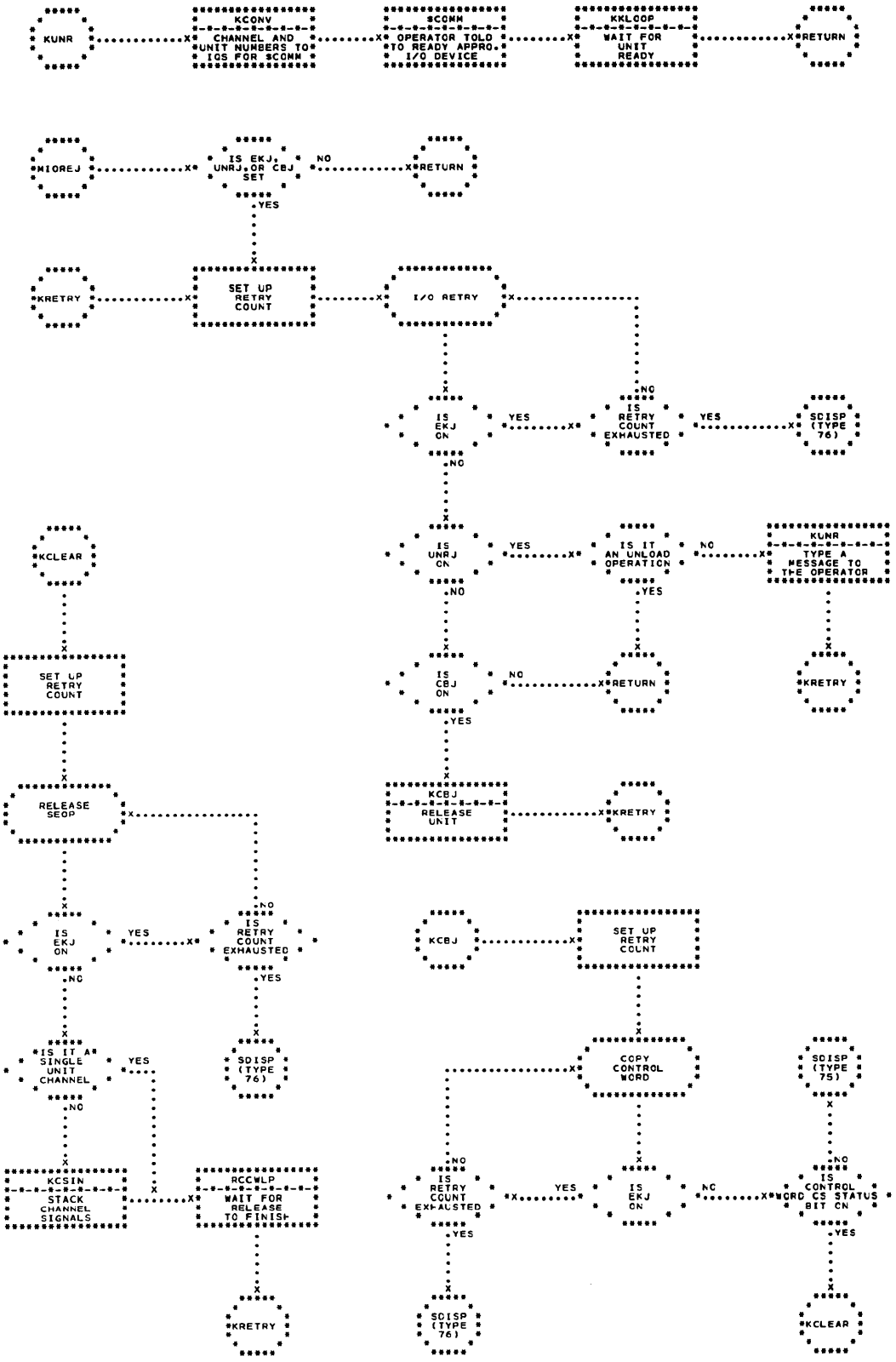


Figure 117. I-O Indicator Check Routine - Chart 3

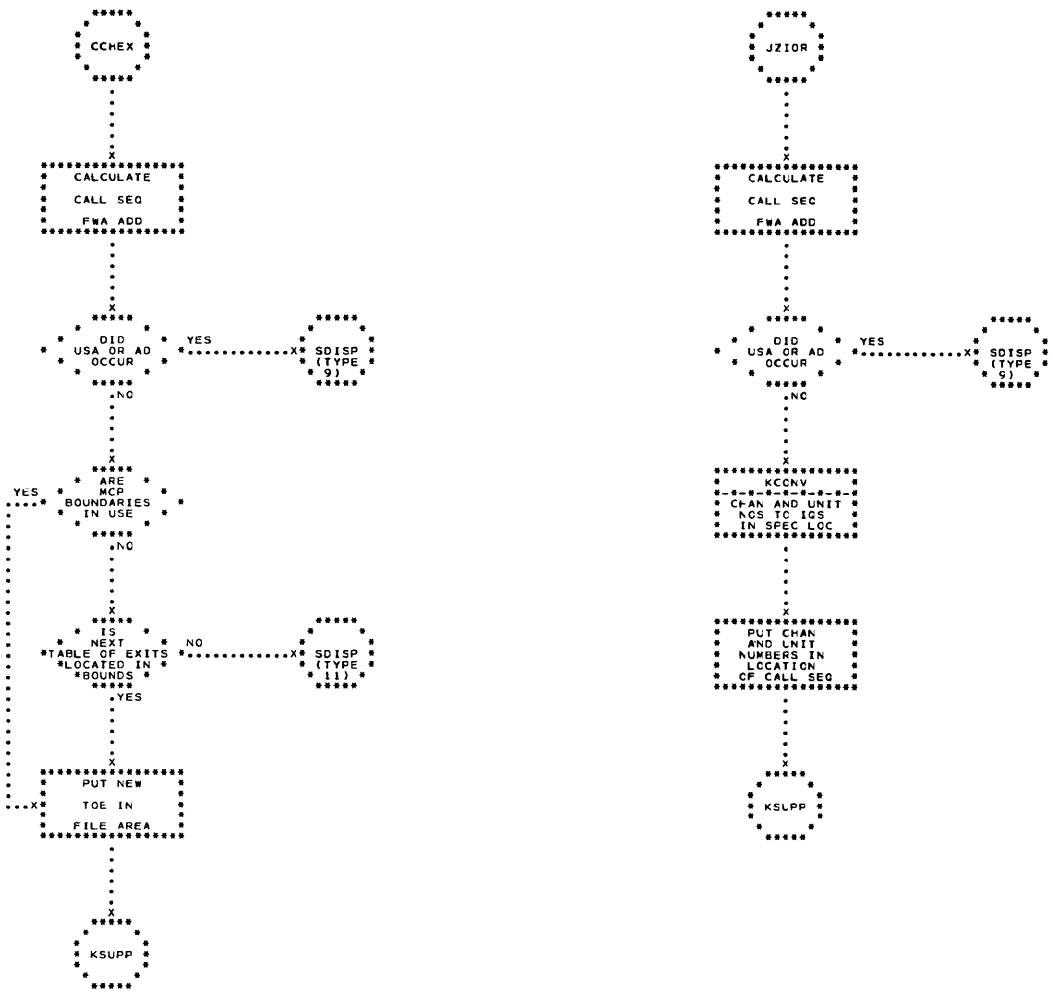


Figure 118. CHEX and IODEF Routines

them in the specified location. If $AD or $USA come on while computing the storage address, control is given to dispatcher error control (SDISP) with error code 9 in $14. Return is made to KSUPP.

Change I-O Table of Exits

This routine is entered at CCHEX (Figure 118) when $CHEX is requested. The location of the new TOE is loaded into the CF of $0. If $AD or $USA occurred, dispatcher error control (SDISP) is entered with error code 9 in $14. If PP boundaries are in effect (the boundary control bit, 3.57=1), the new TOE address is compared to see whether it is within the bounds, and if not, dispatcher error control is entered with type 11 error. The new TOE is stored in the file area table, and return is made via KSUPP.

Free Routine

The free routine is entered at ZFREE (Figure 119). If another unit on the channel is operating, an enabled loop to the linkage is set up via KELOOP and KSUPP. If the channel specified by the $FREE is operating and the I-O set up bit is on in the UST, the same enabled loop is entered. If the unit is operating on a non-set up request, the set up I-O bit is turned on and the return address (SRETAD) set in the UAT as ZFREE+1. Normal return is made via KSUPP.

If the channel is not operating, or when the setup I-O operation is complete, the routine at ZCOMRT is used to perform error checking and to control status evaluation, and the unit not-assigned bit (SUNASG) is turned on in the unit status table. The subroutine SBDTA is entered to compute the location for this reel in the current reel on drive table, and the SBCDP subroutine is used to determine the final tape disposition. If MCP is the owner, the appropriate MCP name is inserted into the unload message, if not, and the tape is a reusable scratch tape, exit is made via the rewind routine (ZREWST). Otherwise, the appropriate disposition is set up (SBDMT), the corresponding slots in the current reel on drive table are cleared and exit is made through the unload routine (KUNLD2).

INTERRUPT MODE CONTROL ROUTINES

The service-ops $SIO, $RIO and $WAIT are used to control stacking, releasing, selection, and synchronization of I-O interrupts. The routines operate disabled and are entered from the identifer.

SIO and RIO Routines

The SIO routine is entered at KSIO. It identifies the requestor by the status of the level bit (SL) and turns on the SIO bit (SSIO) in the corresponding program status table (SPROGS). It adjusts the address in STIC, and returns via the service op return routine (KSUPP).

The RIO routine is entered at KRIO (Figure 120). It turns off the SIO bit (SSIO) in the requestors program status table (SPROGS), and if the requestor is auto-stacked or has no interrupts stacked, returns via the service-op return routine (KSUPP). Otherwise, low registers are saved in SLRPP or SLRMCP according to the level of the requestor, and control is given to the unstack routine (KUNSTC) to release the first stacked interrupt.

Wait Routine

The wait routine is entered at KWAIT when the pseudo-op $WAIT is requested. It exits to one of the following:

KINTY -- The desired interrupt has been found stacked, and is being released.

KSUPP -- 1. With STIC set to the actuated address: The interrupt has not occurred, and an enabled loop to the actuated address is in effect.
2. With STIC set to the return address: Either the referenced unit is not operating and has no interrupt stacked, or the interrupt occurred and was released during the enabled loop, and the unit was reactuated by the fixup prior to issuing $RET.

In order to avoid waiting twice for a unit reactuated by the I-O fixup, the wait routine issues the following controls:

SINTAD -- The interrupted address is set to zero when entering the enabled loop ($WAIT given in mainstream). It is set non-zero by the receptor when the interrupt occurs.

SEWAIL --This bit in the file area table is turned on when entering the enabled loop. It is turned off when $WAIT is requested by mainstream with SINTAD non-zero because if it was on, the interrupt already occurred in the enabled loop and the $WAIT is considered a NOP, since it was the return point after the fixup.

When entered, the wait routine (Figure 120) adjusts the return address in STIC and sets up $9 for the proper program status table (SPROGS). It performs the logic described above to discard a redundant entry, and determines whether or not the unit specified is currently operating. If it is not, a search is performed for a stacked interrupt for the unit. If a

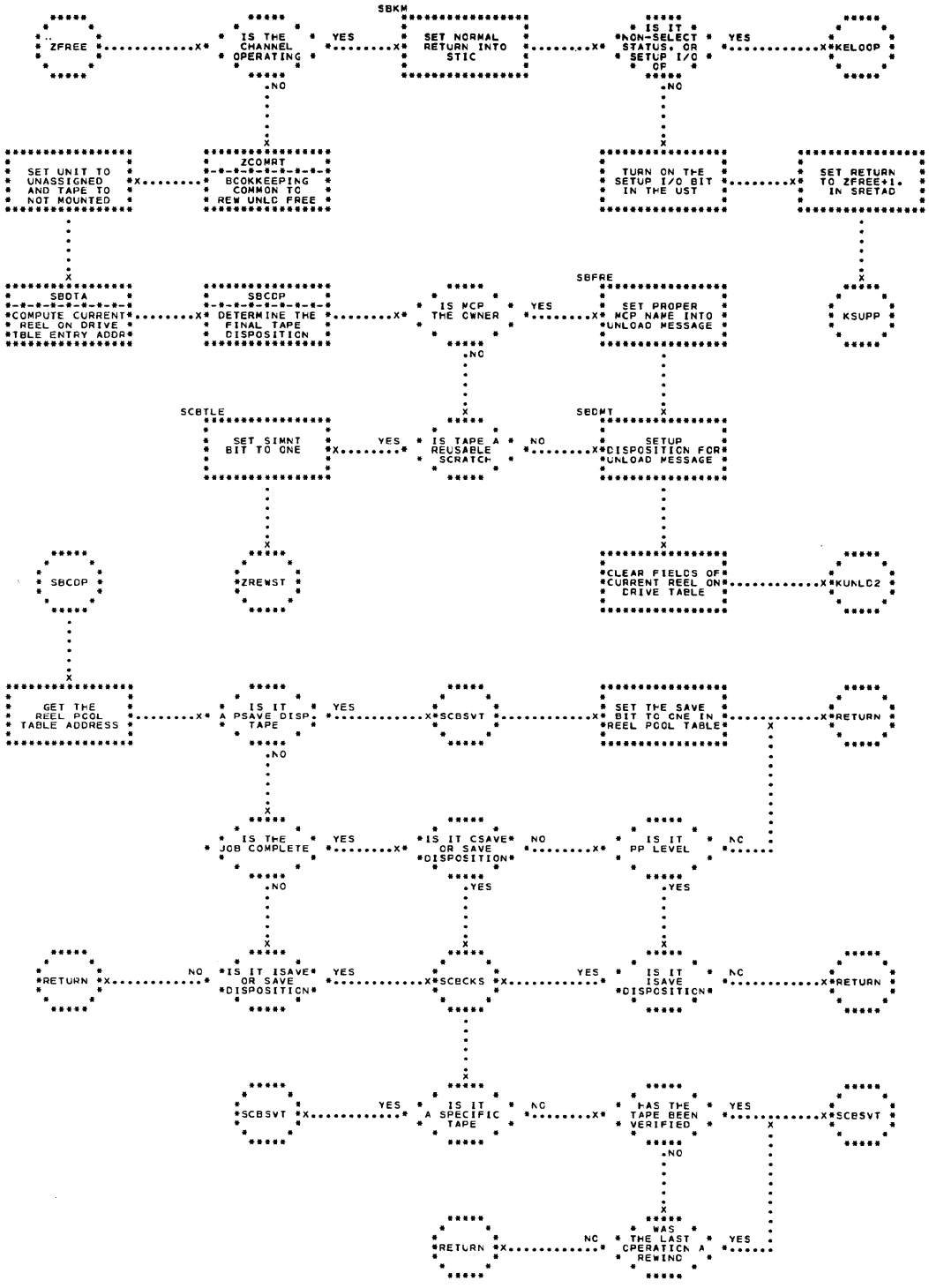


Figure 119. Free Routine

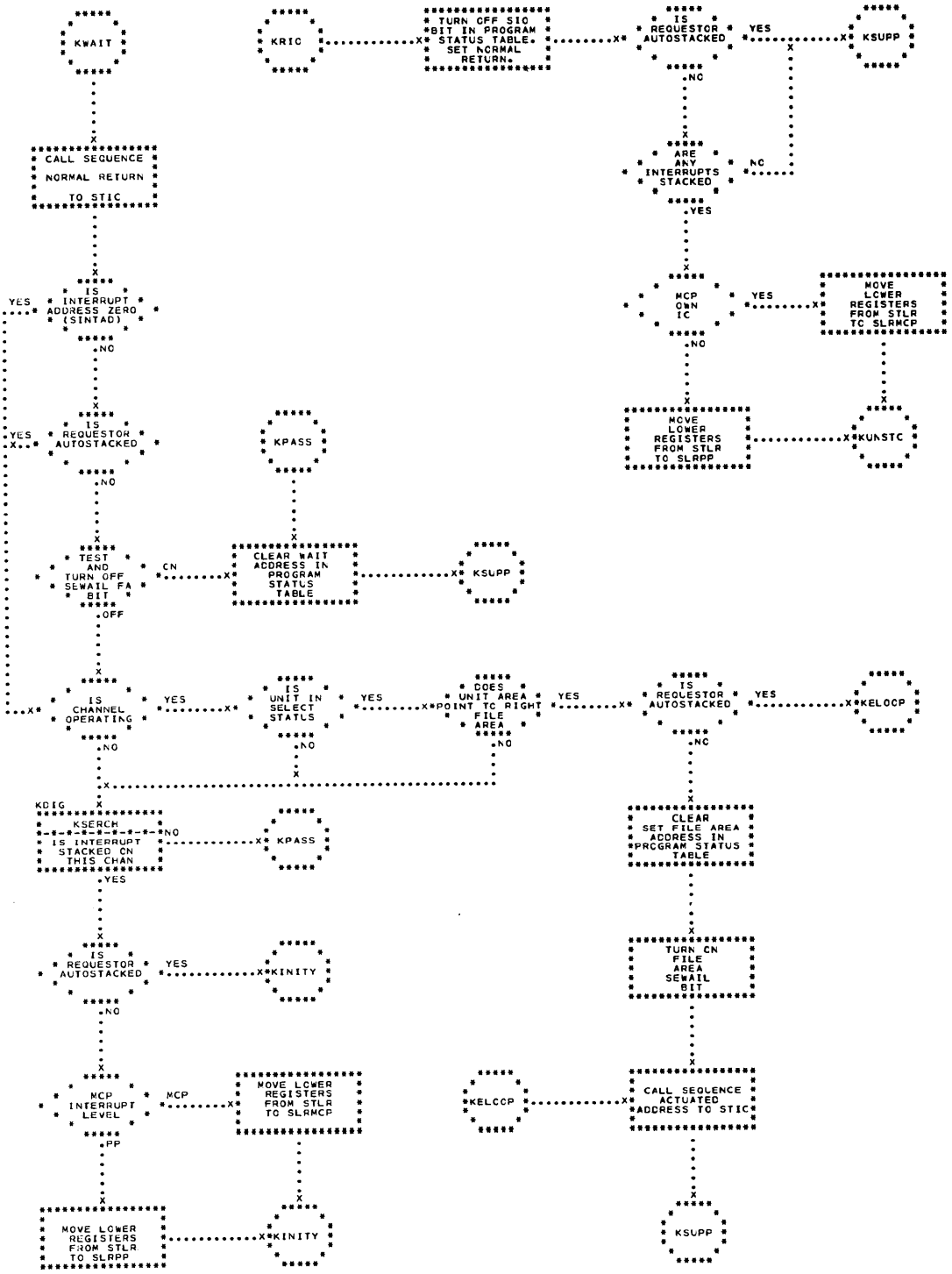


Figure 120. RIO and Wait Routines

stacked interrupt is found, it is released, after moving low registers if necessary. Otherwise, the address of the file area being waited is cleared in the program status table, and control is returned to the normal return via the service-op return routine (KSUPP).

If the specified unit is operating, return is made to the actuating address via the service-op return routine (KSUPP). However, if $WAIT was issued in mainstream, SINTAD is cleared in the file area table, the SEWAIL bit is turned on in the file area table, and the address of the file area table is placed in the program status table before returning.

PROCESSOR COMMUNICATION REGION

When more than one processor is required for a compile run, there must be good communication among the processors. MCP provides a communication region for this purpose, and service pseudo-ops to do the fetching and storing required.

Processing Chains

A processing chain is a sequence of programs which compiles an object program. The type card which is included with the input contains a field identifying a chain whose components will be called in by MCP in the necessary sequence. Links in a processing chain have the essential characteristics of a problem program and, in addition, they have access to a common region wherein they can communicate with one another. I-O devices are assigned to the entire chain so that successive links in the chain may use the same devices.

The LIM card and any IOD cards associated with chain are called in by MCP with the first link in the chain. The limits become the upper and lower boundaries for the entire chain. The IOD's are assigned to the chain as a whole and are not disassigned until the entire chain has been completed. Successive links in the chain can thus use the same physical I-O devices whenever they refer to the same IODNAME's. However, it is the responsibility of each link to give to MCP the locations of its I-O Tables of Exits. This is done through the pseudo-op, $CHEX, Change I-O Table of Exits.

Input

The first link in a processor chain may request its input from the system input source. When FORTRAN, Autocoder, or STRAP II is the first processor in the chain, input must be requested from system input. The first card transmitted will be the card in the source program which follows the type of Problem card. Successive cards may be transmitted through system input until there are no more cards that belong to this job. When system input recognizes this, end return is given to the processor. If data follows the symbolic source program, it is necessary that the processing program recognize the last card of the source program and terminate its requests to system input. Later links in the processor chain may obtain their input from the disk as it is left by a prior link in the chain. This information is obtained through use of disk I-O pseudo-ops.

Intermediate Disk Storage

The disk, except for permanent working storage and PROSA, is available to processors for intermediate storage and for output to the next link in the chain. In the communication region, MCP will enter the number of arcs available in this temporary working storage. At the end of the chain, MCP expects to find in the communication region the number of arcs occupied by the compiled program. On this basis, MCP can then assign disk areas for the use of the compiled program.

References to temporary working storage by processors is made through the disk I-O pseudo-ops whose IODNAME is $TWS. This is a permanently assigned I-O definition, equivalent to:

```
TWS     IOD,     TRACK
```

where the number of arcs which is at the beginning of a track is given in the communication region.

Output

Print or punch output may be transmitted to system output through the use of the appropriate pseudo-ops. Systems output must be used to produce the listing and punched output.

Communication Region

A common communication region is located in core storage. It is used by MCP to communicate with the processors, by the processors to communicate with each other, and by the processors to communicate with MCP. The communication region consists of 25 full words and may be fetched by processors through the following linkage:

```
B, $MCP
, $FECRG
, FWA. (I)
(return)
```

where

FWA. (I) is the first word (18 bit) address of a 25-word block of storage in the processor to which the entire region is transmitted.

The current contents of the communication region may be replaced by means of the following linkage:

```
B, $MCP
, $STRG
, FWA.(I)
(return)
```

where

FWA.(I) is the first word (18 bit) address of a 25-word block of storage in the processor from which the entire communication region is transmitted.

Communication Region Format

The format of the communication region is as follows:

0.0 - 0.17	Number of links in the chain.
0.18 - 0.24	Not used.
0.25	Last link flag.
0.26 - 0.31	Not used.
0.32 - 0.49	Position number of current link.
0.50 - 0.63	Not used.
1.0 - 1.17	Number of arcs in temporary working storage (TWS).
1.18 - 1.20	Type of run: 100 Compile 010 Compile and Go 001 Go
1.21 -	List option: 0 No list 1 List
1.22 -	Punch option: 0 No punch 1 Punch
1.23 - 1.63	Type card option bits.
2.0 - 2.47	Name of the 1st link.
2.48 - 2.63	Not used.
3.0 - 3.47	Name of the 2nd link.
3.48 - 3.63	Not used.
4.0 - 4.47	Name of the 3rd link.
4.48 - 4.63	Not used.
5.0 - 5.47	Name of the 4th link.
5.48 - 5.63	Not used.
6.0 - 6.47	Name of the 5th link.
6.48 - 6.63	Not used.
7.0 - 7.17	Number of arcs occupied by compiled program.
7.18 - 7.24	Not used.
7.25 -	Reject flag.
7.26 - 24.63	Defined by chain of processors.

For the BSS processor chain, words 8 through 13 are defined as follows:

PP LOWER LIMIT	8.0
PP UPPER LIMIT	8.28
ORIGIN OF BLANK COMMON	9.0
ADDRESS OF RELOCATION TABLES	9.32
BRANCH ADDRESS	10.0
MAIN ENTRY POINT	10.32
ADDRESS OF NODE TABLE	11.0
MASK FOR LINK TABLE	12.0-13.63

Communication Region Usage

The first seven words of the communication region are initially provided by MCP, and are used by MCP when control is passed from one link to another. If a link is to modify any of these seven words, any change must be consistent with subsequent usage by MCP (See JC4). The use of the remainder of the communication region is determined by the requirements of the chain.

The number of links in the chain of processors and the position number of the current link are not greater than 5. The position number of the current link is initially set to 1 and is updated by MCP at the end of each link in the chain. The last link flag is set to 1 by MCP when the current link is the last link.

The number of arcs (at the beginning of a track) in TWS is the number available on the disk to the chain. This is not changed by MCP during processing. The links in the chain must communicate to other links about how this total area is utilized. This may be done in the remainder of the communication region.

The type of run was given to MCP on the type of problem card. It is the responsibility of the last link in the chain to leave the binary output on the disk if the run was Compile and Go and the reject flag is not on.

The list and punch options were given to MCP on the type of problem card. These refer to output for the problem program via system output and are independent of the binary output used for Compile and Go.

The names of the links in (A6) BCD are used by MCP to call in the links in the given sequence and are put in the communication region for reference by the links in the chain. The number of the current link acts as a pointer to the name of the current link.

The number of arcs occupied by the compiled problem program must be entered by the last link in the chain if the type of problem is Compile and Go and the reject flag is not on. MCP assumes that the compiled problem program is left in card form and may be loaded from the first arc up to and including the number of arcs occupied in TWS. This field is tested by MCP only after the last link is completed. It may be used for inter-communication between links as long as the last link enters the number of arcs occupied by the compiled problem program.

The reject flag is set to 1 (on) if the Go phase of a COMPILGO job is not to be executed.

The Service Op Routines for Fetch and Store

The communication region is stored in MCP at KSILO. The service routines for $FECRG and $STRG both receive control disabled from the identifier. They start at KFECRG and KSTRG, respectively, and use a common subroutine, KCAB, to set up the transmission and perform error checking.

Control is given to dispatcher error control with error code 9 in $14 if the region specified exceeds PP bounds, or if USA or AD occurs while calculating the region address. Otherwise, control is returned to the parent routine which performs the transmission and returns via the service op return routine (KSUPP).

TIME SERVICE OP ROUTINES

There are two service ops dealing with time: $TIME is a request for the current time clock reading in IQS format, and $SIT is a request to set the interval timer to a specified value.

The $TIME Service Op Routine

This routine will read the time clock, adjust the reading by a predetermined calibration constant, convert the result into an hours-minutes-seconds form, and transmit the result to a requested location. The calling sequence is:

```
B, $MCP
, $TIME
, A. (I)
(Return)
```

where

A. (I) is the full word (18 bit) address to which the edited reading is transmitted. The edited reading will consist of 8 IQS-coded characters and will occupy A. (I) in the format:

.0 - .15	hours
.16 - .23	colon
.24 - .39	minutes
.40 - .47	colon
.48 - .63	seconds

The clock reading is adjusted, and successive divisions by 60 are performed to yield seconds, minutes, and hours, the first two of which will appear in the remainder register. The algorithm is as follows:

$$T + K = Q_S$$

$$\frac{Q_S}{60} = Q_{m(AC)} + S_{(RM)}$$

$$\frac{Q_m}{60} = H_{(AC)} + M_{(RM)}$$

where:

T = Current time clock reading (26 bits from 28-53)
K = Time clock calibration constant
S = Adjusted seconds
M = Adjusted minutes
H = Adjusted hours
AC = Accumulator
RM = Remainder register

Upon entry at ZTIME, the routine adjusts the return address and calculates the storage address from the calling sequence. If an AD or USA occur, or if a PP storage address is outside PP bounds, dispatcher error control is entered with error code 9 in $14. Otherwise, the clock reading is converted according to the algorithm, and return is made via the service op return routine (KSUPP).

The $SIT Service Op Routine

This routine is entered disabled from the identifier at JSITX. It adjusts the return address in STIC, calculates the address of the timer setting, and stores the setting in $IT. If AD or USA occurs while calculating the address of the timer setting, control is given to dispatcher error control with error code 9 in $14. Otherwise, return is made via the service op return routine (KSUPP).

THE COMMENTATOR

The commentator service routine is entered from the identifier when the $COMM pseudo-op is requested. It may also be entered by a disabled MCP user, and from the receptor when a commentator output operation is completed. The function of the commentator is to type messages of up to 10 words in IQS code on the console typewriter.

The commentator write operation is performed as setup I-O. The console may be used in a PP IOD, and is used in an MCP IOD by the debugging package. Thus, the commentator may receive a request and find the console busy with an unrelated operation. The commentator must then be prepared to take over the console, stack another user's interrupt, attend to its own function, and arrange the eventual unstacking of the original user's interrupt.

Main Flow

The commentator program uses a series of subroutines. It is entered at JCOMM (Figure 121) when the pseudo-op $COMM is requested, and at SCOMM when being used by a disabled MCP program. In the latter case, the linkage is:

```
SIC, SCOMIC
BD, SCOMM
(parameters as for $COMM)
```

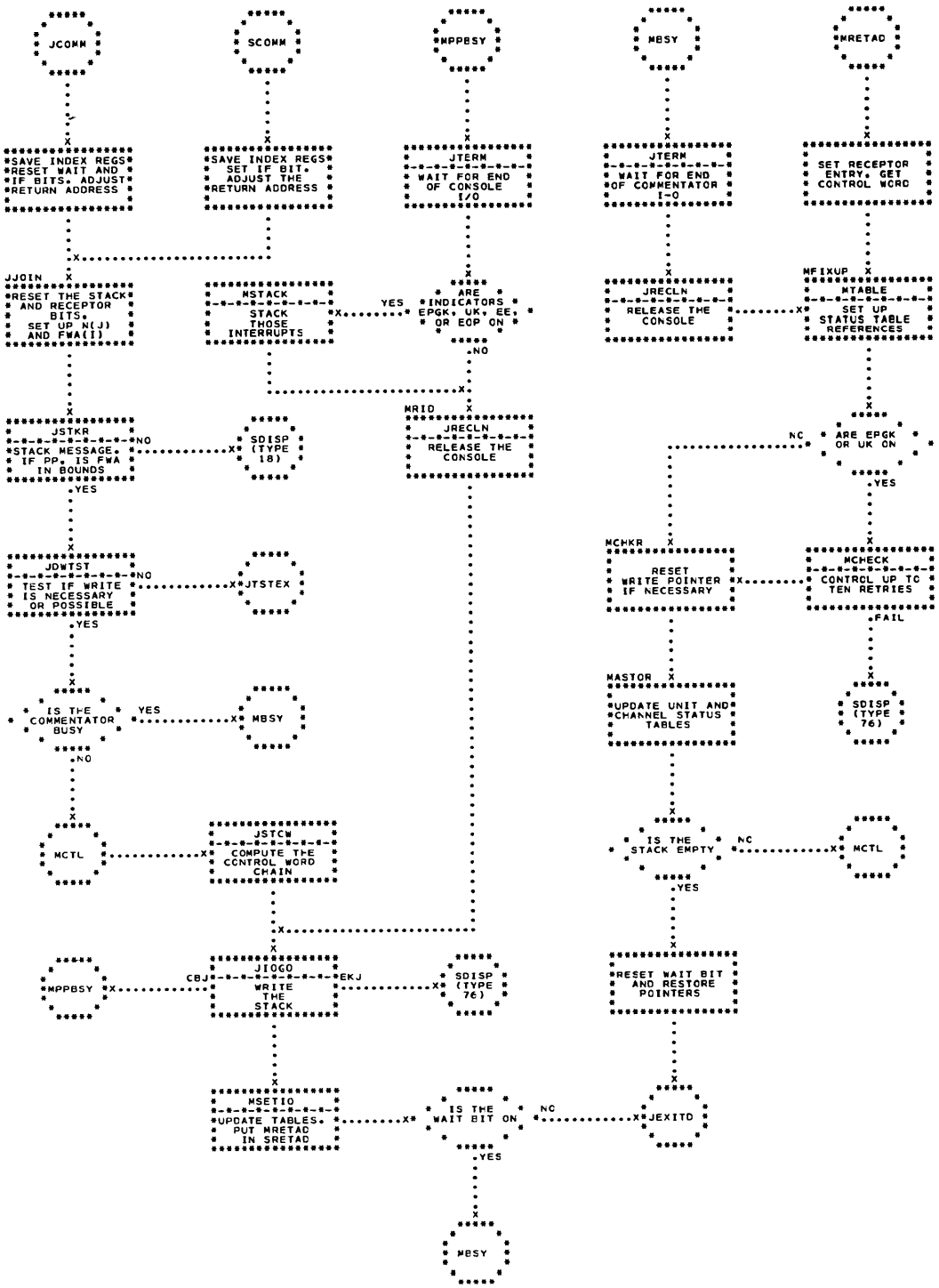


Figure 121. Commentator - Chart 1

The program uses two bits to identify the entry. It also uses a bit to inhibit return until all output is complete, a bit to indicate that it was necessary to stack console interrupts and a bit to indicate the commentator is writing. These bits are:

JIF	0 --	entry from enabled user (through IF analyzer).
	1 --	entry from disabled user.
JRECEP	0 --	entry by requestor.
	1 --	entry from receptor on completion of I-O operation.
JWAIT	0 --	do not wait to perform output if another commentator output is in progress and there is room to stack this request.
	1 --	do not return until the stack buffer is empty.
JSTACK	0 --	no outside interrupts were stacked.
	1 --	console interrupts have been stacked.
JIOBSY	0 --	commentator idle.
	1 --	commentator busy.

When a request is received from a disabled user (SCOMM), the JIF bit is set to identify the entry and to force completion of the output before returning. When a request is received from an enabled user (JCOMM), the JIF and JWAIT bits are reset to identify the entry and to permit stacking the request and returning if a commentator output is in progress.

The flow for the two entries comes together at JJOIN, where the receptor and stack bits are reset, and the parameters evaluated. The stacking subroutine, JSTKR is used to stack the message, entering dispatcher error control if a PP request had a FWA out of bounds. The JDWTST subroutine is entered to decide whether or not to write. It will return to the write return if:

1. the commentator has no output in progress; or,
2. the wait bit is on; or,
3. the message stack is full, that is, a message of maximum length, 10 words, could not be added.

Otherwise, the subroutine will return to the Do Not Write return, which sends control to the exit sequence, JTSTEX.

If the commentator has an output operation in progress, MBSY is entered. Control will ultimately return to MCTL when the operation is complete. Otherwise, at MCTL, JSTCW is used to set up a two-word control word chain, and JIOGO is used to perform the write operation and check EKJ and UNRJ. If CBJ is encountered, the console was being used by another program, and JIOGO returns accordingly. Control is given to MPPBSY to wait for the operation to finish and stack the resulting interrupt if necessary. The write operation is then performed. The subroutine MSETIO is used to update the I-O control tables and place MRETAD in SRETAD in the unit area table. If the wait bit is on, MBSY is entered to wait for the write to finish and, if the stack is not empty, initiate another write. If the wait bit is off, control is given to JEXITD to return.

At JEXITD (Figure 122), the program returns to KSUPP if entry was from the receptor. Otherwise, controls are set, if necessary, to insure unstacking of interrupts before returning to PP, and at JTSTEX return is made to KSUPP or to MCP according to the JIF bit.

When a commentator I-O interrupt occurs, the program is entered at MRETAD from the receptor. The receptor bit is set, the control word copied, and the MBSY flow joined at MFIXUP.

Commentator Subroutines

There are ten commentator subroutines. All are entered with linkage of the general form:

```
LVI, XR, (return address or first parameter)
B, (Subroutine)
```

Some have multiple returns, and some require parameters either in the linkage or index registers. Most are short and straightforward.

The Stack Subroutine, JSTKR

JSTKR picks up the request parameters, branching immediately to JTSTEX if a zero word count is specified. It prefaces all messages with a carriage return and all MCP messages with a red $. It checks a PP FWA against PP bounds, entering dispatcher error control on an error. It checks the word count to be sure it is 10 or less, and moves the words to the stack. It adds a terminating carriage return, stores the new stack pointer (JSTKP), and returns.

The Write Test Subroutine, JDWTST

The function of this subroutine is to decide whether or not to do a write now. It has two returns, one for Write and one for Do Not Write. If the commentator has no write operation in progress, or if the wait bit is on, it will return to the write return. Otherwise, it will select the return on the basis of whether or not the stack has room for another message. If not, it turns on the wait bit and takes the write return.

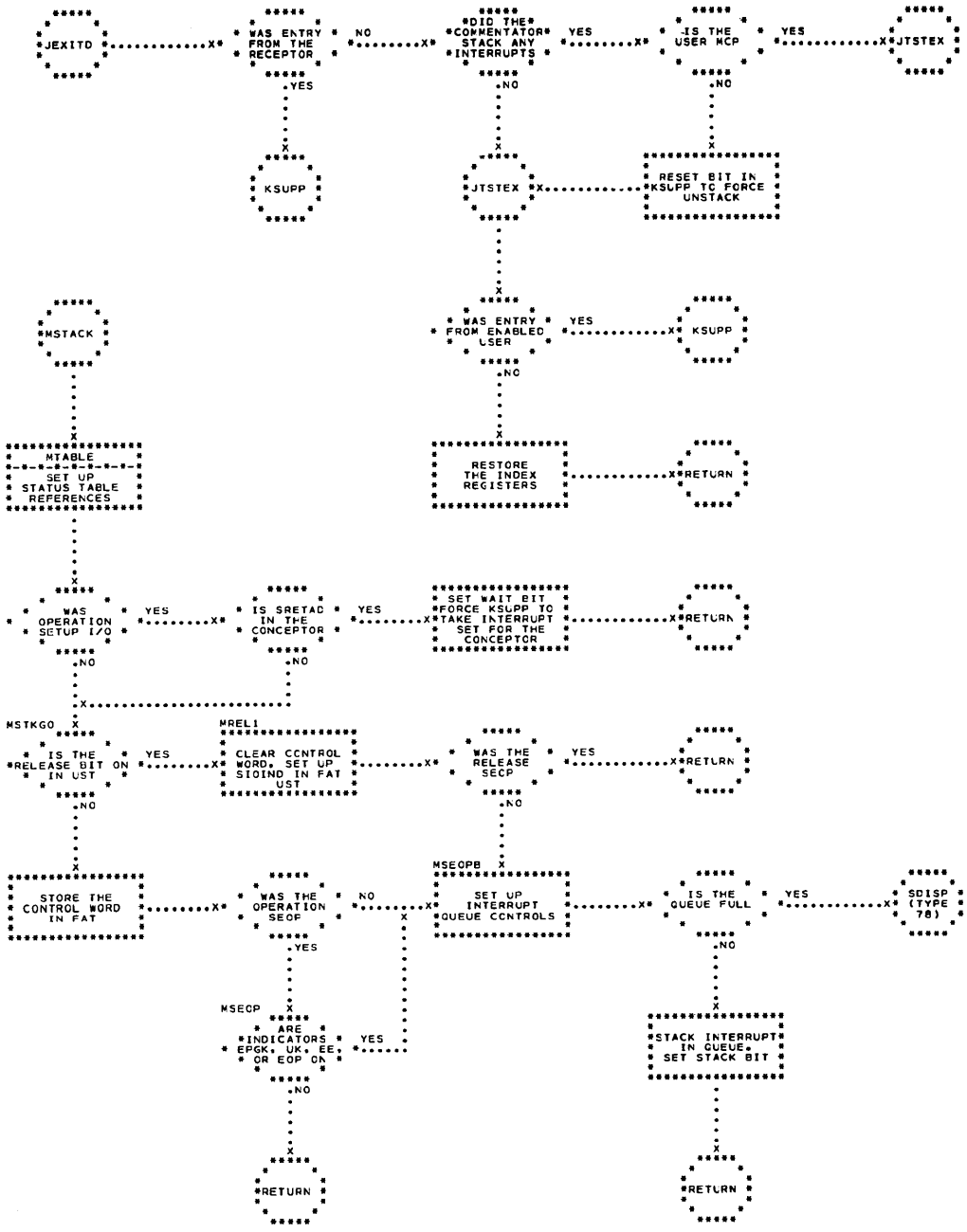


Figure 122. Commentator - Chart 2

The Control Word Setup Subroutine, JSTCW

JSTCW stores blanks in the last partial word, if any, in the stack. It sets up a control word to type the stack from the word specified by the write pointer to the word before the one specified by the stack pointer. This control word will be the second of a chain of two. The first will control writing the display lights as most recently used by the debugging package. If the resulting word count is zero, the chain bit in the first control word is left off. The routine checks the stack pointer, turning on the wait bit if there is no longer room for one more message, and returns.

The I-O Routine, JIOGO

The function of JIOGO is to perform a specified I-O instruction, retry if EKJ or UNRJ occurs, and select one of two returns according to whether or not CBJ occurs. The routine will retry a EKJ ten times before giving a type 76 error, and will retry a UNRJ until the unit accepts the operation. If CBJ occurs, the routine will try the operation ten times before using the CBJ return.

The I-O Bookkeeping Routine, MSETIO

This routine is entered after a successful write operation. Its function is to update the I-O control tables. It should be noted that there are two unit status tables for the console. Although the console is on a single unit channel, it is treated as two logical units, one available for PP and one for MCP. The two UST words are consecutive, the first MCP's and the second PP's.

The routine sets up the addresses of the control tables, updates as shown below, then returns.

1. Sets the select bit (SSEL) in the MCP UST, and resets the select bit in the PP UST (SSEL + 1.0).
2. Sets the channel operating (SCHOP) bit in the CST, and the set up (SSETUP) and write (SWR) bits in the UST.
3. Stores MRETAD in SRETAD, and the I-O instruction in SIOINS in the UAT.
4. Clears the unit field (SUNIT) in the CST.

The Wait for I-O Routine, JTERM

The JTERM routine is used to monitor the completion of the current console I-O operation. The routine is a copy control word loop which is broken under the following conditions:

1. Ten consecutive EKJ's on a CCW.
2. EPGK, UK, EE, or EOP occur.
3. The channel operating bit (SCHOP) is off and the commentator is not the source of the operation.
4. The SEOP bit in the CW if off and the operation was a SEOP release by the commentator.

In case 1, the routine gives a type 76 error message. For the rest, the routine (at JFKCS) tests the CS bit in the CW, and returns if it is off. Otherwise, it uses the KCSIN subroutine in the receptor to stack the CS, and returns.

The Release Routine, JRECLN

This routine is used to release the console when the commentator has retained control until the operation is complete. The routine performs a SEOP release, uses JTERM to wait for its completion, and returns. If EKJ occurs, the routine retries the release ten times before giving a type 76 error.

The Interrupt Stack Routine, MSTACK

This routine is used to stack interrupts for another console use prior to releasing the console for commentator use. The routine (MSTACK, Figure 122) uses the MTABLE routine to set up control table addresses, and then determines whether or not the interrupts must be stacked, and if so how.

If the I-O operation was not set up, and if the release and SEOP bits are on, no interrupt will be stacked. The routine returns after updating the tables. If the operation was a conceptor set up operation, KSUPP and KCONCP are set to cause the interrupt to be taken when KSUPP is entered. If the operation was SEOP and not set up and the release bit is off, the routine returns after updating the tables. Otherwise (MSEOPB), the interrupt is stacked in the interrupt queue, the stack bit turned on, and the routine returns.

The Table Address Routine, MTABLE

The MTABLE routine sets up the addresses of the CST, UST, UAT, and FAT according to the current owner of the unit, and returns.

The EPGK and UK Fixups, MCHECK

This routine is entered when EPGK or UK is detected after a commentator I-O operation. It will count up to ten consecutive retries, and then hang up. A successful operation will cause the count to be restarted.

MCP includes four major packages to provide debugging facilities for the problem program:

$EDUMP - the error dump routine
$DUMP - the dump routine
$HOLD - the instruction counter hold routine
SDDT - the console debugging package

These programs include both dynamic and console debugging features. (Note: it is recommended that the reader review those portions of the MCP Reference Manual concerned with checkout facilities and console usage.)

The $EDUMP is used to override the automatically provided dump parameters used when ABEOJ occurs. On ABEOJ, Job Control 4 uses the dump parameters most recently specified by the $EDUMP pseudo-op, or if $EDUMP was not used, JC4 specifies a dump in octal-hex format from the PP lower bound to the PP upper bound.

A dump is provided any time $DUMP is requested by the PP. The $DUMP routine is used by the debugging package to perform dumps which are requested from the console in checkout mode. The console debugging package (SDDT) may be used by itself or in conjunction with the $HOLD pseudo-op so that the debugging is performed at a particular point in the PP. The console debugging package may be used to examine the flow of the program, change the flow of the program, dump as desired, and examine and change words in PP memory.

THE DUMP ROUTINES

Of the two dump pseudo-ops, $DUMP and $EDUMP, only $DUMP is a bonafide dump request. $EDUMP only specifies dump parameters to be used in the event ABEOJ occurs. Both routines are available to PP which operates in the non-SIO enabled mode.

The $EDUMP Program

The $EDUMP program (RAEDUMP, Figure 123) receives control with the location of a string of dump parameters in its tentacle table. The string may contain as many as six distinct dump specifications. The string is examined; if no errors are detected, is placed in the JC4 buffer, YEDLL, for subsequent use if an ABEOJ occurs. If the location of the string or any of the addresses in the string violate PP bounds, or if any format specification is invalid, $RET is given and the request effectively ignored.

The $DUMP Program

The $DUMP routine prints via $SPR a requested number of memory locations in a format specified by the user. The calling sequence B,$MCP;, $DUMP;, FWA(I) specifies the location of a string of 3 half word parameters of the form: , A.(I);, B.(I);, Format(I), where A.(I) is the first location to be dumped, B.(I) is the last location and the format can be either octal-hex with or without mnemonics, index word or floating point. The format also requests panel or no panel and indicates whether or not this is the last request.

The routine is entered at AWDUMP (Figure 124) with the tentacle table, PDUMPJ, set up. The lower registers are picked up from SLRBU and saved in the dump buffer. A check is made on the parameter, FWA(I) to make sure it is within bounds if the user is a PP. (The user may also be JC4 and the console debugging package.) In case of error a panel dump is given and control is returned to the PP.

If the user is the PP a set of dump parameters is checked for validity. If A(I)$<$SPPLB the programmer's lower boundary is substituted for A(I), or if B(I)$>$ SMARK the programmer's upper boundary is substituted for B(I). If A(I) $>$ B(I), or if the format is not one of the existing formats, only a panel dump is given. Only one substitution per dump request for A(I) or B(I) will be made. If a second such error occurs in the PP's parameters only a panel dump will be performed.

A check is made on the suppress panel bit and the last request bit and corresponding indicators are set up within the dump program. The panel is dumped, if requested, one line at a time by setting up a print image in buffer AMNTOP and writing it out via $SPR. After the panel has been dumped control is given to one of three format subroutines, AOCTLH, AFLPNT, or ANDXWD, which follow the basic pattern stated below. Mnemonics are printed if requested after each line of octal-hex has been printed.

The print buffer, APCOL1, is initialized with a single space carriage control character and a line of blanks. Next a check is made for equal consecutive memory locations. If more than four are found, the message xxxxxx.x TO xxxxxx.x ALL CONTAIN _____ is printed. If fewer than four are found the location counter is set up in the print image followed by four words of dump. Each line is printed as it is formed via $SPR. When the line containing the last location requested has been printed, control is either returned to the user (at APUMP1) if this was the last request,

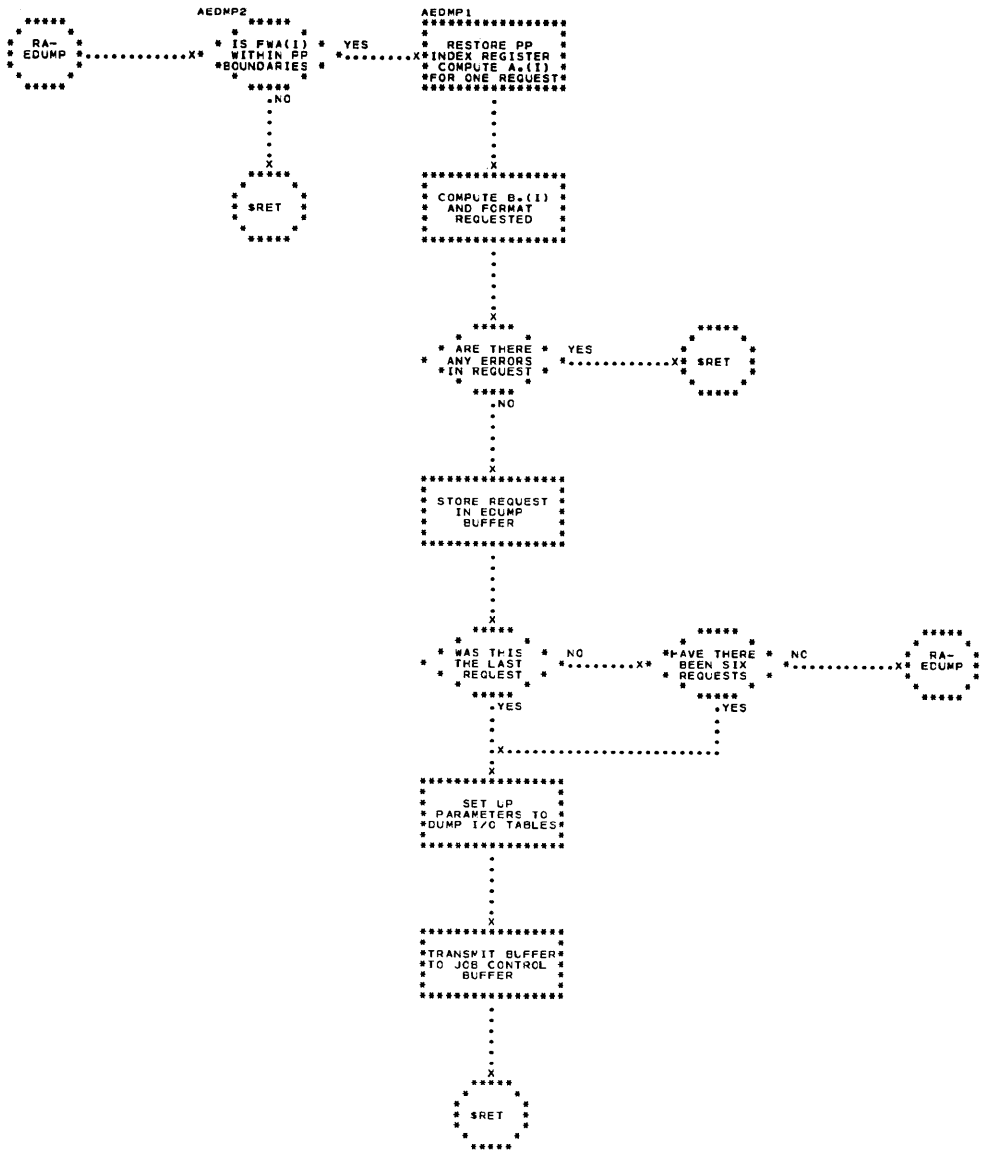


Figure 123. Error Dump Routine

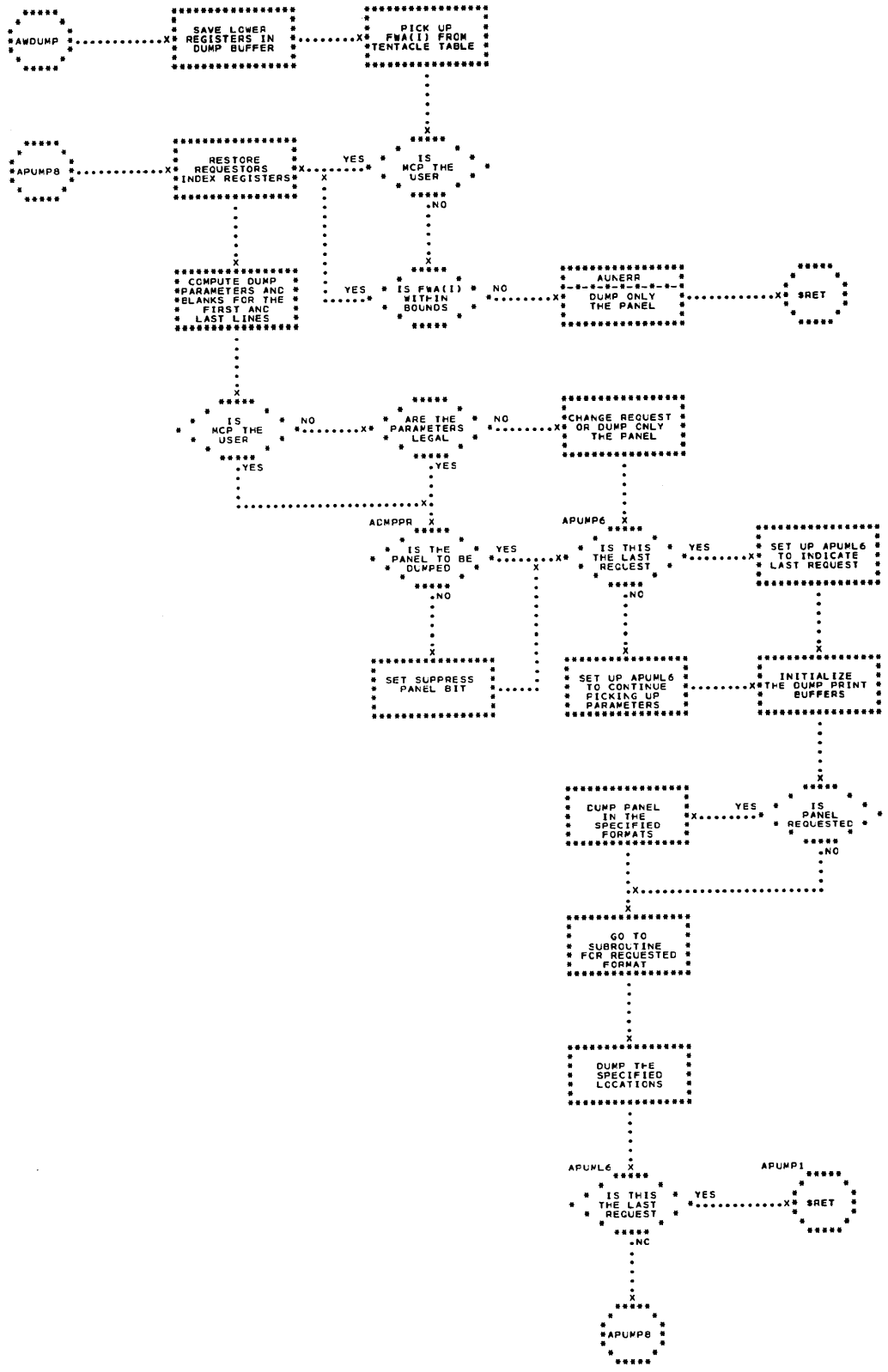


Figure 124. $DUMP Routine

or to the beginning of the dump program (APUML7) to pick up another set of parameters. Four words are printed in each line of the dump except the first and last lines, which include only the locations specified in the dump limits; the uncalled for words are blanked out.

Symbol Definitions

ASPLBI: 0 -- no substitution for A. (I) has been made for this dump request.
1 -- SPPLB has been substituted for A. (I).

ASPUBI: 0 -- no substitution for B. (I) has been made for this dump request.
1 -- SMARK has been substituted for B. (I).

APANEL.19: 0 -- changes instruction at APANEL to a B, AFORMT-.32 which causes printing of the panel to be suppressed.
1 -- changes instruction at APANEL to a NOP which causes the panel to be printed.

ANORMP.19: 0 -- changes the instruction at ANORMP to a B, APUMP7 which causes only the panel to be dumped.
1 -- changes the instruction at ANORMP to a NOP which permits dumping of memory in the specified format.

ANDX4.19: 0 -- the instruction becomes a B, APNL1 and indicates that ANDXWD is being used to dump the panel index registers.
1 -- the instruction is a NOP and means that ANDXWD is being used for index word format.

ADHMN.19: 1 -- a check is made to see if mnemonics have been requested.
0 -- the statement xxxxxx.x TO xxxxxx.x ALL CONTAIN _____ is to be printed in octal-hex and mnemonics are to be suppressed.

ACTLBT: 0 -- not an octal-hex request.
1 -- used by ACMP1 to print out the above-mentioned statement in octal-hex format.

AFLPTI: 0 -- not a floating point request.
1 -- used by ACMP1 to print out the above-mentioned statement in floating point format.

AHLFWD: 0 -- used in the mnemonics routine and assumes that the half-word currently being processed is a half word instruction.
1 -- the half-word currently being processed is the second half of a full word instruction.

APNLI: 0 -- indicates normal floating point dump.
1 -- used to tell floating point conversion routine that $MR is being dumped and that control should be returned to APNL4.

CONSOLE DEBUGGING

Console debugging consists of stopping the PP, either with a $HOLD request or by putting MCP in the check-out mode, performing some sequence of operations at the console, and restarting the PP. The $HOLD operation is used to stop the PP whether or not PP requests it, the distinction being that PP can control the location of the stop by requesting $HOLD at the desired location.

The $HOLD Routine

The $HOLD routine (PHOLD, Figure 125) consists simply of a routine which primes itself and issues $RET. On initial entry (PP user, not primed), it sets the hold bit, PHOLDB. On any primed entry, if the bit is on it primes itself and issues $RET. When the bit is off on a primed entry, it issues $RET without priming itself, thus allowing the return routine to consider returning to PP. The PHOLDB bit is only turned off by the console debugging program when a restart request is received. The initial entry to the routine may be a primed entry, the priming being done by the console debugging routine, which also turned the hold bit on.

The Console Debugging Package

The console debugging major package is made up of its tentacle table, a main line to handle testing and returning, a set of subroutines, and an I-O table of exits. The I-O unit is PCDEF2, defined as console with TOE PDBEX. The initial entry to the debugging package is at the CS entry in the TOE (PDBO, Figure 126) when the ENTER, END sequence is recognized by the conceptor. This fixup reads the console entry into the input buffer and primes the debugging package. This is the only way in which control is given to the debugging package.

The program (PDBBEG, Figure 126) operates in the same memory as JC4, and on initial entry must fetch itself from PROSA using the major package fetch routine in JC4. The main line of the program consists of examining the keys and switches for requests (PDB1), ultimately (PE1) writing to the console displays if

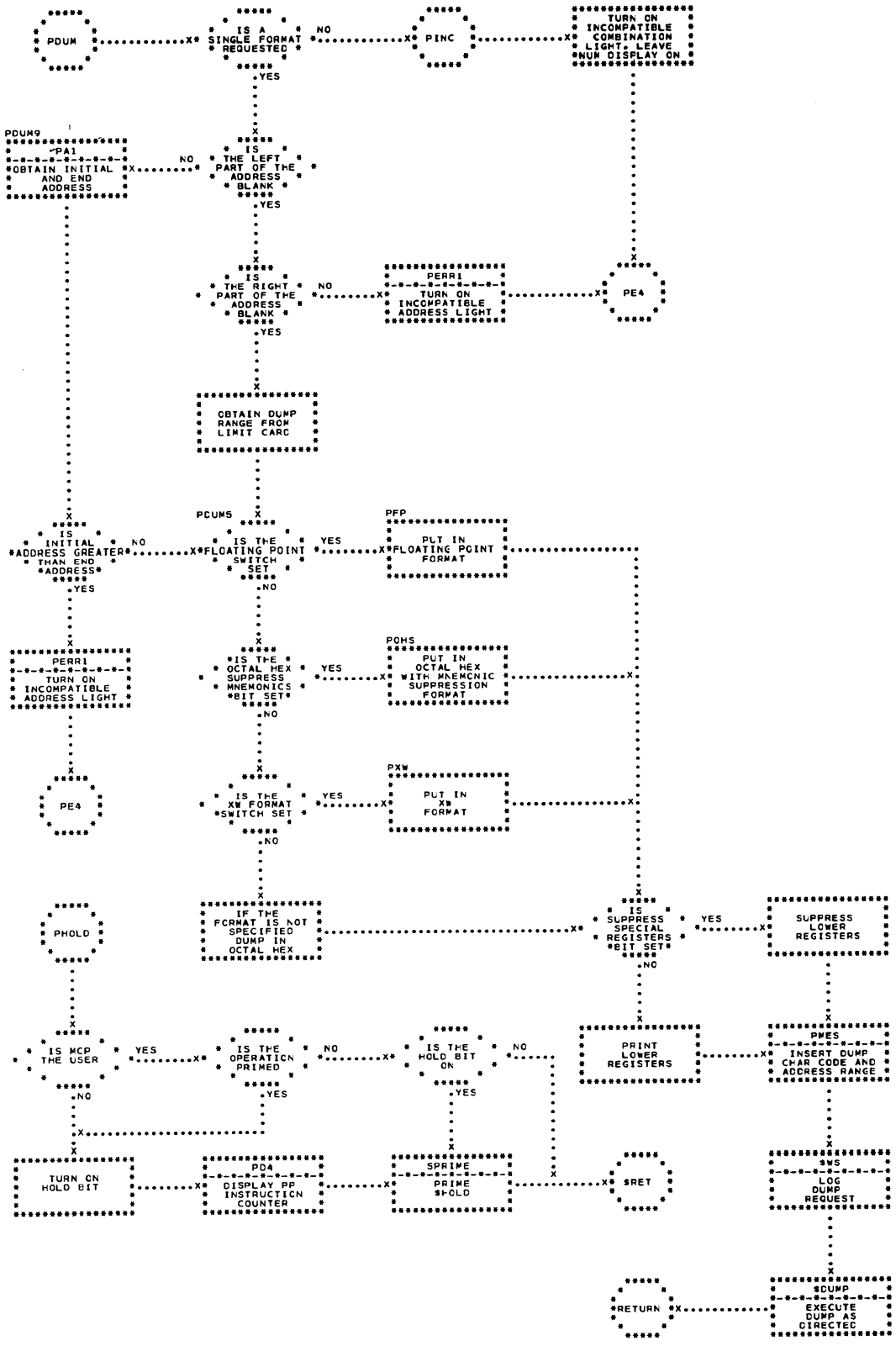


Figure 125. Console Debugging Package - Chart 1

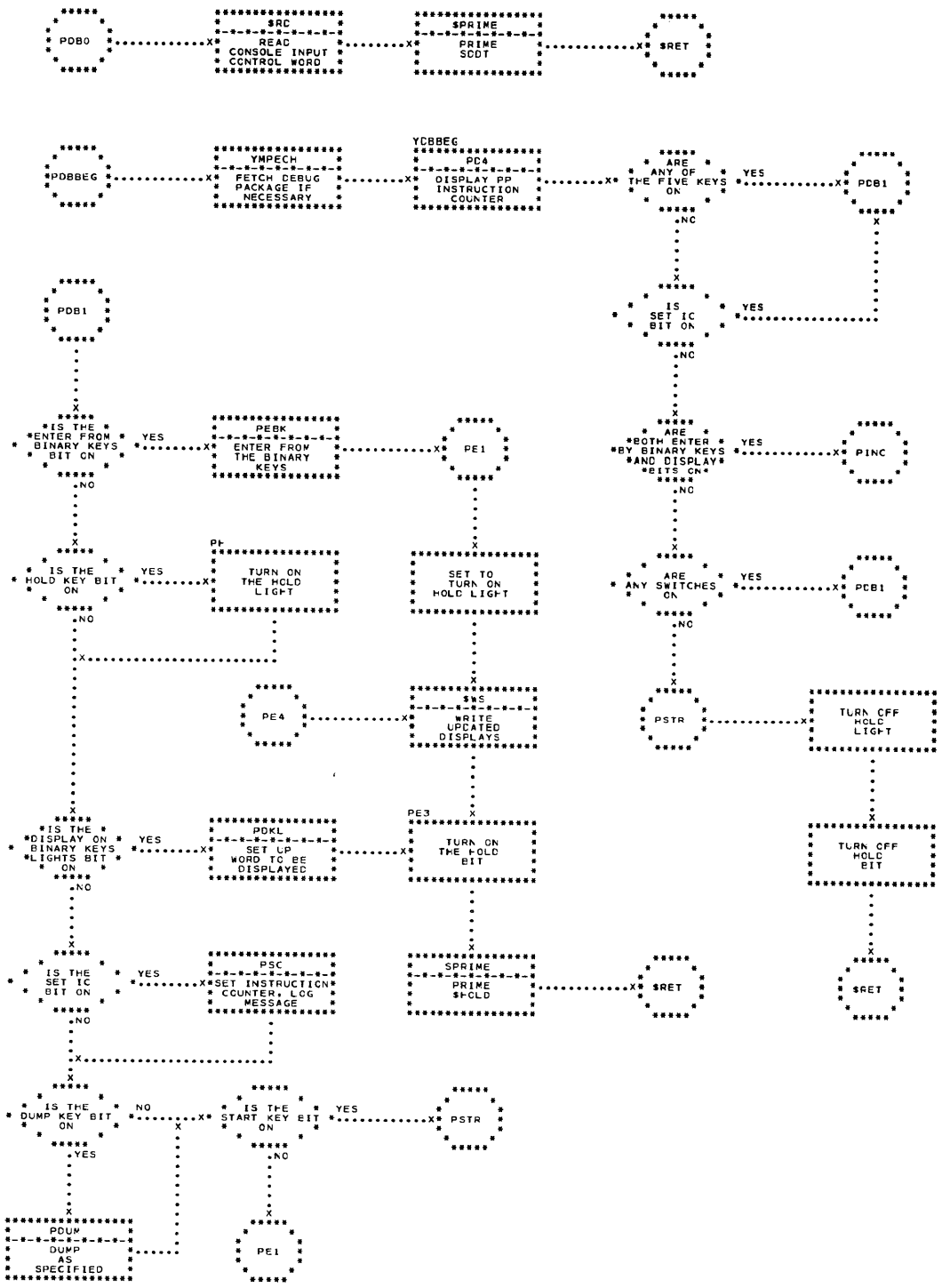


Figure 126. Console Debugging Package - Chart 2

necessary. If a restart has not been requested, the program, at PE3, turns the hold bit on (PHOLDB) and primes $HOLD to insure that control will not return to PP. When a restart is requested, the program turns off the HOLD light, resets the hold bit, and issues $RET. This will break the $HOLD loop and allow control to return to PP. The various subroutines that accomplish the debugging requests are shown in Table 8.

There are also four pseudo-ops that may be used as debugging aids:

$ABEX -- the PP TOE ABEOJ routine
$FELR -- the fetch lower registers routine
$STLR -- the store lower registers routine
$FIXUP-- the maskable interrupt fixup routine

The PP TOE at ABEOJ Routine

Upon entry at YABEX, the PP ABEX table is checked by the boundary checking routine (RSVCODE). If the table is not within PP bounds, an error message is written by priming $SPR and return is given via KSUPP. If bounds are not violated, the count at SCBAX is set to two indicating that an ABEX is in effect, the PP ABEOJ return address is saved in YPPAX and return is made via KSUPP.

The Fetch Lower Registers Routine

Entered at TFCHLR, the PP boundaries are checked by the boundary checking routine (RSVCODE). The

Table 8. Subroutines -- Debugging Requests

Name	Function	Return Normal	Return Error *
PA1	Convert an address from four bit decimal to binary, rejecting it if any blanks (1111) are encountered.	Requestor from PA02.	PERR1
PEBK	Store a quantity of bits in a specified location after conversion to binary.	PE1 via PSC.	PERR1
PH	Perform a console write to turn the hold light on.	Requestor via $14.	
PDKL	Obtain the word requested from PP memory and display it on the binary key lights, using PLT to test for a legal request.	PE3 via PSC.	PINC
PSC (PSC3)	Change the PP instruction counter if requested (SICBU) and (PSC3) set up the IC for display. Perform a log entry on the typewriter.	Requestor via $14.	
PDUM	Convert a dump request to the format required by $DUMP, and test its legality; use $DUMP to perform the dump.	Requestor.	PERR1 PINC
PMES	Used by PDUM to log the dump request on the typewriter.	Requestor.	

* PINC and PERR1 are error routines which set up the output buffer according to the error condition and return to PE4.

error control routine (SDISP) is entered with error code 9 if the PP boundaries are violated, otherwise, the PP lower registers, the base address of the I-O tables, and the converted IOD number are stored into the PP table. Return is made via KSUPP.

The Store Lower Registers Routine

The store lower registers routine is entered at TSTLR where the mode is checked for auto-stacked. If not auto-stacked mode, the error control routine is entered (SDISP) with a type 20 error code; otherwise, the boundary checking routine (RSVCODE) checks the PP bounds and, if violated by the STLR table, enters error control (SDISP) with a type 9 error code. The return from STLR is checked for boundary violation. If a boundary violation exists, the routine exits at KGOOF; otherwise, the indicator register is saved and the lower registers from the STLR table are stored into the PP backup registers (SLRPP). The IF Mask register bit (21) is set to one, TS and EXE are reset according to the saved indicator register contents in SLRPP, and return is made via KSUPP.

The Fixup Routine

Enter $FIXUP at FFXP (Figure 127) and save the return address in STIC. Compute the effective parameter addresses, checking for legal boundaries. If not within PP bounds, exit via the error routine (SDISP) with a type 9 error code.

Store the pattern word from B(J) into A(I), if the latter is non-zero. Remove all extraneous ones from the pattern word, and return via $RET if the result is zero, otherwise, save the count of the number of fixups to be handled.

At FCKBT, adjust the control indices and test the pattern bits. For each one bit, save the contents of the relative interrupt table location at A(I), if A(I) is non-zero. Test the corresponding fixup word and if illegal, exit via the error routine (SDISP) with a type 19 error code.

Store the legal fixup word directly into the relative interrupt table location and decrement the count. When the count becomes zero, return via $RET, otherwise continue at FCKBT.

ENTRY POINT NA

Exits To:	Condition	Registers Restore	Remarks
SDISP	error; code in $14	none	error possible: Type 9 - parameters not with PP bounds Type 19 - illegal FIXUP word
$RET	$FIXUP complete	none	1 - zero pattern word 2 - all changes completed

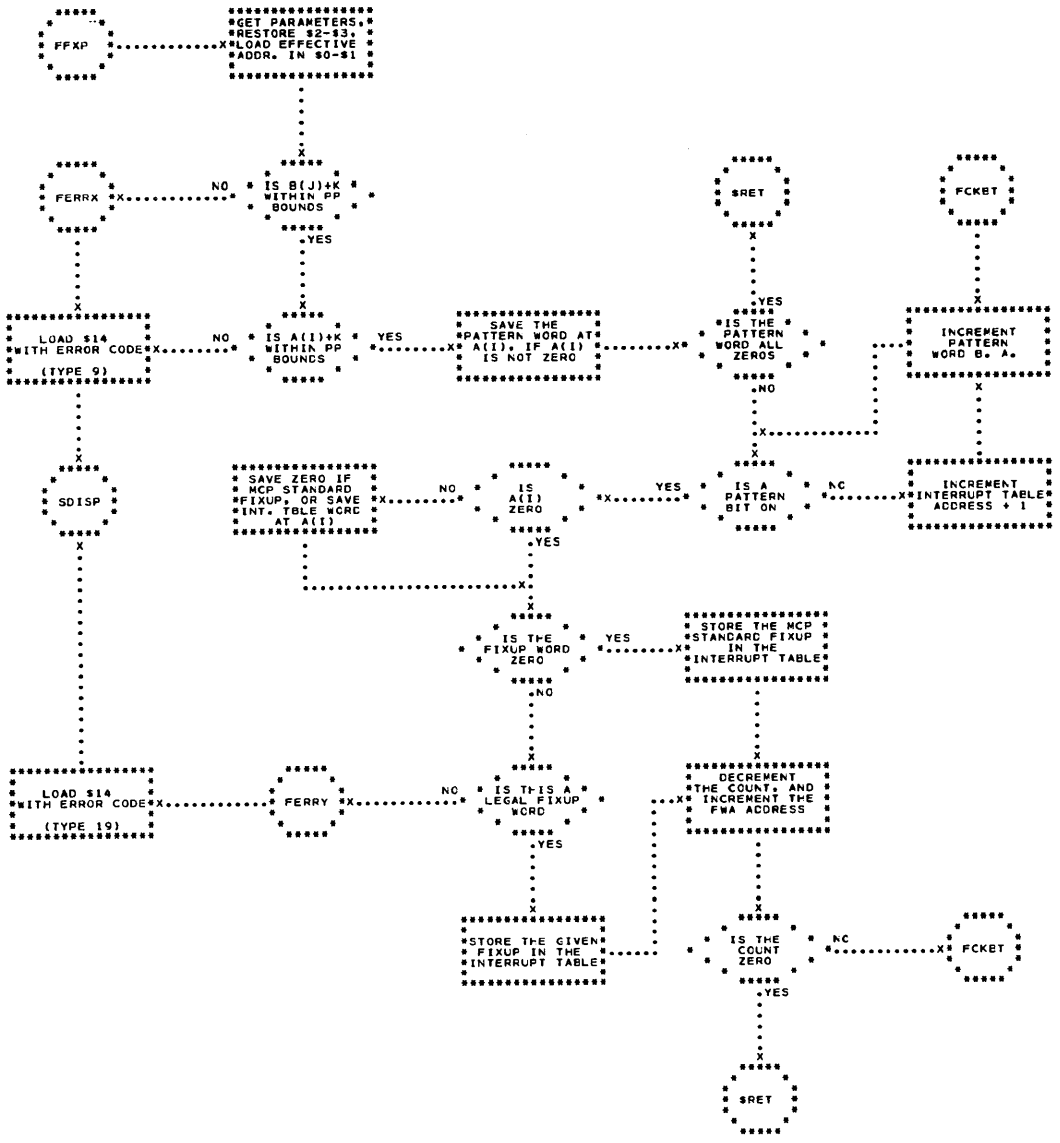


Figure 127. $FIXUP Routine

The master control program (MCP) is supported by update programs and peripheral input/output programs. The update programs provide a method for maintaining symbolic tape files, and for updating the permanent read-only storage area on disk.

The peripheral input/output programs prepare tape labels and input tapes for use by MCP and provide an off-line output medium for MCP.

UPDATE PROGRAMS

The update programs consist of two individual programs:

1401 Symbolic Update Program
Master System Update Program

The 1401 Symbolic Update Program allows the programmer to update symbolic programs on tape by merging files on a master tape with changes recorded on cards to generate a new master symbolic tape.

The Master System Update Program provides a means for updating any system program contained on the permanent read-only storage area (PROSA) maintained on disk. The program prepares a new IPL (initial program load) tape by merging the current system on the disk with changes read from system input or a specially prepared tape. It also includes a means for updating FORTRAN generator libraries.

Symbolic Update Program (1401)

Symbolic Update (Figure 128) is a IBM 1401 program which aids in maintaining symbolic files. Symbolic files are recorded on tape in column binary format, one card per tape record. Each card image has a unique identification in columns 73 through 80. The first two, columns 73 and 74, contain two alphameric characters which identify the file. These characters are initially selected by the programmer; any combination of two characters being acceptable except two blanks. Within each file, cards are numbered sequentially, starting with 000001 in columns 75 through 80. These are called their identification (ID) numbers. Any number of files may be stored on a "master" tape, restricted only by capacity of the tape. The logical end of the master tape is indicated by a double file mark.

By means of the Symbolic Update program the (current) master tape is read from unit 1 and merged with changes read from the card reader. Each file is resequenced as it is updated, and a new master tape is written on unit 2. Files which are not updated or deleted are copied onto the new master tape.

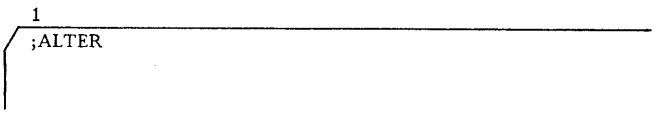
Each specific update procedure is initiated by a control card read from the card reader along with new symbolic cards (change deck). Control cards are identified by a semicolon (an 11-6-8 punch on the IBM 1401) in column 1. The program does not look at the contents of tape records (except for ID numbers) nor any cards other than control cards. Within one update pass, the change deck must be ordered according to the current master tape. That is, all intrafile changes must be in numerically ascending order and changes to various files must be arranged in the same order in which these files are written on the tape.

Control cards and intrafile symbolic correction cards will be printed on-line as they are read. However, the programmer must rely upon the compiler output for a listing and for new sequence numbering of a file on the new master tape.

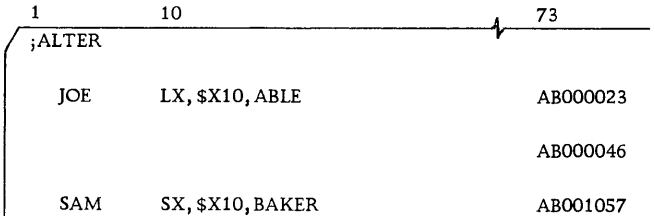
Usage

Intrafile Changes: Intrafile changes are initiated by ALTER control cards, (punched ;ALTER in columns 1 through 6). In addition, the ALTER card may have information punched in columns 10 through 17 and 20 through 27.

Case 1



When columns 10 through 17 are blank, all change cards which follow, up to the next control card, are inserted according to their own ID numbers (columns 73 through 80). If the change card has no punches in columns 1 through 72, then the corresponding record on the current master tape will be deleted. All change cards after an ALTER card must refer to the same file as follows:



Record AB000023 on the current master tape will be replaced by the information on the first card.

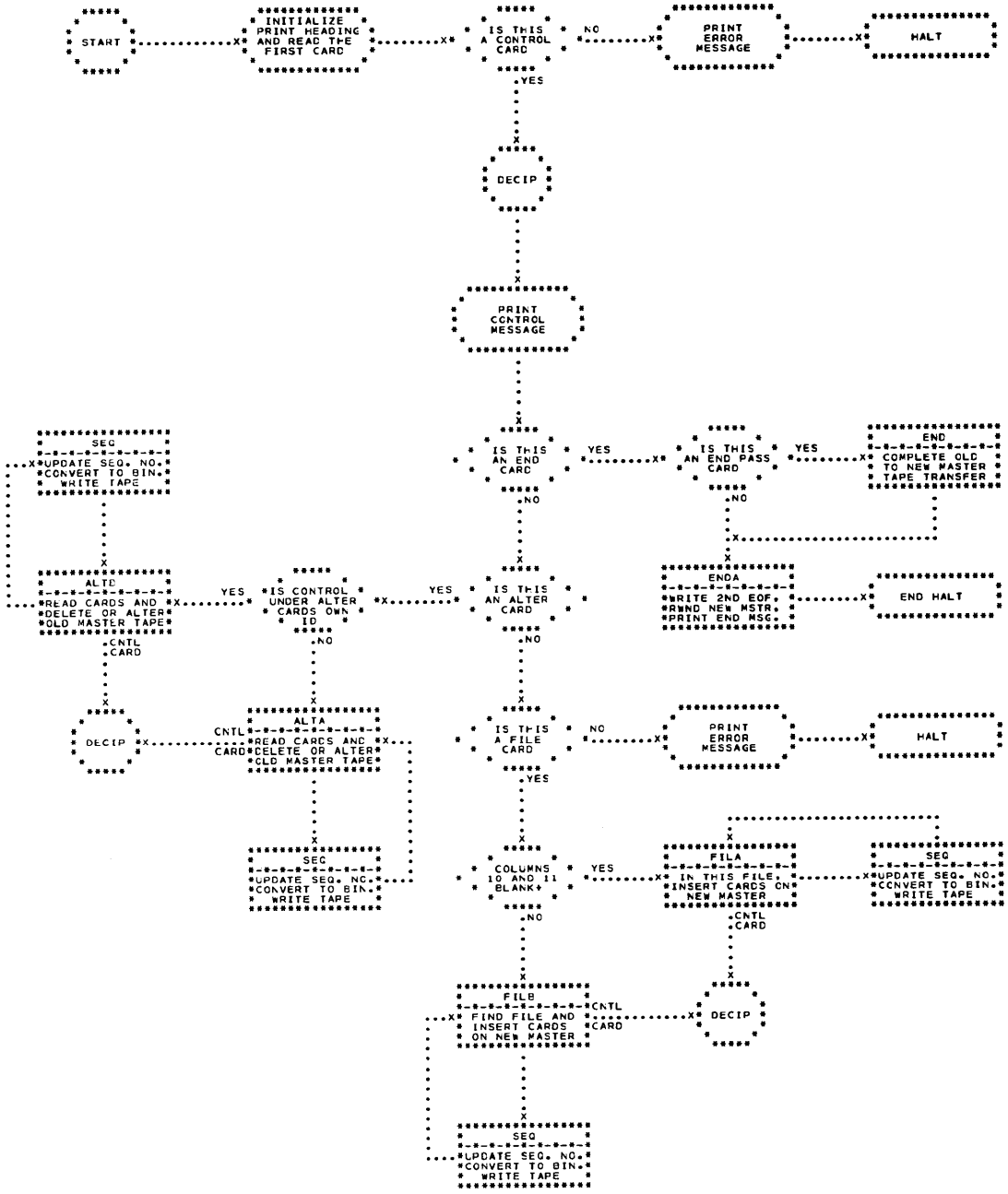


Figure 128. Symbolic Update Program

Record AB000046 will be deleted. Record AB001057 will be replaced with the information on the third card. Notice that because of the deletion card, SAM will not have number 1057 on the new master. The ID numbers always refer to the records to be replaced or deleted on the current master tape.

Case 2

```
1          10
;ALTER     XXyyyyyy
```

When columns 10 through 17 contain an ID number and columns 20 through 27 are blank, all change cards which follow, up to the next control card, are inserted after record XXyyyyyy on the current master tape. These cards are automatically given the proper sequence numbers and the rest of the file is resequenced as the file is copied onto the new master tape. Information in columns 73 through 80 on the change cards is ignored as follows:

```
1          10
;ALTER     DQ000519
   JOE     LX,$X10,ABLE
   SAM     SX,$10,BAKER
```

The two cards shown will be inserted after record 000519 in file DQ.

```
1          10          20
;ALTER     XXyyyyyy    XXzzzzzz
```

When columns 10 through 17 contain an ID number, and columns 20 through 27 contain another ID number within the same file, all records yyyyyy up to and including record zzzzzz on the current master tape will be deleted and any change cards which follow, up to the next control card, will be inserted in their place. For this case a punch option is available. If the letters PUNCH appear in columns 30 through 34 of the ALTER card, the deleted records will be punched on-line.

```
1          10          20          30
;ALTER     EE000100    EE000327    PUNCH
;ALTER     EE000438    EE000445
   JOE     LX,$X10,ABLE
   SAM     SX,$X10,BAKER
```

Records 00100 through 000327 in file EE will be deleted and punched on line. Records 000438 through 000445 in file EE will be deleted and the two cards shown will be inserted in their place.

Note: Any number and type of ALTER controls may be used together on the same file as long as they referenced the records in that file in numerically ascending sequence.

File Changes: File changes are initiated by FILE control cards (punched ;FILE in columns 1 through 5). Columns 73 and 74 of the FILE card contain two alphameric characters to identify the new file being read from the card reader. This file will be automatically sequenced as it is read onto the new master. The new file is defined to be all cards which follow the FILE card up to the next control card. Any information in columns 75 through 80 of these cards will be ignored. FILE cards may also have information punched in columns 10 and 11 and columns 20 and 21.

```
1                                   73
;FILE                               XX
```

When columns 10, 11, 20, and 21 are blank, the cards which follow, up to the next control card, are written on the new master tape and sequenced starting with XX000001. The current master tape is not altered except when the previous control card was an ALTER card, in which case, the program will finish copying the rest of that file on the new master tape before writing the new file as follows:

```
1                                   73
;FILE                               AB
```

The symbolic cards which follow will be written and sequenced as file AB on the new master.

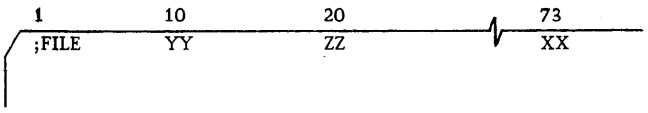
```
1          10                       73
;FILE      YY                       XX
```

When columns 10 and 11 contain a file ID and columns 20 and 21 are blank, the cards which follow, up to the next control card, are inserted and sequenced as file XX after file YY on the master tape. For example:

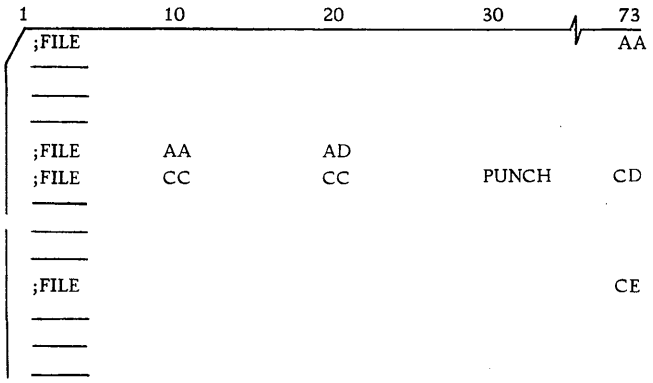
```
1          10                       73
;FILE      BB                       BC
```

The symbolic cards which follow will be inserted as file BC after file BB on the master tape.

Case 3



When columns 10 and 11 contain a file ID and columns 20 and 21 contain another file ID, all files YY up to and including ZZ will be deleted, and any cards which follow, up to the next control card, will be inserted as file XX. For this condition a punch option is available. If the letters PUNCH appear in columns 30 through 34 of the FILE card, the deleted files will be punched on line. For example:

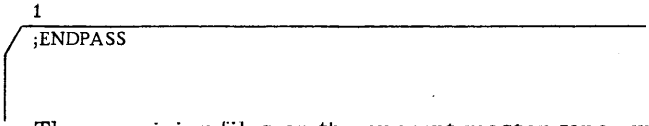


A new file will be written at the beginning of the new master tape as file AA. Files AA through AD on the current master tape will be deleted. File CC will be deleted and punched on line, and the new files CD and CE will be inserted in its place. The second FILE card performs a deletion only, and therefore columns 73 and 74 may be left blank. The last FILE card also contains blanks in columns 10 and 11, because the tape is already properly positioned for insertion of file CE.

Note: The alphameric identification for files on the master tape need not follow a collating sequence. However, such an arrangement is strongly recommended to reduce errors in arranging the change deck.

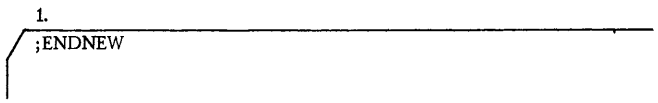
End Cards: The last card of the change deck must be an end control card and may take one of two forms:

Case 1

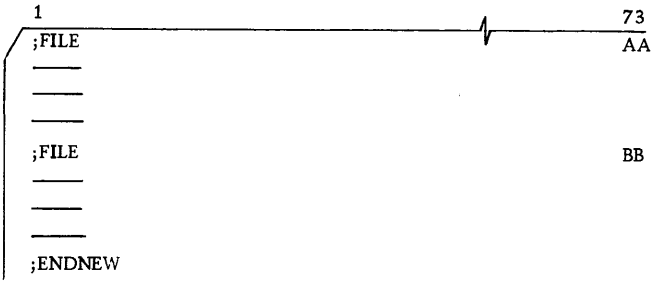


The remaining files on the current master tape, up to the double file mark, are copied onto the new master tape and a double file mark is written. The current master tape is rewound and unloaded, the new master tape is rewound, and the program halts. This is the normal end card, used in all update passes. It may be used as the sole card of a change deck to copy the current master tape onto a new tape.

Case 2



A double file mark is written on the new master tape; it is rewound, and the program halts. Tape unit 1 is not touched. This end card is used when creating a master tape for the first time. For example:



Two files, AA and BB, are read from the card reader, sequenced and copied onto tape unit 2. A double file mark is written after BB.

Operating Procedures for the Symbolic Update Program

1. The deck consists of the self-loading program deck followed by the programmer's change deck of control cards, correction cards, and new files. The standard 1401 loading procedure is used.
2. The current master tape is mounted on tape unit 1 and the new master tape is mounted on tape unit 2.

Error Messages

Three types of error messages may be printed on-line during the running of a Symbolic Update Program: programmer errors, read-write errors, and end-of-reel errors.

Programmer Errors: Upon finding an error in the change deck, the program stops after printing out the error message, and a rerun is required. One of the following error messages may be printed:

1. CORRECTION CARD MISPUNCHED OR OUT OF SEQUENCE.

2. DOUBLE FILE MARK. CHECK SEQUENCE OF CONTROL CARDS.
3. CORRECTION CARDS CROSS FILE BOUNDARY.
4. FILE CARD HAS NO ID IN COLUMNS 73, 74.
5. FIRST CARD IS NOT A CONTROL CARD.
6. ILLEGAL CONTROL CARD.
7. LAST CARD IS NOT AN END CARD.

Since information on the master tape is written in column binary card image, the 1401 program cannot check the collating sequence of file ID's nor the numerical sequence of records within a file. In order to scan the tape the program can only look for an equal condition between an ID punched on a card and read in column binary, and the corresponding ID in column binary on the tape. For example, upon reading the control card

```
1         10
;ALTER    AA000105
```

assuming a preceding ALTER in file AA has left the current master tape positioned at record AA000200, the program will now start reading (and copying) the current master tape looking for AA000105. When the end of file is reached and AA000105 has not been found, the program will print message 1 and halt.

If a preceding FILE update has left the current master tape positioned at the beginning of file BB, the program will start searching for file AA. When the double file mark is reached and file AA has not been found, the program will print message 2 and halt.

Error message 3 is used in case 1 of an ALTER control, where the individual correction cards reference more than one file as follows:

```
1         10                                  73
;ALTER
   JOE    LX,$X10,ABLE                       AD000398
   SAM    SX,$X10,BAKER                      BB000052
```

The other error messages in this category are self-explanatory.

Read-Write Errors: The program provides for automatic retry on tape errors, and restart procedure on card reader errors. Printer and punch errors are not checked. One of the following messages may be printed:

1. TAPE CHECK: RECORD ON OLD MASTER WAS UNREADABLE AFTER TEN TRIES.
2. TAPE CHECK: WRITE ERRORS ON NEW MASTER. BAD SPOT WAS ERASED.
3. READER CHECK: EMPTY HOPPER AND FIX CARDS FOR RESTART. PUSH START.

If the current master tape is unreadable after ten tries, the program prints message 1 and halts. If the new master tape is unwritable after three tries the program prints message 2, erases a gap and continues. On a reader check, the program prints message 3 and halts. The operators should initiate the standard restart procedure. If a reader check is encountered while the 1401 program deck is being loaded, the error will not be discovered until execution time. In this case, message 3 will be the only line printed (normally the name of the program is the first line printed) and the program will halt. The entire deck must then be reloaded.

End of Reel Errors: If the end of tape reflective spot is sensed while writing the new master tape, the following procedure is taken: a double file mark is written, a card is punched which contains the ID of the last file in columns 1 and 2, one of the following messages is printed, and the program halts.

1. LAST FILE ON NEW MASTER IS INCOMPLETE. CARD PUNCHED.
2. LAST FILE ON NEW MASTER IS COMPLETE. CARD PUNCHED.

Normally, the last file written will be incomplete and message 1 will be printed. The incomplete file can be deleted on another pass with a FILE card, and the remainder of the current master tape can be copied onto another tape by deleting the files which were completed. It is possible, however, to sense end of reel while writing an end of file. In this case, message 2 will be printed.

Master Update Program (UPDATE-30)

The STRETCH programming system is maintained on a master tape which is called in at Initial Program Load (IPL). During IPL, the entire system is written in the permanent read-only storage area (PROSA) on the disk, MCP is placed in position in core storage and the master tape is rewound and unloaded.

The master update program, UPDATE-30, merges the current system on PROSA with changes read from system input ($SCR) or a specially prepared change tape, and the updated system is written on a new master tape. If the new master tape is to be used, it must then be called in with IPL.

Master Tape

The master tape contains two binary files in odd parity, high density. The first is a single record containing the IPL bootstrap program. The second consists of 513-word records each containing an image of an arc of PROSA on the disk. Each change affects an entire logical unit of PROSA called a type area.

HED Control Cards

Changes to the program are initiated by HED control cards punched as follows:

```
1          10
B          HED, operation, ttaaaaaa, LOAD
```

The second field (op) may contain one of four operation codes:

DEL	delete
REP	replace
ADD	add
LIB	librarian

The third field (tag name) contains the name of the type area on the disk being operated upon.

The fourth field (LOAD) is optional and only affects REP and ADD operations. If present, the deck which follows the HED card will be loaded, and the new type area will contain the information in memory image. Otherwise, the deck will be read, and the new type area will be in card image.

Operation Codes

The following codes may appear in the second field of a HED control card:

Delete (DEL): The named type area will be deleted from the current system.

Replace (REP): The deck which follows, up to the next HED card, will be read or loaded from the input source and its contents will replace the named type area in the current system.

Add (ADD): The deck which follows, up to the next HED card, will be read or loaded from the input source and its contents will be added to the current system as a new type area with the given name.

Librarian (LIB): The named type area specifies the generator library to be updated according to the control cards and generator decks which follow, up to the next HED card. The updated library and its associated table of contents, which is a separate type area automatically updated by the librarian, will then replace the two corresponding type areas in the current system.

Librarian: The four generator library type areas are: LFORTRAN, LPASSONE, LPASSTWO, and LSMACONE. The names of their associated tables of contents are: TFORTRAN, TPASSONE, TPASSTWO and TSMACONE. The library control cards (punched starting in column 10) are defined as follows:

1. SMADDMAC, A
   Adds the following generator to the library, naming it "A". The sort field is assigned a zero value.
2. SMREPLAC, A
   Replaces generator "A" with the following generator. The sort field remains the same.
3. SMDELETE, A
   Deletes generator "A" and its synonyms from the library.
4. SMRENAME, A, B
   Changes the name of "A" to "B". Also changes any synonyms of "A" to synonyms of "B".
5. SMYNONYM, A, B
   Arbitrarily assigns generator "A" a sort field of "X" for optimal I-O times, where "X" is a 4-digit number between 1-1000.
6. SMARANGE
   Orders the generators by ascending sort fields. SMARANGE may not be used during an update in which SMADDMAC or SMREPLAC have also been used. A zero field has highest priority.

If a HED, LIB card is followed immediately by another HED card, the only action taken by the librarian will be a review of the contents of that library.

The librarian has its own loader which is restricted to standard generator decks, as defined by the SMAC language.

Operating Procedure for the Master Update Program

UPDATE-30 operates as an ordinary problem program and is loaded by MCP from the system input.

The change deck is prepared as follows: The first card must be a date card containing xx/yy in columns 10 through 14, where xx is the current month and yy is the current day. It may also have COMM punched in columns 2 through 5. (See section on Print-Out). Following the date card are the HED cards and new decks for all type area changes.

The change deck may be put on system input directly following the program deck, or it may be put on a separate tape as a single file, (odd parity, high density). For the first option, the third IOD card (for the change tape, named CHT) should be removed from the program deck to eliminate unnecessary tape mounting. For the second option, a REEL card must be added behind the third IOD. The program will automatically determine the input source by trying to read initially from the system input ($SCR).

The program deck includes IOD and REEL cards for the new IPL tape to be written, which is unprotected, unlabeled, and named NWIPL.

If UPDATE-30 is kept as a type area on the disk, it is called in as a compiler by using a job card, type card, and date card ahead of the binary decks of the type areas to be updated. The type card is punched as follows:

```
1         10
B         TYPE, COMPILE, UPDATE30, NWIPL
```

The NWIPL field is optional. If NWIPL is punched in the type card, MCP sets bit KSILO+1.30 in the communication region. This bit is checked by UPDATE-30 at the end of a successful update. If the bit is on, UPDATE-30 changes the update completed message to also state that a new IPL is starting. Then UPDATE-30 gives an EOJ and MCP executes an automatic IPL of the NWIPL tape if the NWIPL option was requested.

Print-Out: UPDATE-30 will list on system output ($SPR) the following information:

1. A title line including original date of current system and current date for the new IPL tape.
2. HED cards as they appear in the change deck. For LOAD options, all T, C, D, and P cards will be printed as they are encountered. For librarian operations, individual changes and a review of the new contents of the library will be printed.
3. The new MCP dictionary, containing the name, arc-word location and size of each type area on the disk.
4. B cards in the IPL program which define the individual system configuration.
5. UPDATE COMPLETED.

The following information will be printed on the console typewriter:

1. A title line including original date of current system and current date for the new IPL tape.
2. xxxxxxxx LIBRARY UPDATE WAS SUCCESSFUL for each librarian request, where xxxxxxxx is the type area name.
3. UPDATE COMPLETED.
4. UPDATE COMPLETED NWIPL STARTING if the NWIPL option was called for on the type card.

If the date card contains COMM in columns 2 through 5, the following additional information will be printed on the typewriter:

1. Individual changes to each generator library.
2. The new MCP dictionary.
3. B cards in the IPL program.

Errors: Input-output errors and any detected errors in the change deck will cause an explanatory error message to be printed on both system output and the console typewriter. UPDATE-30 will then be terminated with an abnormal end-of-job.

Messages for input-output errors are as follows:

1. REPEATED DATA ERRORS ON DISK.
2. REPEATED READ ERRORS ON CHANGE TAPE.
3. END OF REEL ON NEW MASTER TAPE.
4. I-O HARDWARE FAILURE. (UNIT)

Data errors will be retried three times for the disk and ten times for the change tape (if used) before termination. Data errors in writing the new master tape will be retried three times before erasing a long gap and continuing. UK without EOP, and EPGK, are considered I-O hardware failures. The name of the unit (disk, change tape, or new master tape) will be included.

Messages for change deck errors are as follows:

1. NO INPUT CHANGE DECK.
2. FIRST CARD IS NOT DATE CARD.
3. SECOND CARD IS NOT A HED CARD.
4. HED,_____ IS ILLEGAL.
5. NO HED CARD AFTER DELETE_____.
6. NEW TYPE AREA_____HAS ZERO WORD COUNT.
7. LOADER REJECTED__(reason)_____. SEQUENCE NO. OF LAST CORRECT BINARY CARD WAS_____.

   The reason for loader rejection will be one of the following:

   CHECK SUM ERROR
   SEQUENCE ERROR
   ILLEGAL CARD TYPE
   CARD BELOW 1ST ORIGIN
   DECK EXCEEDED BUFFER
   C OR P CARD MISPUNCHED
   NO ORIGIN ON 1ST CARD
   NO ORIGIN CARD
   BSS FUNCTION ILLEGAL

8. TWO HED CARDS HAVE TYPE AREA NAME_____.
9. TRIED TO ADD_____, ALREADY ON PROSA.
10. DELETE OR REPLACE_____, NOT ON PROSA.
11. FIRST FOUR TYPE AREAS ARE__,____, ____,____.
12. TOO MANY TYPE AREAS IN TYPE___. MAX IS 256.
13. TOO MANY TYPES. MAX IS 40.
14. CHDIC BUFFER WAS EXCEEDED.
15. NEW DIC BUFFER WAS EXCEEDED.
16. P CARD PRIOR TO IOCD CARDS.

Messages 4 through 12 will have specific information inserted, in most cases the type area name. Messages 5 and 6 refer to the restrictions that a HED, DEL card must be followed by another HED card, and a HED, ADD card, or HED, REP card must not be immediately followed by another HED card. Messages 11, 12 and 13 refer to MCP restrictions. If the sequence of the first four type areas is violated (refer to the following section on Notes and Restrictions),

message 11 will contain the sequence that was attempted. Message 7 will contain the sequence number, in octal, of the last correct binary card loaded.

There are three miscellaneous error messages:

1. UNABLE TO FETCH CURRENT BOOTSTRAP.
2. UNABLE TO FETCH CURRENT DICTIONARY.
3. NEW DIC IS MISFORMED. PROGRAM OR MACHINE ERROR.

If any of these are printed, either the current system or the program has been contaminated.

The librarian has its own set of librarian error messages for errors in the change deck. They are:

1. ILLEGAL CONDITION, LIBRARY DOES NOT EXIST.
2. ILLEGAL CONDITION, LIBRARY HAS ALREADY BEEN UPDATED.
3. SMARANGE MAY NOT BE REQUESTED WITH SMADDMAC OR SMREPLAC.
4. PARAMETER ERROR. NOT EXECUTABLE.
5. LOGICAL ERROR. NOT EXECUTABLE.
6. ILLEGAL SITUATION -- LIBRARY IS EMPTY.
7. TABLE OVERFLOW. LIMIT IS 200 ENTRIES PER LIBRARY.

Messages 2 and 6 will include the name of the type area, and 4 and 5 will include the library control card which is in error.

An error in loading a generator deck will cause the following message to be printed:

ERROR IN BINARY_____(reason)_____
LAST CORRECT SEQUENCE NUMBER WAS_____.

The reasons for an error in binary deck are also printed as follows:

A CARD IS OUT OF SEQUENCE.
CHECKSUM ERROR.
BITS TO LOAD GREATER THAN 23 HALF WORDS.
ORIGIN IS OUT OF RANGE.
FLOW CARD IS OUT OF RANGE.
(ORIGIN PLUS MODIFIERS) IS OUT OF RANGE.
C CARD OUT OF RANGE.
UNIDENTIFIABLE CARD.
NO TRANSFER CARD.

Notes and Restrictions:

1. If the date card is the only card in the change deck, a new IPL tape will be written containing the current system without alterations.
2. The type areas on PROSA are ordered (in BCD collating sequence) according to their names. MCP requires that the first four type areas be:

   11A11DIC
   11B11IND
   11C11IPL
   11D11MCP

   The first two (dictionary and index) are automatically updated by the program.
3. The IPL bootstrap program, which UPDATE writes as the first file on the new tape, is included as a type area in PROSA name 11E11BSP. It may be updated like any other type area in the system.
4. When the LOAD option is used, the deck must fit within the buffer assigned from JCTBL. The size of the buffer is determined by the size of the machine and is dynamically assigned at the beginning of the UPDATE-30 program (JC001).

Program Description

In essence, UPDATE-30 performs the following tasks:

1. Reads cards from the change deck, one by one.
2. Makes change dictionary entries as required.
3. Merges current dictionary with change dictionary to form new dictionary and new index.
4. Writes new master tape.
5. Computes the check sum and PROSA arc count of the system and stores this information in type area))))))).

Main Flow: Detailed flow of the UPDATE-30 program is as follows:

UPDATE-30 begins at START (Figure 129) where the error dump limits are set. The buffers are dynamically assigned and control words are set up for writing the new master tape. The input source is determined, the date card is read, the IPL bootstrap is fetched from PROSA for the previous date, and the title line is printed.

At HEDCTL (Figure 130), the change deck is read one card at a time from the input source. For every HED card read, an entry (2 words) is made in the change dictionary. The deck which follows either a HED, ADD card or a HED, REP card is read or loaded into the buffer and written on the disk. The word count is computed and stored in the change dictionary entry. The word count is set to zero for a delete operation. When the change deck is exhausted, a dummy last entry with type area name))))))) is stored in the change dictionary.

The READ1 routine (Figure 131) reads a single card from the input source into the card buffer. The card is checked to see if it is a HED card and if so, the HEDBT bit is set. Control is returned to the mainstream of the program according to the value in $15.

The READA routine (Figure 131) reads a deck into the main I-O buffer. When the buffer is filled, the contents of the buffer are written on the disk and the buffer address is initialized. The routine will continue to read the deck until a HED card is read. Return to the mainstream of the program is given from READF.

For a librarian operation two entries are made in the change dictionary, one for the library and one for its associated table of contents. The library control cards are read and each new generator deck is loaded

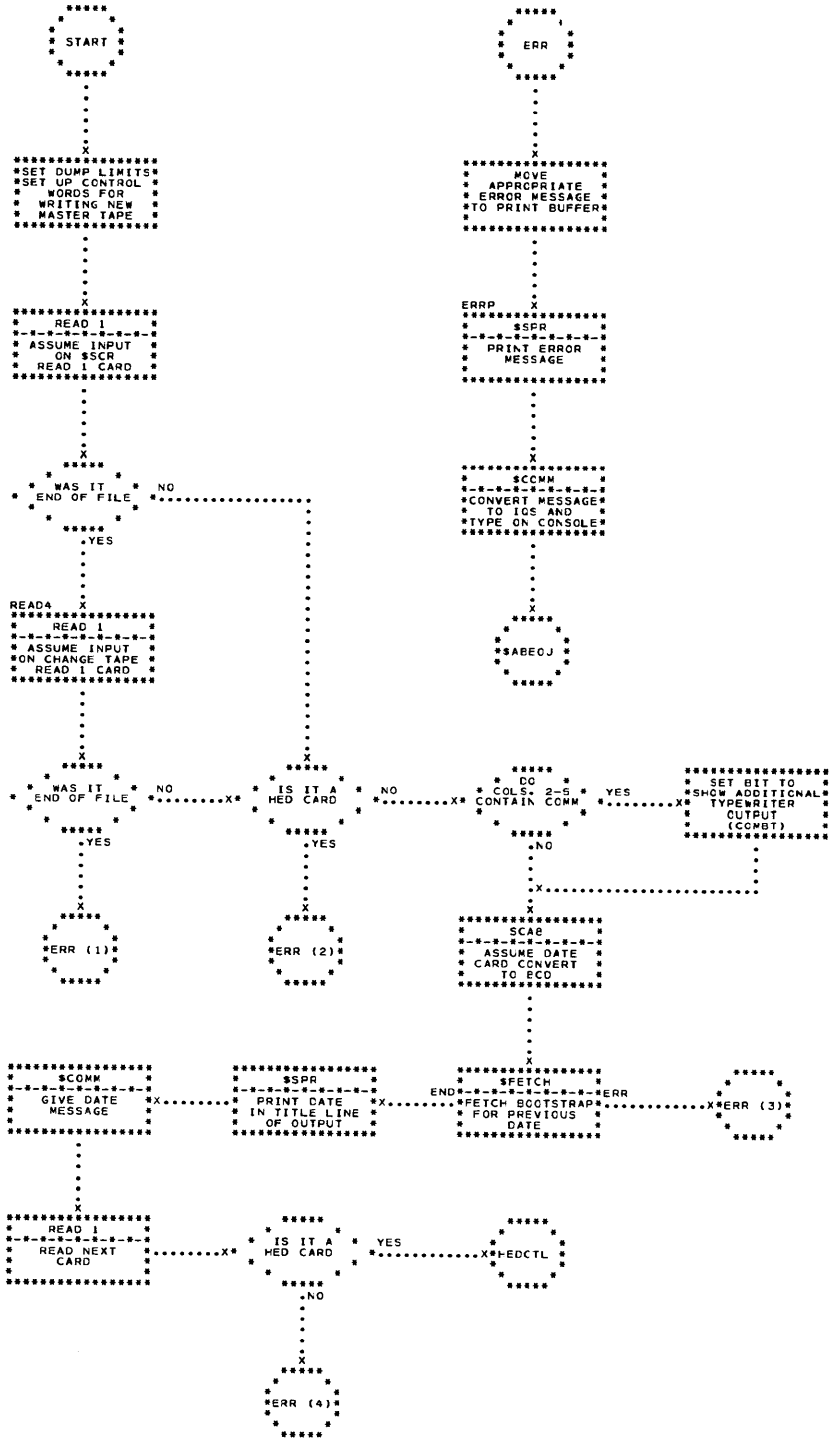


Figure 129. Update 30 - Chart 1

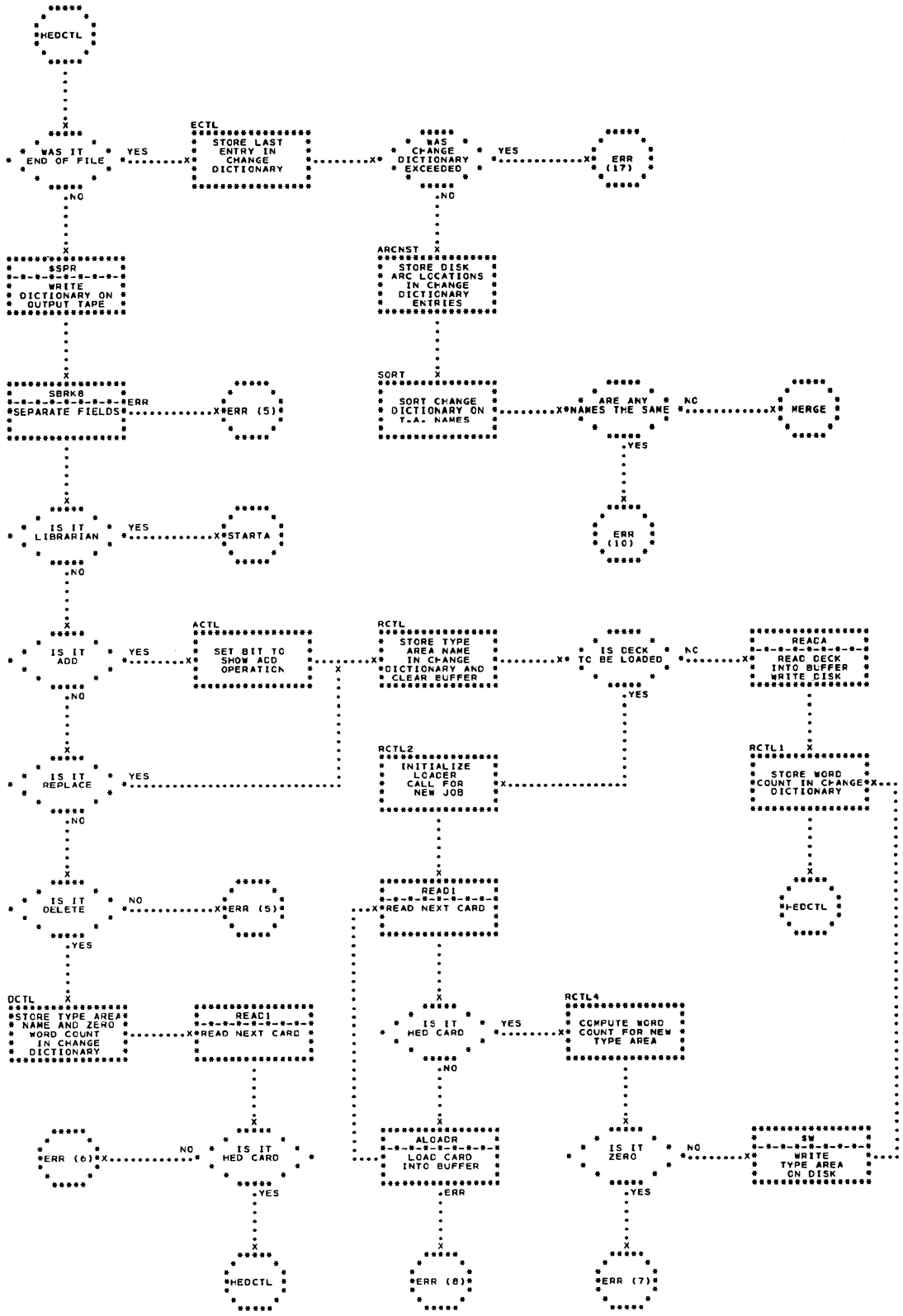


Figure 130. Update 30 - Chart 2

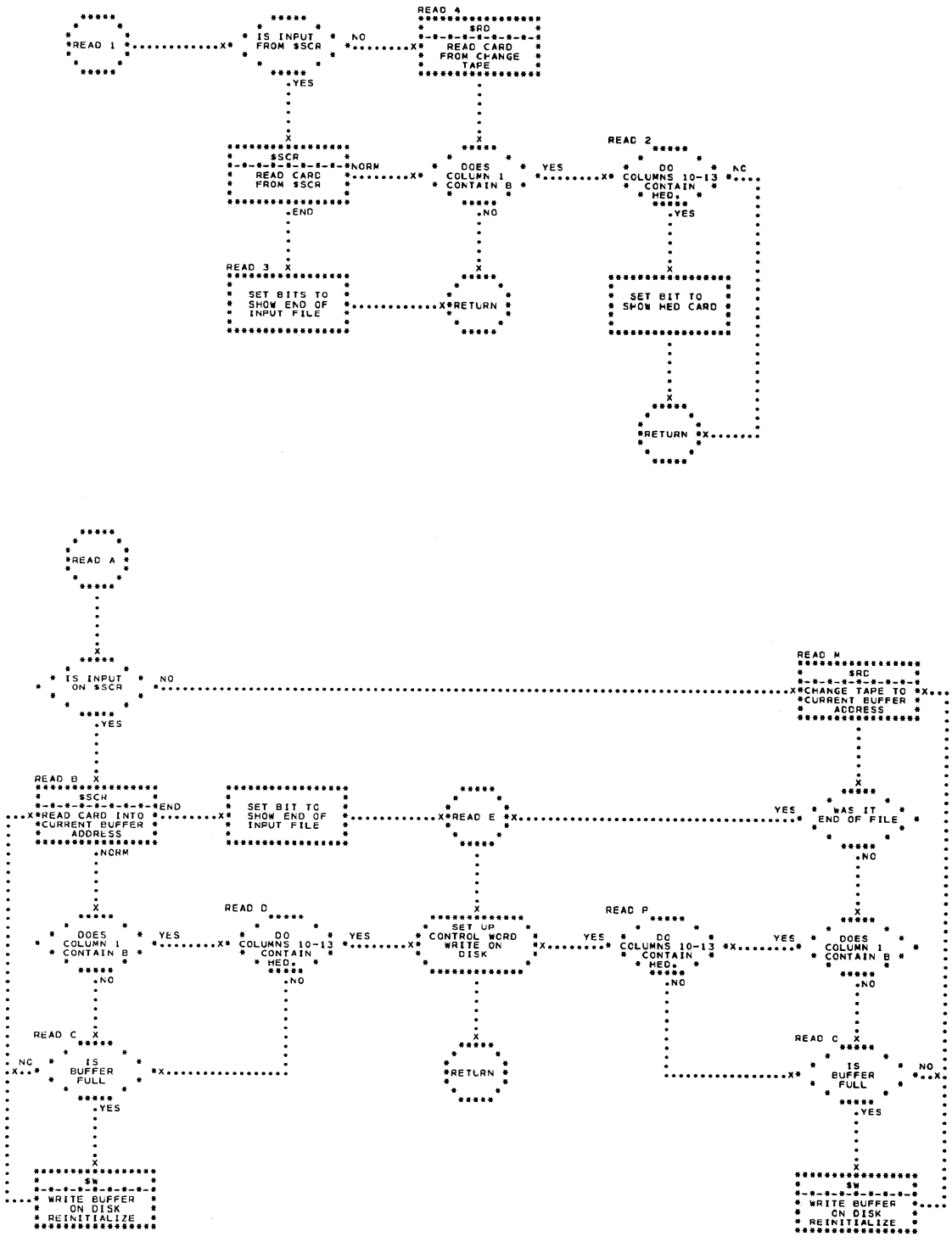


Figure 131. Update 30 - Chart 3

into a special buffer by the librarian. The new library is written on the disk as it is updated. The new table of contents is written on the disk, and the proper word count is stored in the two change dictionary entries. From this point, these two type areas are treated exactly as if they had been put on the disk by two replace operations. (See Generator Librarian.)

At ARCNST, the relative arc address of each new type area on the disk is computed and stored in each change dictionary entry. The computation is made by adding the word count of the previous entry to its relative arc address and rounding up to the next full arc. The first type area starts at relative arc zero.

At SORT, the change dictionary entries are sorted by type area name and checked for the illegal condition of two entries having the same name.

At MERGE, (Figure 132) the current dictionary is fetched from PROSA and merged with the change dictionary to form the new dictionary (stored in the NEWDIC buffer). The following illegal conditions are checked for: a delete or replace operation for a non-existing type area, and an add operation for an already existing type area.

At CHECK, the first four type area entries in the new dictionary are checked for the required MCP sequence.

At BSTRAP, (Figure 132) the new dictionary is searched for the IPL bootstrap entry. The bootstrap is read from the disk (if new) or fetched from PROSA and the current date is inserted. It is then written on the new master tape, and followed by an end of file.

At IND1, (Figure 132) the new index is constructed and the new dictionary is moved into the buffer. The move is made because the MCP requires that the entries for any given type must all be contained within the same arc of the dictionary, and that the dictionary must occupy an integral number of arcs on PROSA. The unused portion of any dictionary arc and any unused portion of the new index are cleared.

At TAAD1, (Figure 133) the size of the new dictionary is stored in the first dictionary entry, and the size of the new index (set to 40 decimal words) is stored in the second entry. The absolute arc-word address of each type area as it will appear on the new PROSA is computed and stored into each dictionary entry. The new dictionary is printed at the same time. The arc-word address for the last (dummy) entry)))))))) is rounded up to the nearest track address before being stored. The new index is moved into the buffer following the dictionary.

At ENTRY, (Figure 133) the entries in the NEWDIC buffer are used to pick up each type area following the new index and pack it into the buffer. If the type area is new, it is located and read from the disk; if not, it is fetched from PROSA. When the buffer becomes full, the contents of the buffer are written on the new master tape in blocked records with a check sum made of the contents of the buffer before writing, the current buffer position is reset, and the process continues. After the MCP IPL type area is brought into the buffer the new PROSA size is stored in the proper place and the B cards in the IPL program are printed.

The current date is stored in the bootstrap type area after it is brought into the buffer (Figure 134). When the NEWDIC entries have been exhausted, the check sum of the final portion of the buffer is made. The total check sum for the system is calculated and stored in the first word of the check sum type area))))))). and the arc count of PROSA is stored in the second word. The buffer is written on the new master tape followed by an end of file. The end message is printed and UPDATE-30 is terminated.

Input-Output: UPDATE-30 uses three I-O devices in addition to system input, system printer, and MCP commentator. They are: the disk, the new master tape, and the change tape (optional). The associated tables of exits are labeled DSKEX, NMTEX, and CHTEX. After activating an I-O device, the program always waits for a successful completion before continuing.

The EOP routine for the disk automatically keeps track of the current arc location (ARCLOC) in case a data error occurs while writing. The disk is always located before reading and the new arc location is stored in ARCLOC at the same time. The librarian uses special blocking routines for writing the disk, and changes the table of exits to PETXIT. DSKEX and ARCLOC are properly restored before the librarian returns to the main program.

Error Routines: Each error routine sets up the appropriate message in a print buffer (ERRMS) and transfers to location ERRP, which outputs the error message. The order of these routines does not correspond to the order in which the messages are listed in the program description.

Loader: UPDATE-30 includes a version of the standard MCP relocatable loader with modifications. The master update program contains a special loader that accepts absolute and relocatable decks, but not BSS decks. When the LOAD option is used, the first card loaded sets the lower bound of the program, which is the first location in the new type area. Absolute decks should begin with an origin or C card, and relocatable decks with a loader adjustment card.

The deck may contain B cards (other than JOB, COMD, or HED cards) that are read into the load buffer

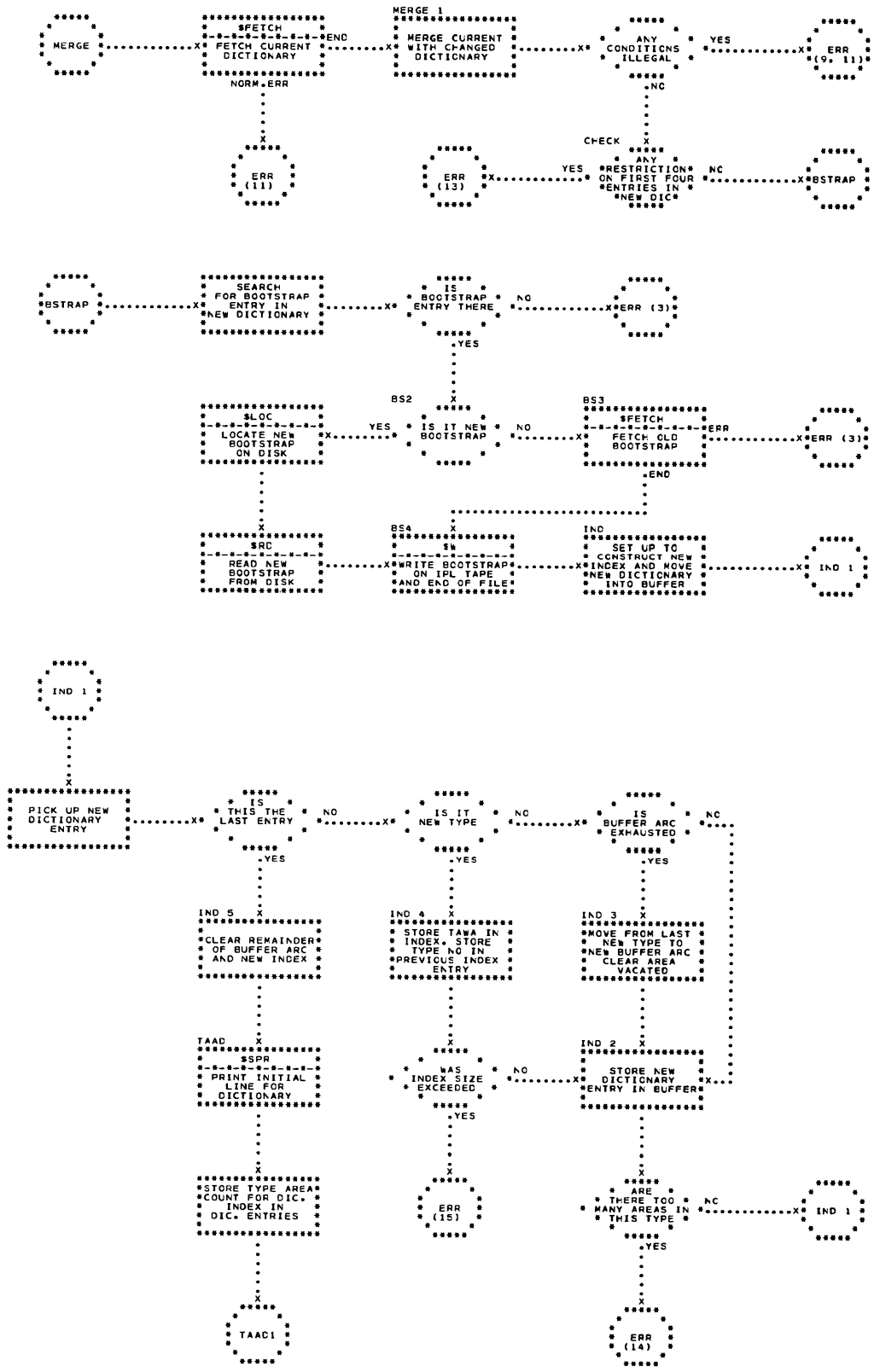


Figure 132. Update 30 - Chart 4

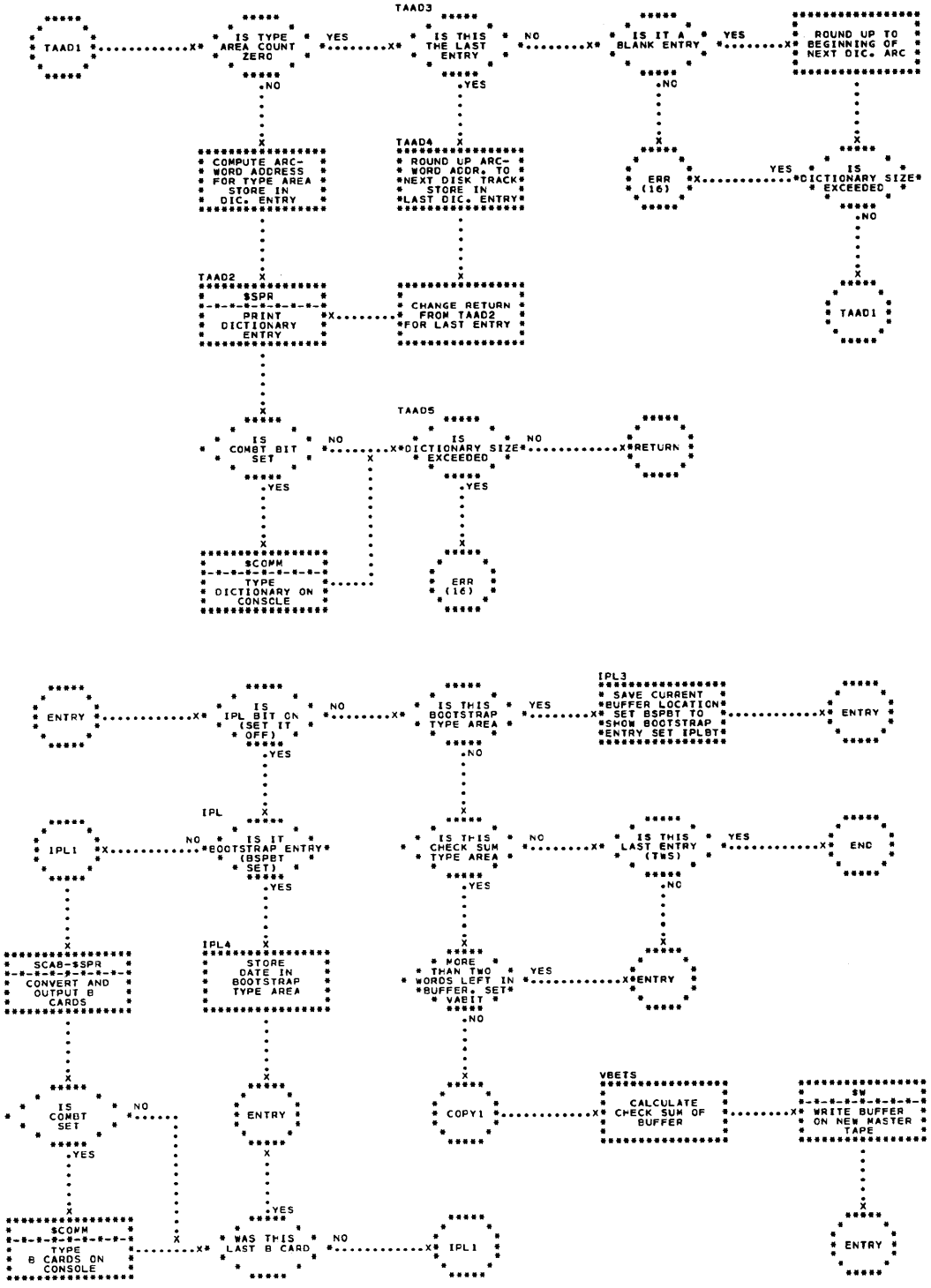


Figure 133. Update 30 - Chart 5

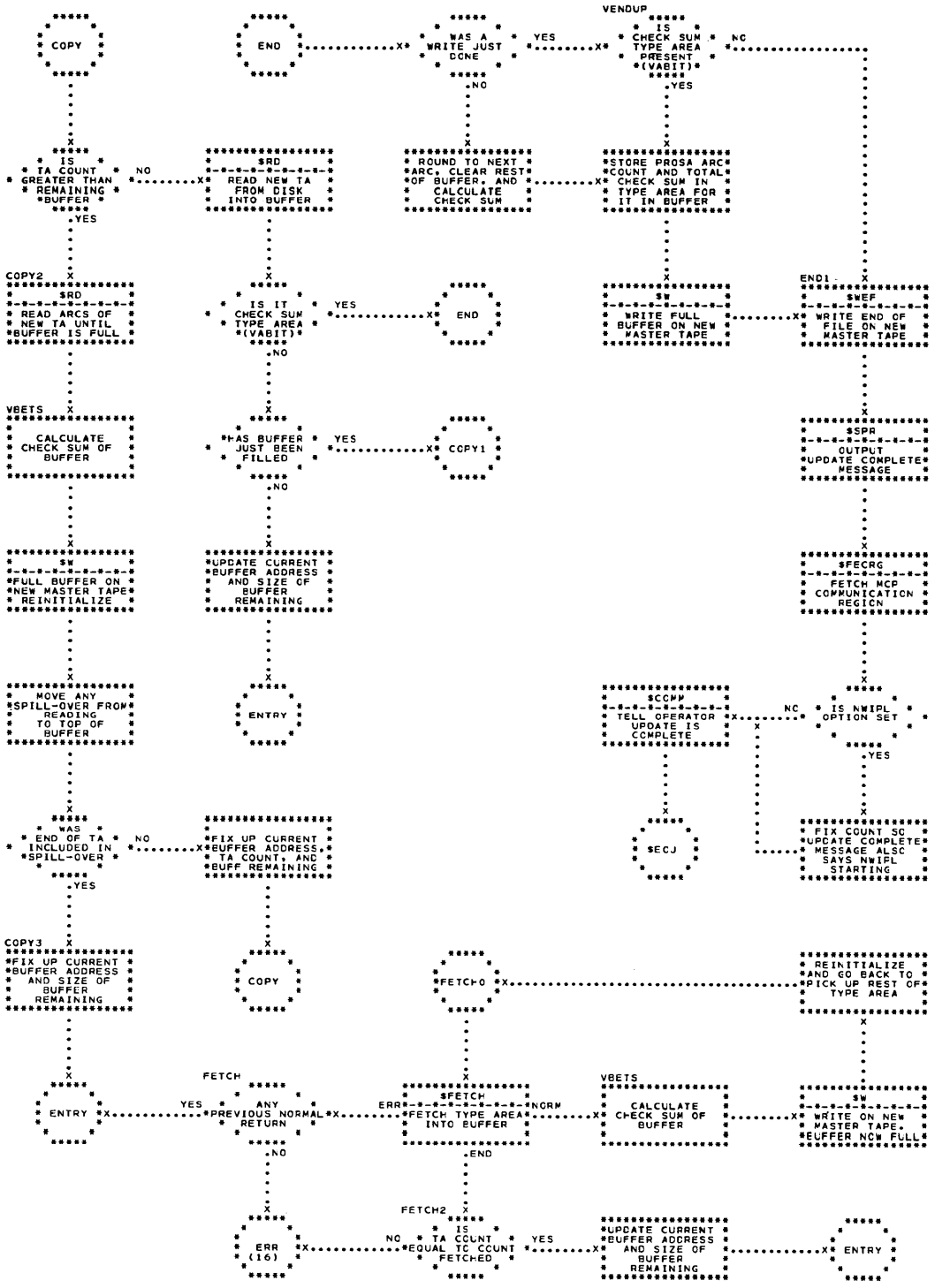


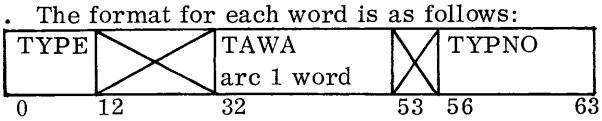
Figure 134. Update 30 - Chart 6

(15 words per card) starting with the first full word beyond the highest location loaded at that point. Any branch cards in the deck are ignored. The buffer is not cleared before loading. In all other respects the update loader operates like the standard MCP loader. A detailed write up of the MCP loader may be found elsewhere herein.

System Subroutines: UPDATE-30 uses three standard MCP conversion routines: SAxIQS, SCAx, and SBRK8, shown in Table 9.

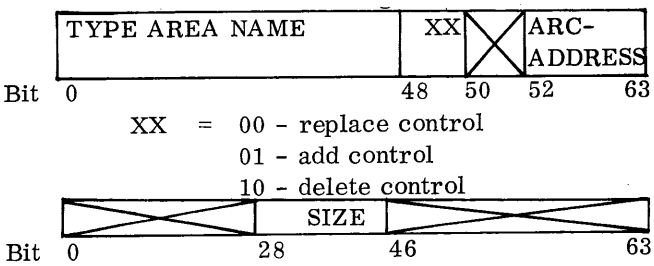
Index and Dictionary Formats: The following paragraphs describe the formats for the index, change dictionary, and the new dictionary.

1. Index - the symbol for the new index is NEWIND (new index). It consists of 40 words (plus a zero word before the table and an extra word at the end). The format for each word is as follows:



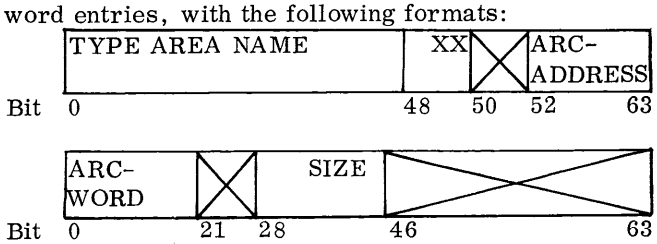
One word for each type on PROSA. (The first two characters in a type area name identify the type.) TAWA is the arc-word address of the first dictionary entry for this type. TYPNO is the number of type areas within the type.

2. Change Dictionary - The symbol for this table is CHDIC (change dictionary). It consists of 512 two-word entries, with the following formats:



One entry (2 words) for each HED card (two entries for each HED, LIB card). SIZE is the size of the new type area. ARC-ADDRESS is the location of the new type area on the disk (filled in by ARCNST routine).

3. New Dictionary - The symbol for this table is NEWDIC (new dictionary). It consists of 2048 two-word entries, with the following formats:



One entry (two words) for each type area on the new PROSA. ARC-WORD field is zero if this is a new type area. XX bits and ARC-ADDRESS are not moved when NEWDIC is moved into the buffer. The absolute ARC-WORD location of each type-area on the new PROSA is stored in the moved and final version of the new dictionary, but not in the NEWDIC buffer.

Table 9. Standard Conversion Routines

1.	BCD to IQS	SAxIQS	LVI, 15, $+1. B, SAxIQS VF, A CF, B VF, C (return)	Conversion, "x" may be 6 or 8 for 6 bit or 8 bit BCD. "A" specifies the input location, "B" the number of characters to be converted, and "C" the output location.
2.	Hollerith to BCD	SCAx	LVI, 15, $+1. B, SCAx VF, A CF, B VF, C (return)	Conversion, "x" may be 6 or 8 for 6 bit or 8 bit BCD. "A" specifies the input location, "B" the number of characters to be converted, and "C" the output location.
3.	Breakdown	SBRK8	LVI, 15, $+1. B, SBRK8 VF, A CF, B VF, C VF, D B, ERROR (return)	Breakup a series of fields separated by commas and store them in consecutive full word locations. Input is in 8 bit BCD and output is 6 bit BCD. Blanks are right appended for fields of less than 8 characters. "A" specifies the input location, "B" the number of input characters, "C" the output location, and "D" (set by the routine) the number of output characters.

Flag Symbols

Table 10 shows flag symbols used by UPDATE-30.

Generator Librarian

The librarian section of UPDATE-30 deals with the maintenance of the four generator libraries: LFORTRAN, LPASSONE, LPASSTWO, LSMACONE. It also automatically updates their tables of contents: TFORTRAN, TPASSONE, TPASSTWO, TSMACONE. The libraries and their tables of contents each occupy a type area on disk.

Generator libraries are composed of variable length blocked records, each generator being a record. They are blocked in memory image form, after being loaded at update time. In the related type area, the table of contents keeps track of the position of each record within the generator file and is used by compilers to locate generators for expansion of FORTRAN and MACRO statements.

Binary decks of generators to be added to a library, or to replace existing generators, follow a control card in the change deck. Logical sequencing of changes within a library is important if more than one operation is performed on a single name. A library may be updated only once per update run.

The updating of a library requires four phases:

1. Initialization - All control bits are set to zero, the table of contents is fetched, and the library name is verified and stored in messages.
2. Read-in of Changes - The control cards and binary decks are read, loaded, and blocked on the new master file on the disk. Changes are made to the table of contents to reflect new record numbers within the file: size, number of entries, etc. . Error checking is done on the commands and logic to avoid duplication, illegal origin or size, etc. .
3. Merge of Old and New Libraries - At the end of the change file (B card) the old library, minus any deletions, is re-blocked after the new generators on the disk. Old generators are re-numbered to reflect new positions.
4. Cleanup and Print-out - The new table of contents is written on the disk, the print-out of the changes is made, and the table of contents is written on $SPR. Appropriate changes are made to the master update change dictionary, and control is returned to HEDCTL.

Table 10. Flag Symbols

Symbol	Set Means	Control
DSKBT	Disk in use.	Turned on before I-O call to DSK. Turned off by successful EOP.
NMTBT	New master tape being written.	Turned on before I-O call to NMT. Turned off by successful EOP.
CHTBT	Change tape being read.	Turned on before I-O call to CHT. Turned off by successful EOP.
HEDBT	Card just read in change deck was HED card (or EOF condition).	Turned on in READ1 routine. Turned off by instructions which test it.
EOFBT	Change deck - END OF FILE condition.	Turned on in READ1 routine (or EE on change tape). Turned off by instructions which test it.
DSRBT	Disk I-O was a read.	Turned on in mainstream after all disk writing has been finished.
IPLBT	IPL main program or bootstrap is being loaded into the buffer.	Turned on before ENTRY or in IPL3 routine. Turned off by instruction which tests it in ENTRY routine.
BSPBT	Bootstrap is being loaded into the buffer.	Turned on in IPL3.
FETBT	A PROSA fetch has been made.	Turned on in FETCHO. Turned off if "normal" fetch return.
COMBT	On if "COMM" in date card.	Turned on when date card is read.
VABIT	On if check sum type area available.	Turned on when check sum type area found.
VTWSBT	On if check sum only buffer entry.	Turned on in END routine.

Errors: Error conditions of any kind terminate the update. Information necessary to correct the situation is typed and written on $SPR.

1. SMADDMAC errors
   a. Name too long or blank.
   b. Name duplicated.
   c. Any error in binary deck.
2. SMREPLAC errors
   a. Name too long or blank.
   b. Name not in table.
   c. New library being generated.
   d. Attempt to replace a new or a replaced generator, or any synonym.
3. SMYNONYM errors
   a. Parent name too long or blank.
   b. Parent name not present in table of contents.
   c. Synonym too long or blank.
   d. Synonym already in table of contents.
4. SMUPDATE errors
   a. Name too long or blank.
   b. Name not in table of contents.
   c. Updating a synonym.
   d. Sort field zero or greater than $1000_{(10)}$.
5. SMDELETE errors
   a. Name too long or blank.
   b. Name not in table of contents.
   c. Deleting a new or replaced generator, or a new synonym.
6. SMRENAME errors
   a. Old name too long or blank.
   b. Old name not in table of contents.
   c. New name too long or blank.
   d. New name already in table of contents.
   e. Replacing a new, replaced, new synonym, or already renamed name.
7. SMARANGE errors
   Arrange requested during an update which included a SMADDMAC or SMREPLAC.
8. Other Errors
   a. Any command which is not recognized by the librarian.
   b. Updating a library which does not exist.
   c. Updating a library more than once per update.
   d. Table overflow. The maximum number of entries allowed is 300 per library.
   e. The following errors in a binary deck
      (1) Sequence error.
      (2) Checksum error.
      (3) Bits to load on a card greater than 23 half-words.
      (4) Origin out of range (not $1000_{(10)}$).
      (5) Flow card out of range (Range is 1000 to $3045_{(10)}$).
      (6) Origin plus modifiers out of range.
      (7) Unidentifiable card - not origin, flow, C card, or transfer.
      (8) C card out of range.
      (9) No transfer card.

Operating Procedures: The program prints the sequence number of the last correct binary card (Figure 136). Code within the librarian is divided into four parts:

1. Tables, Bits, and Buffers - All permanent accumulation of information is maintained in the table of contents for eventual replacement of the old table. Bits within the table define the type of entry, and bits within the librarian record temporary flow and control information during an update cycle.
2. Executive Routines - These routines perform the operation necessary to implement a command. They provide sequencing through the appropriate subroutines, and do the majority of the error checking. Return is to RETURN.
3. Control Routines - These routines provide continuity to the library update mechanism: i.e., they are concerned with transition between phases, initialization, flow, and cleanup.
4. Subroutines - These are service routines, reached by a calling sequence, and in general used at any time by the executive and control routines.

Tables, Bits, Buffers:

Tables

Table of Contents (TOFC) - A table of three-word entries, one entry for each statement in the library. One word at the top of the table, TCTLD, contains the number of entries, the starting address of the table, and its own address for indexing the table. The name of the entry is left adjusted with trailing blanks in IQS in the first word. The record number field (bits 0 through 17 of word 2) contains the number to be used for ZTAKE by the librarian or the compiler. The record number may reflect the old or new positions at various times during update. The sort field, bits 28 through 45, contains the binary number assigned by the SMUPDATE command. The control bit field, bit 46 through 63, contains bits used by the librarian and the compiler to identify the type of entry and current status during update. (See Bits.) The memory address field is used only during compilation by the FORTRAN compiler. The size field is created by the librarian for use by the FORTRAN compiler. The third table entry illustrates the format for a synonym, where NAME3 is a synonym for NAMEX. See Figure 135.

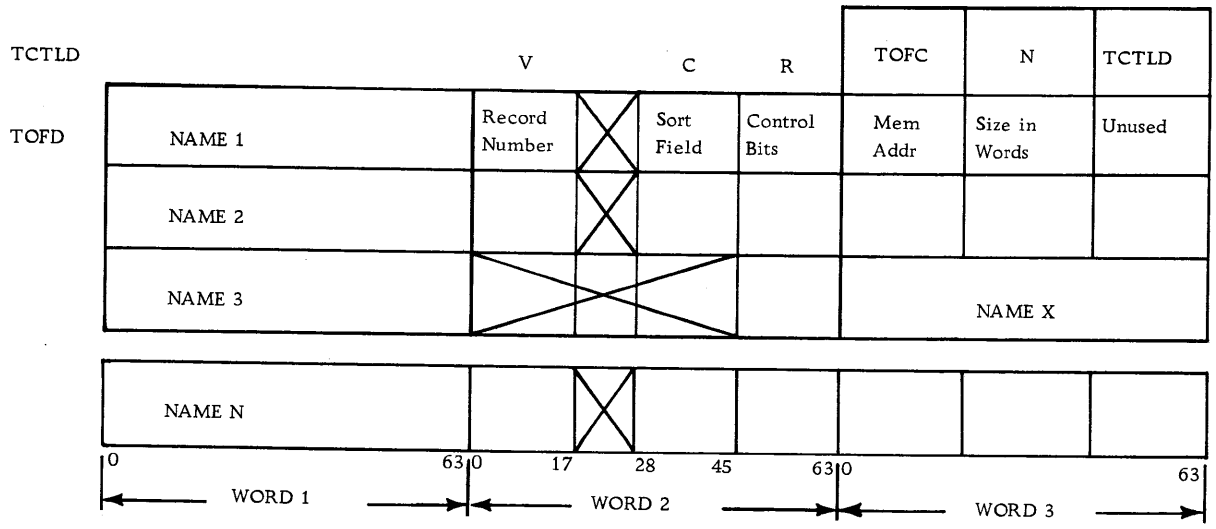


Figure 135. Table of Contents (TOFC)

Bits

There are two classes of bits used by the librarian: the PG bits and the control bits in each table of contents entry. PG bits are within the librarian code, and initialized each cycle through the librarian. Control bits are used for update and the FORTRAN compiler.

PG Bits - PG is a word reserved in memory for use as switches, indicators, etc. by the librarian. Bit meanings are:

PG 1 - There has been a SMADDMAC or SMREPLAC command accepted. Do not allow a SMARANGE command. Set by RETURN, tested by ARNG and ARANG.

PG 5 - A SMARANGE command has been accepted. Set by ARANG, tested by TAPMRG.

PG 7 - Print "added" heading. Additions have been made to the file. Set by ADDA, tested by EXIT.

PG 8 - Additions have been made to generator file. Set by PUTT, tested by TAPMRG.

PG 9 - Deletion of a generator has taken place and re-numbering of the generator file is necessary. Set by DELETE and REPLC, tested in TAPMRG.

PG 10 - A rename has taken place. Set by RENAME, tested by EXIT.

PG 11 - A replace has taken place. Set by REPLC, tested by EXIT.

PG 12 - Set by REPLC to cause return from JLOAD. Set and turned off by REPLC, tested by ADDA.

PG 13 - A synonym has been added. Set by SYNM, tested by EXIT.

PG 14 - Print "deleted" heading. Set by DELETE, tested by ALPH.

PG 15 - A synonym has been deleted. Set by DELETE, tested by EXIT.

PG 20 - This is a new library. No old library file exists. Delete and replace operations are illegal. Final merge is skipped. Set in STARTA, tested in TAPMRG and ARANG.

PG bits are initialized to zero in STARTA by a Z, PG instruction.

Control Bits - Bits 1.47 through 1.55 in each table of contents entry are used to maintain information relating to a particular generator for use by the librarian and the compiler. Control bits are stored temporarily or permanently as specified:

1.47 - Delete this entry. Set by DELETE, tested in ALPH. (temporary)

1.48 - This generator has been added. The record number in bits 1.0 through 1.17 is the number on the NEW file. Set by ADDA, tested by EXIT and TAPMRG (temporary).

1.49 - This generator has been replaced. Record number refers to new file. Set by REPLC, tested by EXIT and TAPMRG (temporary).

1.50 - This synonym has been added. Set by SYNM, tested by EXIT. (temporary)

1.51 - This entry has been renamed. Set by RENAME, tested by EXIT. The new name is in the third word (temporary).

1.52 - This entry has synonyms. Set by SYNM, tested by DELETE, RENAME and the FORTRAN compiler (permanent).

1.53 - This entry is a synonym of the parent name in word three. Set by SYNM, tested by UPDATE, REVUE, REPLC, ALPH, ARNG, and the FORTRAN compiler (permanent).

1.54 - Used by ARNG to indicate a generator has been arranged.

1.55 - Used by TAPMRG to indicate a generator has been merged.

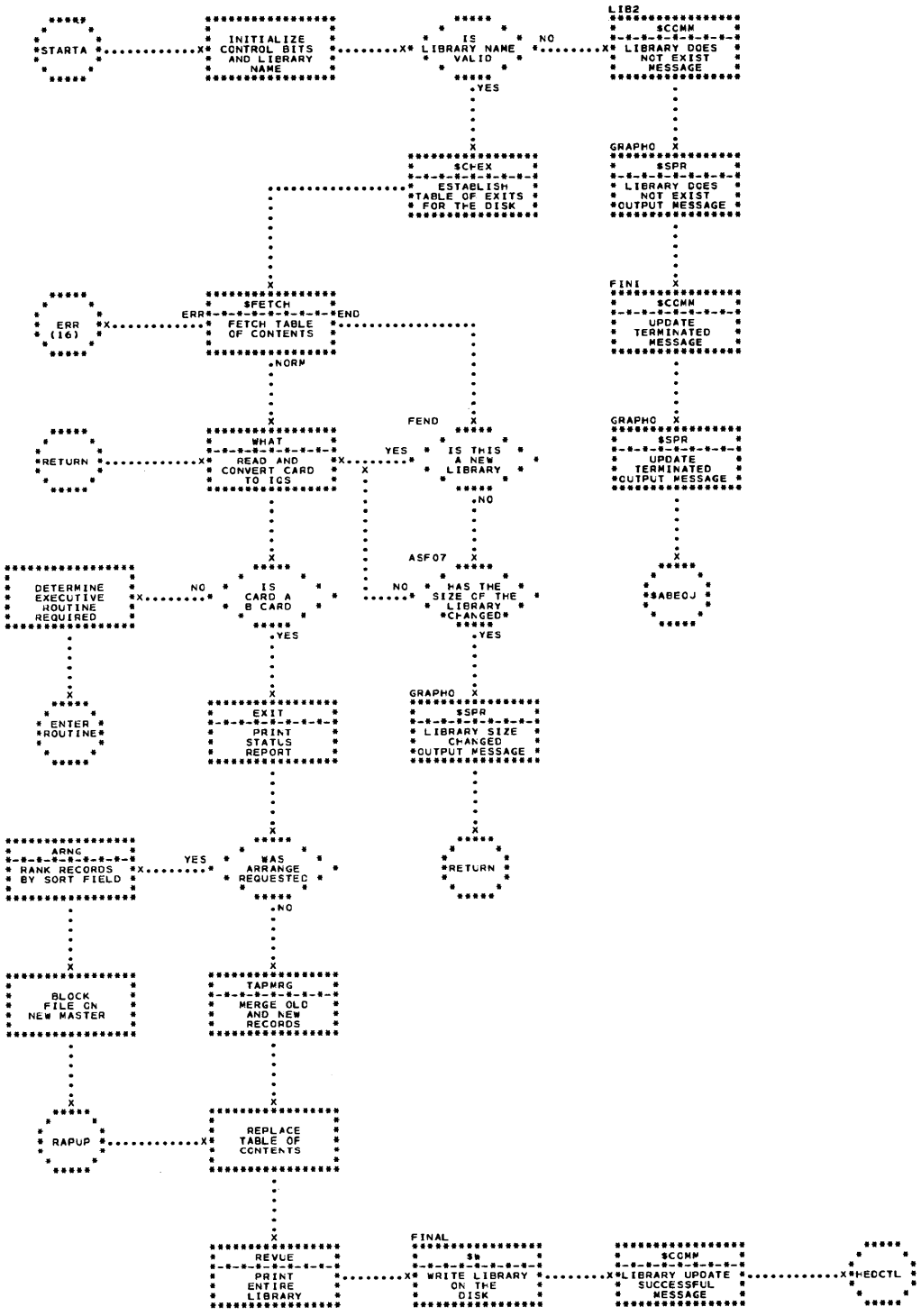


Figure 136. Generator Librarian - Chart 1

Buffers

ABUF - A three-word buffer in which entries to the table of contents are built up for subsequent entry by ADNTY. Used by ADDA, SYNM, REPLC, and RENAME.

SYNM5 - A two-word buffer in which the scan routines, ONFDR and NNFDR, build up a name to be examined by FDTNM.

IOWRD - a 3001-word buffer used for intermediate storage of a generator before or after blocking or unblocking. The first word contains the length of the record, exclusive of the first word.

AXX - The buffer into which command cards are converted and packed by WHAT. Used by RETURN and the scan routines.

RBUFER - A nine-word buffer used to build a print image.

Executive Routines:

1. Add (ADDA) - Figure 137 - Adds a generator to the library, checking for duplication and illegal format. Controls sequence through subroutines NNFDR, FDTNM, JLOAD, ADNTY and PUTT. ADDA turns on PG bit 7 and table of contents bit 1.48, tests and turns off PG bit 12, and exits to RETURN.

2. Replace (REPLC) - Figure 137 - Replaces a generator already in a library, checking for validity and logic. Controls sequencing through subroutines NNFDR, FDTNM, JLOAD, PUTT. It tests PG bit 20, and turns on PG bits 9, 11, and 12 and table of contents bit 1.49, tests table of contents bits 1.48, and 1.50, and exits to RETURN.

3. Delete (DELETE) - Figure 137 - Deletes a generator and any synonyms from a library. Deletion is immediate, and the name may be used later in the change deck. Checks for validity and logic. Controls sequence through NNFDR, FDTNM. Turns on PG bits 9, 14, and 15 (if necessary); table of contents bit 1.47, tests table of contents bits 1.48, 1.49, 1.50, 1.52, 1.53, and exits to RETURN.

4. Update (UPDAT) - Figure 137 - Enters an arbitrary numeric sort field into the table of contents. Checks for validity and range of number. Controls sequence through ONFDR, FDTNM, tests table of contents bit 1.53, and exits to RETURN.

5. Arrange (ARANG) - Figure 137 - Sets bit for eventual arrangement of records by sort field. Checks for illegal combination, such as SMARANGE used with SMADDMAC or SMREPLAC. Sets PG bit 5, tests PG bits 1 and 20, and exits to RETURN.

6. Synonym (SYNM) - Figure 138 - Enters a macro into the table of contents, making it a "synonym" of another macro. The "parent" macro may not be another synonym. It checks for duplication and logic, and controls sequence through ONFDR, FDTNM, NNFDR, FDTNM, ADNTY. Turns on PG bit 13, table of contents bits 1.50 and 1.53 in new entry, 1.52 in parent, and exits to RETURN.

7. Rename (RENAME) - Figure 138 - Changes the name of a generator or synonym. Changes synonyms of the generator to synonyms of the new name. Checks for validity, duplication, and logic. Controls sequence through ONFDR, FDTNM, NNFDR, FDTNM, ADNTY. Turns on PG bits 10 and 15, table of contents bits 1.47 and 1.51 in the old entry, tests table of contents bits 1.48, 1.49, 1.50, 1.51 in old entry, and exits to RETURN.

Control Routines:

1. Return (RETURN) - Reads and converts a card from the change deck to IQS code. By a series of COMPARE instructions, it determines the operation to be performed, and branches to the appropriate executive routine. Uses subroutine WHAT, and turns on PG bit 1. Exits to EXIT when file is exhausted or to ERROR 1 if command is not recognized.

2. Exit (EXIT) - Figure 139 - Start of the final phases of the library update. It produces a print-out of the changes that have taken place, using the PG bits (7, 10, 11, 13), the table of contents, and GRAPHO. Uses ALPH to clear the deleted entries (after print-out) and alphabetize the new table of contents. Turns off table of contents bits 1.51, 1.50. Exits to TAPMRG.

3. Alphabetize (ALPH) - Prints deleted entries using GRAPHO (in conjunction with EXIT) and clears the entry from the table of contents. Subtracts one from the count of entries for each deletion. The second part, UPD17, tests for an empty table of contents condition, and alphabetizes the table. Tests PG bits 10, 14, and 15, turns off table of contents bit 1.52, and exits to ALPHX, within the EXIT routine.

4. Tape Merge (TAPMRG) - Figure 140 - Performs the merge of old and new library files. If there have been no deletions from the old file, the old file is moved to the new file following any additions. Record numbers of old records are increased by the number of additions. If records have been deleted (or replaced), the table of contents is searched for each record number to determine its validity, and then the record number is added to the new file. Uses ZTAKE and PUTT. Tests PG bits 20, 9, 8, tests and turns off bit PG 5. Tests and turns off table of contents bits 1.48, 1.49, 1.55, tests table of contents bit 1.53, and exits to RAPUP.

5. Arrange (ARNG) - Figure 139 - Sequences generators on the new file by ascending sort fields. Generators with the same value are arranged alphabetically. Records starting with value zero are fetched from the old file sequentially and written on

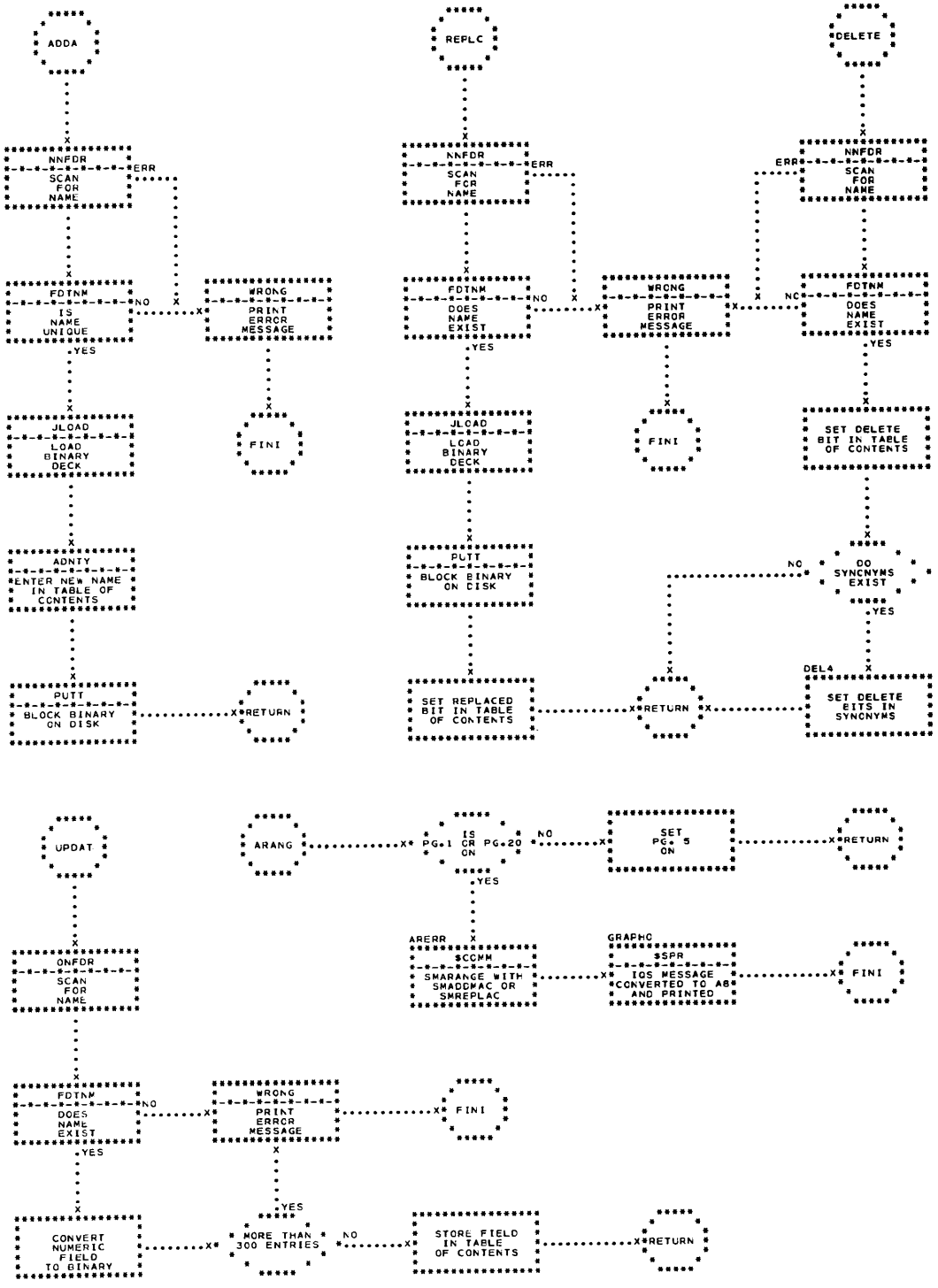


Figure 137. Generator Librarian - Chart 2

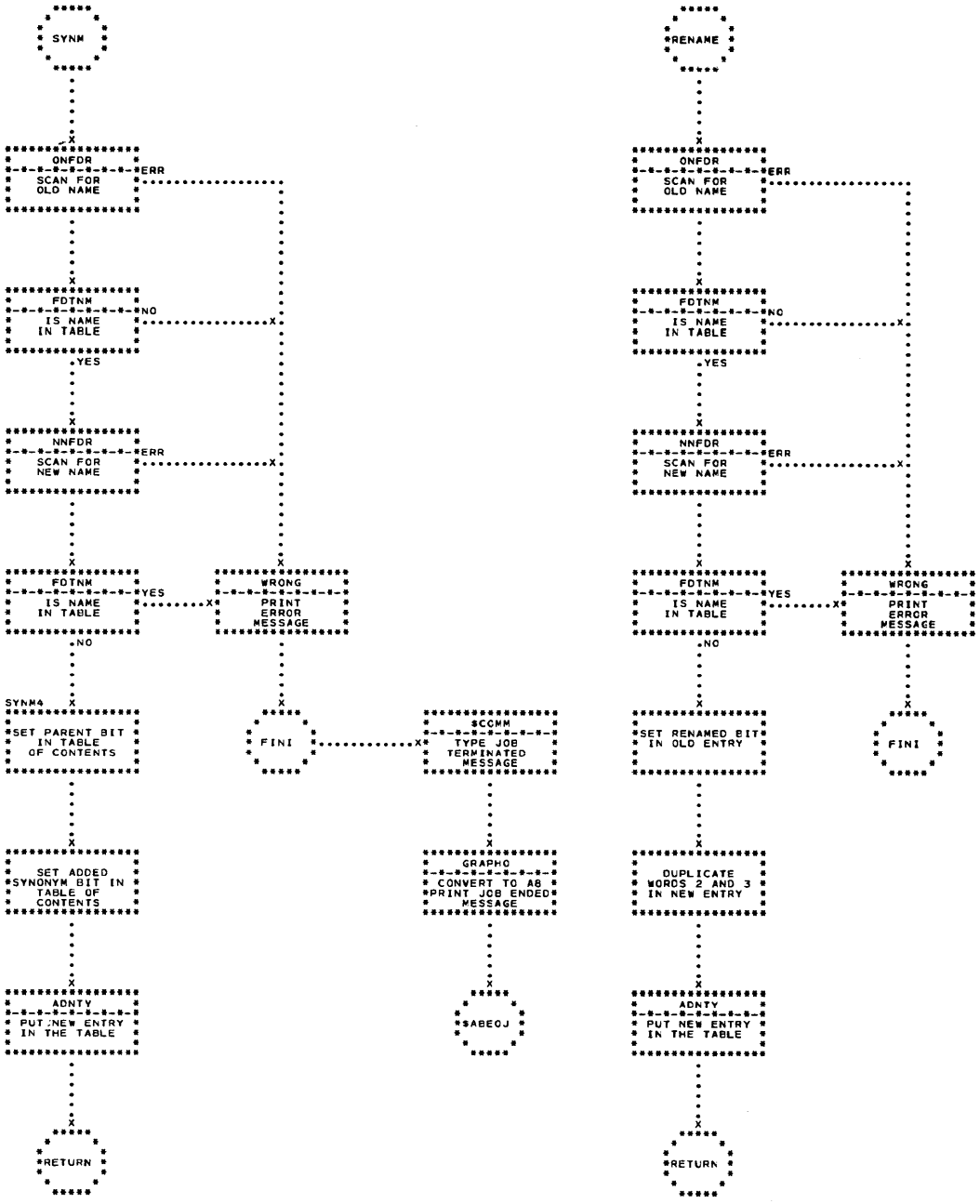


Figure 138. Generator Librarian - Chart 3

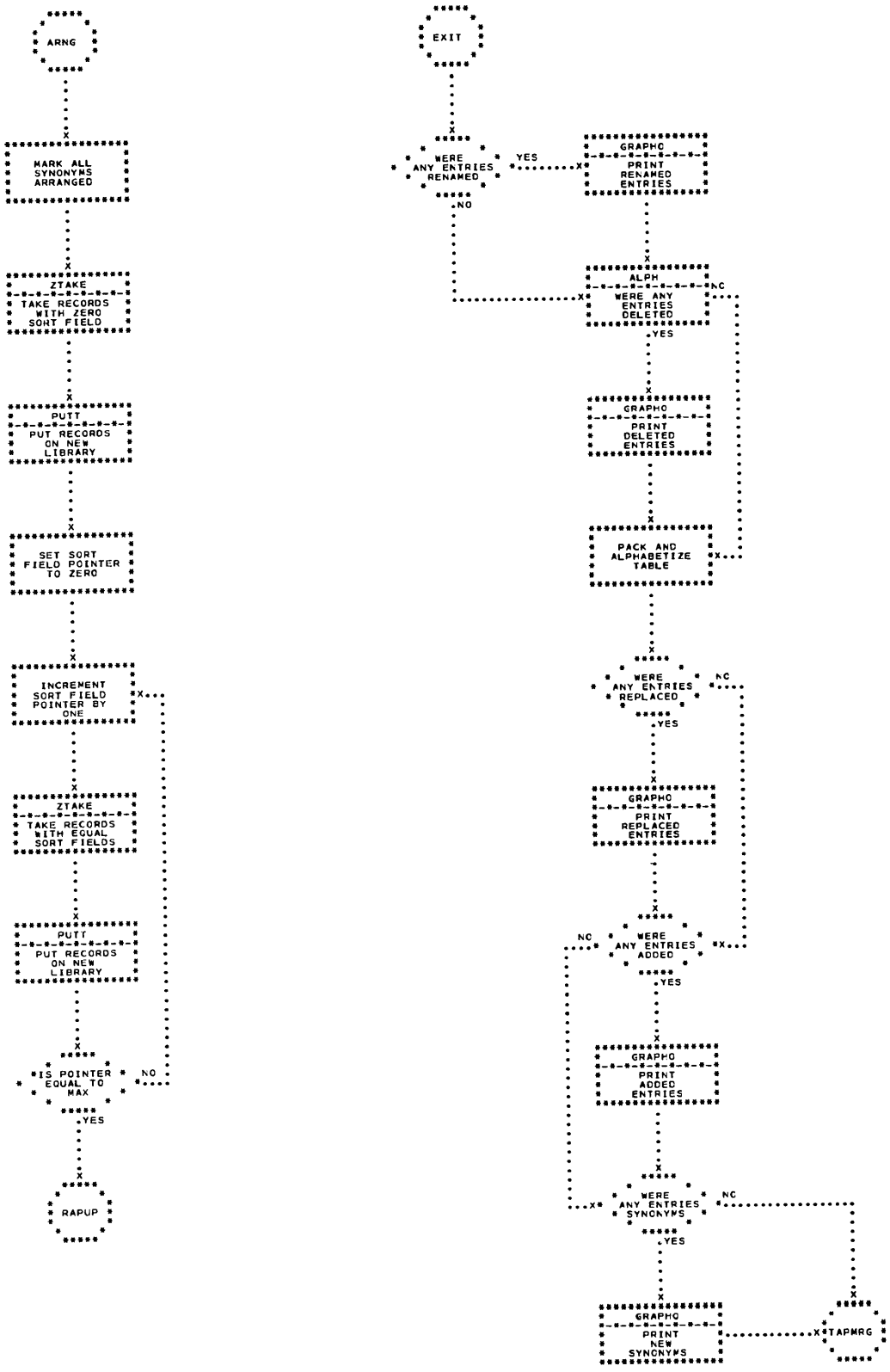


Figure 139. Generator Librarian - Chart 4

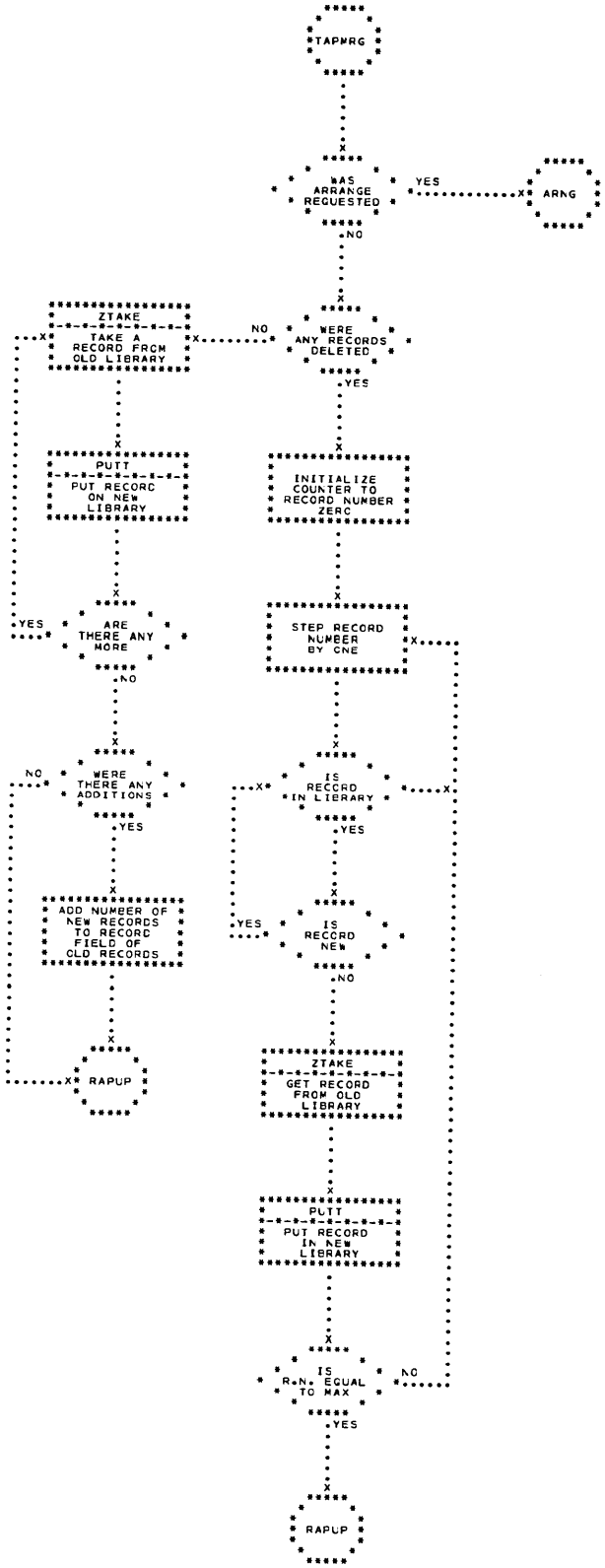


Figure 140. Generator Librarian - Chart 5

the new. Tests PG bits 1, 8, tests table of contents bits 1.48, 1.49, 1.53, tests and sets bit 1.55 and exits to RAPUP.

6. Start (STARTA) - The point to which control is relinquished from the main section of master update. Initializes symbol file tables, clears all PG bits, fetches table of contents, and checks validity of library being updated. Initializes TCTLD, clears PUTER, changes DSK exit to PETXIT, and generates change dictionary entries.

7. Wrap Up (RAPUP) - Final phase of librarian update. Forces the write-out of any partial buffers of blocked generators, establishes ARCLOC, and changes DSK exit to DSKEX. Writes the table of contents on disk, types the final message, and returns to HEDCTL.

Subroutines:

1. New Name Finder (NNFDR) - The new name finder subroutine scans AXX for a single name, or the second of two names. It checks for a blank field or for parameters greater than 8 characters. It also builds the name left justified in SYNM5 and assumes $2 points to the first character of the parameter, and that blank is the terminating character.

```
Entry sequence: LVI,$1,$+1.
                B, NNFDR
                B,              Error return
                - - - - - -     Normal return
```

2. Old Name Finder (ONFDR) - This subroutine scans AXX for the first of two parameters and checks for a null field and parameter larger than 8 characters. A comma is the terminating character. It builds parameter left justified in SYNM5. If an error is detected, it exits to WRONG.

```
Entry sequence: LVI,$1, $+1.0
                B, ONFDR
                - - - - - -     Return
```

3. Add Entry (ADNTY) - Adds an entry to the table of contents. Entry is assumed to be 3 words starting at location ABUF. It increases the count of entries in the table of contents (TCTLD+.28) and in addition, checks the total number of entries for overflow, (Maximum is $300_{(10)}$), with exit to OVRFLO if an overflow condition exists.

```
Entry sequence: SIC, ADNTYX
                B, ADNTY
                - - - - - -     Return
```

4. Find Table Name (FDTNM) - Performs a linear search of the table of contents for the name in SYNM5. It checks for zero entries and deleted names. XR3 points to the entry if found.

```
Entry sequence: LVI,$1, $+1.0
                B, FDTNM
                B,              Not found return
                - - - - - -     Found return
```

5. What (WHAT) - Converts Hollerith card code to IQS format, packing 72 columns left justified in AXX. Using READ1 to read, it checks HEDBT for an end of file, exiting to EXIT if HEDBT is on. The converted Hollerith characters are moved to AXX from CARD. The first eight characters after the last non-blank are made zero.

```
Entry sequence: LVI,$1,$+1.0
                B, WHAT
                - - - - - -     Return
```

6. Graph O (GRAPHO) - Converts a given number of IQS words to A8 format and prints a $SPR line. Checks COMBT for $COMM option.

```
Entry sequence: CNOP
                SIC, $15
                B, GRAPHO
                XW, 1st character,
                    # of words,
                    carriage control
                - - - - - -     Return
```

7. PUTT - Blocks one record, starting at IOWRD, on the new library file. Turns on PG bit 8, and keeps a running account of the number of records in the new file (PUTER). This number is stored in the record number field of the table of contents entry. Uses ZPUT. XR2 is assumed to point to the entry.

```
Entry sequence: SIC, PUTX
                B, PUTT
                - - - - - -     Return
```

8. Generator Loader (JLOAD) - In addition to normal loading, it checks for origin, which must start at $1000_{(10)}$, with subsequent cards not exceeding $2045_{(10)}$, and the absence of P, T, or super T cards. There are 9 error exits for illegal binary decks, JER1 through JER9. It uses READ1 to read cards and exits to DABLE in ADDA routine.

9. Wrong (WRONG) - Common error print-out routine. Prints the entire command and the parameter error or logical error message on both $COMM and $SPR (using GRAPHO). Exit is to FINI. (WRONG has 3 synonyms: ERROR, GOOF, and ERROR1.)

10. Final Error (FINI) - Final error message print-out for all error messages. Prints on $COMM and $SPR, and exits to $ABEOJ.

11. Blocking Routines - The blocking routines, ZPUT, ZTAKE, and ZENDFILE are the I-O routines that fetch the old library file through $FETCH, and write the new library file on the disk using DSK IOD. They maintain information about the two files in tables called DFILE and FILE1.

ZPUT - Moves a logical record from IOWRD to a buffer (BUFF). When records fill a buffer, 2045 words, the routine writes the block on disk, updates ARCLOC and the count of words within the file. (1.28($5))

Entry sequence: LVI, $14, $+1.0
B, ZPUT
T1, 0, DFILE1, IOWRD
- - - - - - Return

ZTAKE - Transmits a requested record from BUFF to IOWRD. Uses $FETCH to read a block at a time from PROSA. When the record number requested is not in the block currently in memory, it fetches another block until the record is found or the END return is taken from $FETCH. Should the END condition occur, it returns to the librarian at the address found in NEXT+2.

Entry sequence: LVI, $14, $+1.
B, ZTAKE
SWAP, $XR, DFILE, IOWRD
- - - - - - Normal return

ZENDFILE - Forces the write-out of a block even though it is partially empty. Writes an end of file after the block.

Entry sequence: LVI, $14, $+1.
B, ZENDFILE
, DFILE1
- - - - - - Return

PERIPHERAL INPUT-OUTPUT

Peripheral input-output programs consist of four IBM 1401 programs:

1. Tape Labeling
2. System Peripheral Input I
3. System Peripheral Input II
4. System Peripheral Output

These programs prepare tape labels and input tapes to be used by MCP. In addition, the System Peripheral Output program prints or punches data from MCP-generated output tapes.

Tape Labeling (1401)

The IBM 1401 Tape Labeling program labels tapes used by the MCP. Input for the label files is supplied by a card reader. A label file consists of 11 six-bit characters written in high-density, BCD code, using the even parity mode. Label cards follow the labeling program and specify the labels to be written. There is one label card for each tape.

The format for a label card is as follows:

Column 1	L punch (all non-L cards are ignored).
Columns 2 through 9	Blank.
Columns 10 through 14	Label (5 BCD characters)
Column 15	Density (H=high, L=low). If column 15 is blank, an H for high density is inserted by the program.
Column 16	Mode (O=odd, E=even, C=odd with ECC). If column 16 is blank, odd parity is assumed and an O is inserted.
Columns 17 through 24	Blank.
Columns 25 through 80	Information contained in these columns is printed but not recorded on tape.

Operator Instructions

1. Ready the card reader, printer, and console; place Sense Switch A in the on (up) position and I-O Check Stop switch in the off (down) position. (Sense Switch A must be on because it is used as the last card indicator.)
2. Ready the first tape to be labeled and all succeeding tapes in turn, on unit 1 in high density.
3. Place the labeling program deck and label cards in the card reader hopper, then push Load key.

NOTE: IBM 1401 binary tapes are always written in the odd parity mode, therefore, if the tape is to be used by the 1401 System Input Program I, column 15 must contain an H for high density and column 16 must contain an O for odd parity mode.

Stop Conditions

Non-Error Stops: Two non-error stops are possible:

1. STOP 1 - After a label is written on tape and concluded with an end of file, the tape is rewound and unloaded. The information on the label card associated with the tape is printed by the 1403 printer, in addition to operator instructions as follows:
   a. Write the above information onto outer label of tape on unit 1.
   b. Ready new tape on unit 1 - High Density.
   c. Push Start to continue.
2. STOP 2 - A message is printed to end the operation.

Error Stop: Only one error stop may be encountered.

1. ESTOP - A card reader error has occurred. The last card in the N/R stacker should be checked, repunched if necessary, and reloaded into the hopper. When an error card is reloaded, make certain proper card sequence is maintained.

Program Flow

Label cards are read, partially checked (columns 15 and 16), and the information in columns 1 through 80 printed by the printer. See Figure 141. Columns 10 through 20 are recorded on tape with an end of file

mark and the tape is unloaded. Operator procedures are printed and a temporary stop condition occurs.

System Peripheral Input I

System Peripheral Input I is a 1401 program used to create a system input tape for MCP. The tape is blocked into 17 card records. Jobs are separated by file marks, with a limit of N jobs per tape, where the nominal value of N is ten. (See the MCP System Input Program.)

The first card of each job must be a B card (a B punched in column 1), with either JOB or COMD starting in column 10. This type of card separates jobs and causes end of file marks to be written on the input tape. When the last card of the last job has been written on tape, two end of file marks are written on the input tape. If the I-O check switch (on the console) is in the off (down) position, tape and reader errors are handled internally by the program. The only input source for the program is the card reader.

Input Deck

The input deck to be written must consist of a sequence of jobs, each job starting with a job card. COMD cards may occur only between jobs (ahead of a job card).

Operator Instructions

1. Ready scratch tape (with a label) on tape unit 2 in high density. This will now become the input tape.
2. Ready the printer, reader and console. Place I-O check switch off, and Sense Switch A in the on position. NOTE: Sense Switch A, the last card indicator, must always be in the on (up) position.
3. Load the reader hopper with the system input program, followed by the jobs.
4. Push the reader Load key.

Programmed Halts

STOP 1 - The last card of the last job has been read, end of file marks are written on the input tape and the tape is unloaded.

ESTOP 1 - A card reader error occurred during the loading of the object deck. For this condition, there is no program recovery. To restart, the card reader must be cleared, and the system input program reinserted in the card hopper and the Load key pressed. Proper card sequence must be maintained. If the error condition is repeated, the programmer should reload the program with the I-O Check Switch in the on (up) position to ascertain which card is incorrect.

ESTOP 2 - A card reader error has occurred during the reading of the first card. The error card falls into the normal read stacker, and a message is printed by the printer. To continue, the card reader must be cleared, and the error card plus all succeeding cards placed in the hopper with the error card first. Proper card sequence must be maintained. The printed message is: CARD READER ERROR RESTART WITH CD IN NORM STACKR.

ESTOP 3 - The first card read was not a B-JOB or B-COMD card. An error message is printed and the operation stops. The message is as follows: THE FIRST CARD IS NOT A B-COMD OR B-JOB CARD, REJECT THIS JOB AND RESTART WITH THE NEXT JOB. This error condition may be corrected by having the operator punch a B-JOB or B-COMD card and reinsert the job in the reader hopper. If no correction is made, the remainder of the job must be removed and the program restarted with the next job.

ESTOP 4 - A card reader error has occurred during the reading of any card after the first. The error card falls into the normal read stacker, and a message is printed by the printer. To continue, the card reader must be cleared, and the error card plus all succeeding cards placed in the hopper with the error card first. Proper card sequence must be maintained. The printed message is as follows: CARD READER ERROR RESTART WITH CD IN NORM STACKR.

ESTOP 5 - During the writing of tape, if it becomes necessary to erase tape more than three consecutive times to rewrite the same record, ESTOP 5 occurs. This stop condition may be construed as an equipment malfunction. The following message is printed: TAPE TROUBLES, RELOAD - RESTART ON NEW TAPE DRIVE.

ESTOP 6 - When an end of reel condition occurs, a printed message shows which job must be rewritten on a new loaded tape. The message is as follows: END OF REEL, LOAD NEW LABELED TAPE ON UNIT 2. RELOAD PROGRAM AND JOBS STARTING WITH JOB .

Program Flow

When the system input program has read the first card, the buffers are reset and the tape label passed. See Figure 142.

If the first card containing a B in column 1 is a job or command card, and is error free, a branch to R3 occurs. The branch to R3 sets C1 into the instruction and the first card is moved into the buffer. If the card read is not the 17th card of the record, C1 is stepped one count and the program is looped back to R1.

If a card has a B punch, a branch to WRT is again performed to determine if the card is a new job or

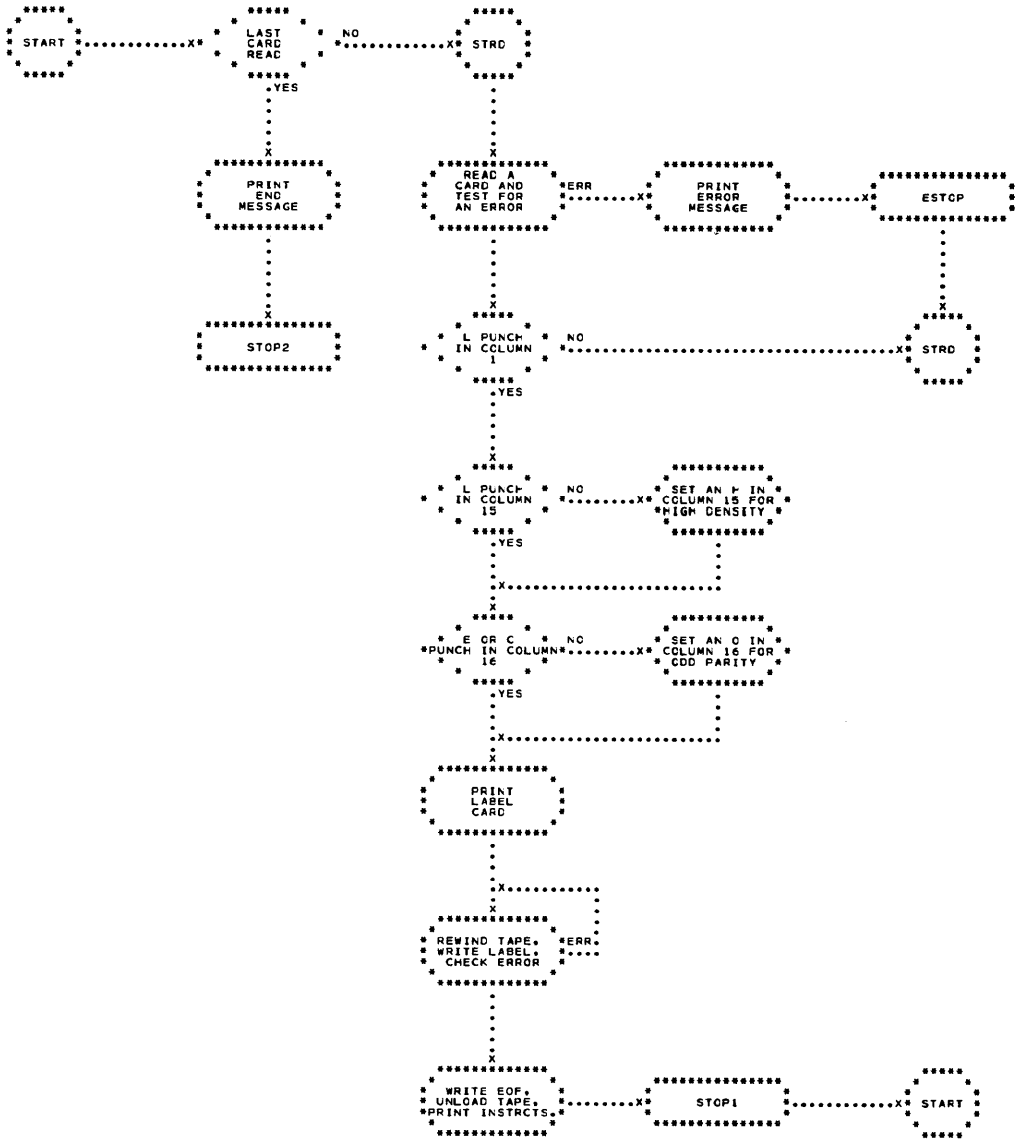


Figure 141. Tape Labeling Program

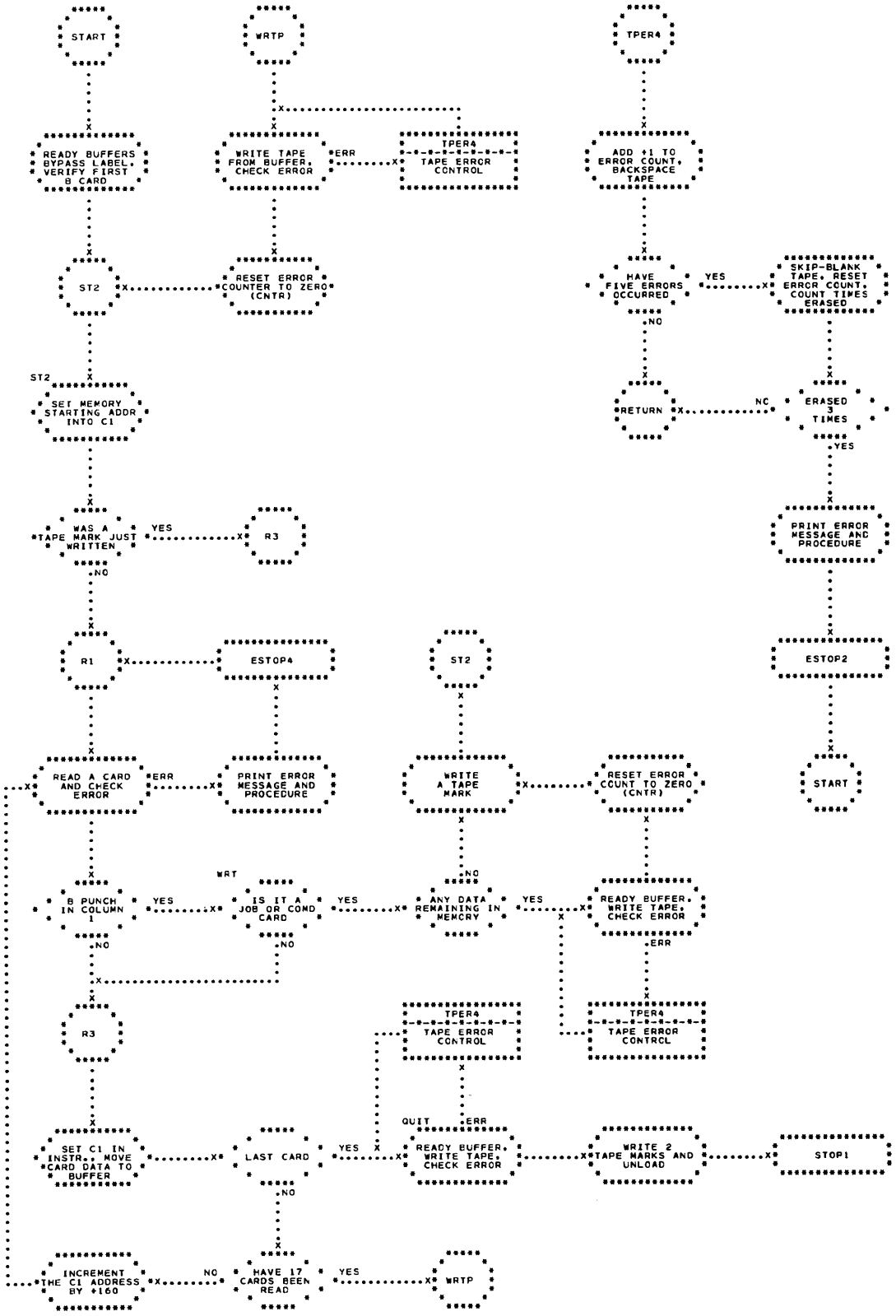


Figure 142. System Peripheral Input I

command card. If the card is neither a job card nor a command card and re-entry is made at R3, the card is entered into the buffer and C1 is sequenced. When 17 cards have been read, a tape writing procedure is set up, the contents of the buffer are written on tape, the error counter is reset to zero, and re-entry made at ST1.

When a new job or command card is encountered, a check is made to determine if any data remains in the buffer from the previous job. If so, the remaining data is written on tape, the error counter is reset and a tape mark (end of file) is recorded on the tape. When the last card from the hopper is detected, the program branches to QUIT, and a group and word mark are set into the buffer. The contents of the buffer are written on tape followed by two tape file marks, and the program halts.

A message is printed if an error, such as a multi-punch, is detected during a card reading procedure. If an error exists, the operator must clear the reader and return the cards to the hopper, making certain proper sequence is maintained, and initiate start. The program causes a branch to R1 and the read and error check is again performed.

System Peripheral Input II

System Peripheral Input II is a 1401 program used to generate a system input tape for MCP. System Peripheral Input II differs from Input I as follows:

1. Input II can furnish its own tape label file or have no label file.
2. Data for Input II can be supplied by either a master tape or a card reader.

The program writes the input tape in the same format as System Peripheral Input I. (Figure 143.)

Card Format

Control cards follow the object deck and select the input media (card reader or tape) for the input tape. The control cards are distinguished by an 11-6-8 punch in column 1, which corresponds to a 1401 semicolon. When read by the card reader, the information contained on the control card is printed by the printer. Three types of control cards (BCD punched) are used to control the operation of the input tape program:

1. Label card
2. Master tape
3. Card reader

The formats for the individual cards are:

Label Card: The label card must follow the object deck. The label control card contains the information necessary to write the label on the input tape. Columns 10 through 20 (as defined below) may be written as the first file on the input tape.

Column 2	L - defines a label card.
Columns 10 through 14	A five character label. If the label NOLAB is used, columns 10 through 20 will not be written on the input tape.
Column 15	Program assumes H for high density.
Column 16	Program assumes O for odd parity.
Columns 17 through 20	Additional BCD information to go on the input tape label file.

Master Tape Select Card: The file or files specified by the control card will be read from the master tape and written in 17 card records on the input tape. The output from the master will be buffered until a record (17 cards) is accumulated. Two consecutive End of File marks will identify the end of master tape.

Column 2	T - denotes input from the master tape.
Columns 10 and 11	Denote the first file to be read from the master tape.
Columns 14 and 15	Denote the last file to be read from the master tape. If columns 14 and 15 are blank, only one file from the master tape will be written on the input tape. If columns 14 and 15 contain a file number, the file specified in columns 10 and 11 through the file specified in columns 14 and 15 will be written as one file on the input tape.
Columns 20 through 80	Any additional descriptive information will be contained in these columns. The information will be printed, but not recorded on tape.

Card Reader Select Card: The card deck to be read onto the input tape must immediately follow the card reader select control card. The deck will be read and blocked into 17 card records, and these records written on the input tape. When a new control card is read, an end of file will be written on the input tape.

Column 2	R - denotes input will be from the card reader.

Columns 20 through 80	Any additional descriptive information will be contained in these columns. The information will be printed, but not recorded on tape.

Input Deck

The first cards of the input deck are arranged as follows:

Label card
T or R control card
(Job deck if R control card)

These cards may be followed by control cards or job decks as previously explained to make up the complete input deck. COMD cards may precede job cards in a job deck.

Operator Instructions

1. Ready the master tape on tape unit 2 in the density specified by the outer tape label. The input tape should be readied on tape unit 3 in high density.
2. Ready the printer, reader and console. Set sense switch A and the I-O check switch off.
3. Load the object deck and input deck in the reader.
4. Press the reader Load key.

Programmed Halts

STOP 1 - When the last card is read, the system peripheral input is completed. A second end of file is written on the input tape, the tape is rewound and unloaded. This completes the system input program; there is no recovery other than reloading.

ESTOP 1 - The first control card of a system input must be a label card. If the first card is not a label card, NO LABEL CARD is printed and the program halts. A label card must be made, the cards reloaded with the label card first, and the Start key depressed to start the operation again.

ESTOP 2 - If a control card contains an error, or is not in correct sequence, an error message is printed and the operation halts. The error message is: CNTRL CD ERR - CORRECT AND RESTART. Because all control cards are printed, the last card printed is the incorrect card and will be stacked in the NR stacker. The operator may correct the control card in error, reload the reader with the card in error plus all succeeding cards and press the Start key. If the control card cannot be corrected, the program must be removed.

ESTOP 3 - If five consecutive tape errors occur during the reading of the master tape, the program prints the following message and the operation stops: STOP - BAD MASTER TAPE - STOP. There is no recovery via the program.

ESTOP 4 or ESTOP 5 - If a reader check error occurs during the reading of card files, the message READER ERR - START WITH CD IN NR is printed and the program stops. To recover from the stop condition, the last card in the NR stacker and all succeeding cards must be reloaded in the reader hopper, and the Start key pressed.

ESTOP 6 - If an end of reel condition occurs on the input tape, the message END OF REEL is printed and the program stops. The entire tape must be rewritten, removing the last job processed. The last control card printed will identify the last job processed.

NOTE: It is possible for a non-programmed halt to occur during a program run. The input program assumes that the input tape files are in an ascending order, so that the master tape will only go in the forward direction during normal operation. When two end of file marks are read from the master tape, the master tape will be rewound and unloaded. Any future reference to the master tape will cause machine hangup, because the tape unit is in the not-ready condition.

Program Flow

When the System Peripheral Input II program has been loaded, the input program then checks the first card to determine if it is a label card. See Figure 143. If the first card is not a label card, an error message is printed and the program stops (ESTOP 1). The operator receives printed instructions and can prepare the required label card. The next control card indicates the input medium for the input data. If the card reader is the input selected, the succeeding data cards are sequentially stored in the tape buffer until a 17 card record has been accumulated. The contents of the tape buffer are then written on the input tape, and another record accumulated. If a new control card is read, the input program writes the data accumulated in the buffer followed by an EOF on the input tape and loops to CNTL to decode the control card.

If a master tape is the input selected, the tape is read and passed until it is positioned at the desired file. The files indicated on the control card are read one record at a time up to a maximum of 17 records, and stored in the input tape buffer. When the seventeenth tape record is read, the contents of the input

buffer are written on the input tape, and another 17 records are read and written, continuing until the input specification is satisfied. If the end of the master tape has been reached, as indicated by two end of file marks, it is rewound and unloaded. Any further reference to the master tape causes a machine hangup.

If the input tape reaches its physical end during the writing of a file, the entire tape must be done over, removing the last job processed. An ESTOP 6 error message is indicative of this condition. When the last card is read, the input program ends the input tape by placing a second end of file mark on the tape and then unloads the tape (STOP 1).

System Peripheral Output

The System Peripheral Output program is a 1401 program used to print and punch (unblock) the records from the MCP-generated output tape. The output tape will be read and each record will be printed or punched. When a punched file is completed, a blank card is read from the card read hopper to be used as a file separator. When a file is printed, the printer carriage is restored. Errors are handled internally by the program if the I-O check switch on the console is off.

The punched card output is stacked in stacker 8/2 (middle stacker). To allow the last card of a file to be stacked in stacker 8/2, an extra card is punched. This extra card is stacked in the normal punch (NP) stacker. Any mispunched cards are also stacked in the NP stacker; therefore, all cards in the NP stacker should be ignored and discarded.

Operator Instructions

1. Ready output tape on tape unit 2 in high density.
2. Ready the punch, reader and console; set sense switch A on, I-O check switch off.
3. Load object deck and separator cards in reader hopper.
4. Load blank cards in punch hopper. (Separator cards should be a different color and different corner cut than cards to be punched.)
5. Press Load key on console.

Programmed Halts

For all programmed halts, a message is printed giving procedures for the operator to follow in order to correct errors and resume operation.

STOP 1 - When the output tape reaches its physical end, a trailer record is read which tells whether or not the job is continued on another tape. The messages are: TAPE END - JOB CONT ON NEXT PAGE, or TAPE END.

STOP 2 - If a tape error is detected, the tape is backspaced and the tape record is read again. If the error occurs five successive times, the operation stops and the following error message is printed: BAD TAPE RECORD, IF START IS PUSHED, NEXT TAPE RECORD WILL BE READ. The operator may either continue the operation by depressing the Start key or stop the operation. However, if the Start key is depressed, the error tape record will be ignored.

STOP 3 - If a print error is detected, an error message is printed, the printer carriage is restored and the operation stops. The error message is: PREVIOUS LINE HAD A PRINT CHECK - PUSH START TO PROCESS NEXT LINE.

STOP 4 - If a punch error is detected, a new card is punched. The card containing the error is stacked in the NP stacker. If five successive attempts to punch card correctly have failed, the operation stops and the following error message is printed: PUNCH GOOFED 5 TIMES ON THE SAME CARD - IF START IS PUSHED, NEXT CARD WILL BE PUNCHED. The operator may either continue the operation by pressing the Start key and omitting the error card, or halt the operation to determine the area of malfunction and correct the malfunction.

Program Flow

As the output tape records are read, a tape read error check is performed. See Figure 144. If a tape read error is detected, the tape record is backspaced, the error counter is incremented by one, and the record is read again. If, after five successive attempts at reading the tape record, the error still exists, an error message is printed and the operation stops (STOP 1). With no tape read error detected, the first character of each record is checked to select the output media, either printer or card punch. If C is the first character, the record (ten cards) will be punched. The absence of a C indicates the record (12 lines) will be printed.

With the card punch selected as the output media, the data is punched, error checked, and, if error free, stacked in stacker 8/2. If a punch error is detected, the error card is dropped in stacker NP, the error counter is incremented by one, and the same card is re-punched. Five attempts are made to punch the card correctly. If all five attempts fail, an error message is printed and the operation stops (STOP 4). With the card punch operation error-free, the punch address is incremented by 160 and the next card is punched. This operation continues until the last (tenth) card is punched. After the tenth card is punched, checked, and stacked, the program branches to NEXT and reads the next record.

If the first character of the record is not a C, the printer is readied, a line is printed and then error checked. If an error is detected, an error message is printed and the operation stops (STOP 3). As each line is printed, a line count is maintained. When the twelfth line is printed the routine loops and the next record is read.

When a tape mark is read, a separator card is passed to stacker 8/2 from the read hopper. The punch operation is delayed until the separator card is passed. A second tape mark immediately following indicates the physical end of data, and the trailer record is read to determine if EOJ coincides with the end of tape. A non-zero trailer record specifies EOJ.

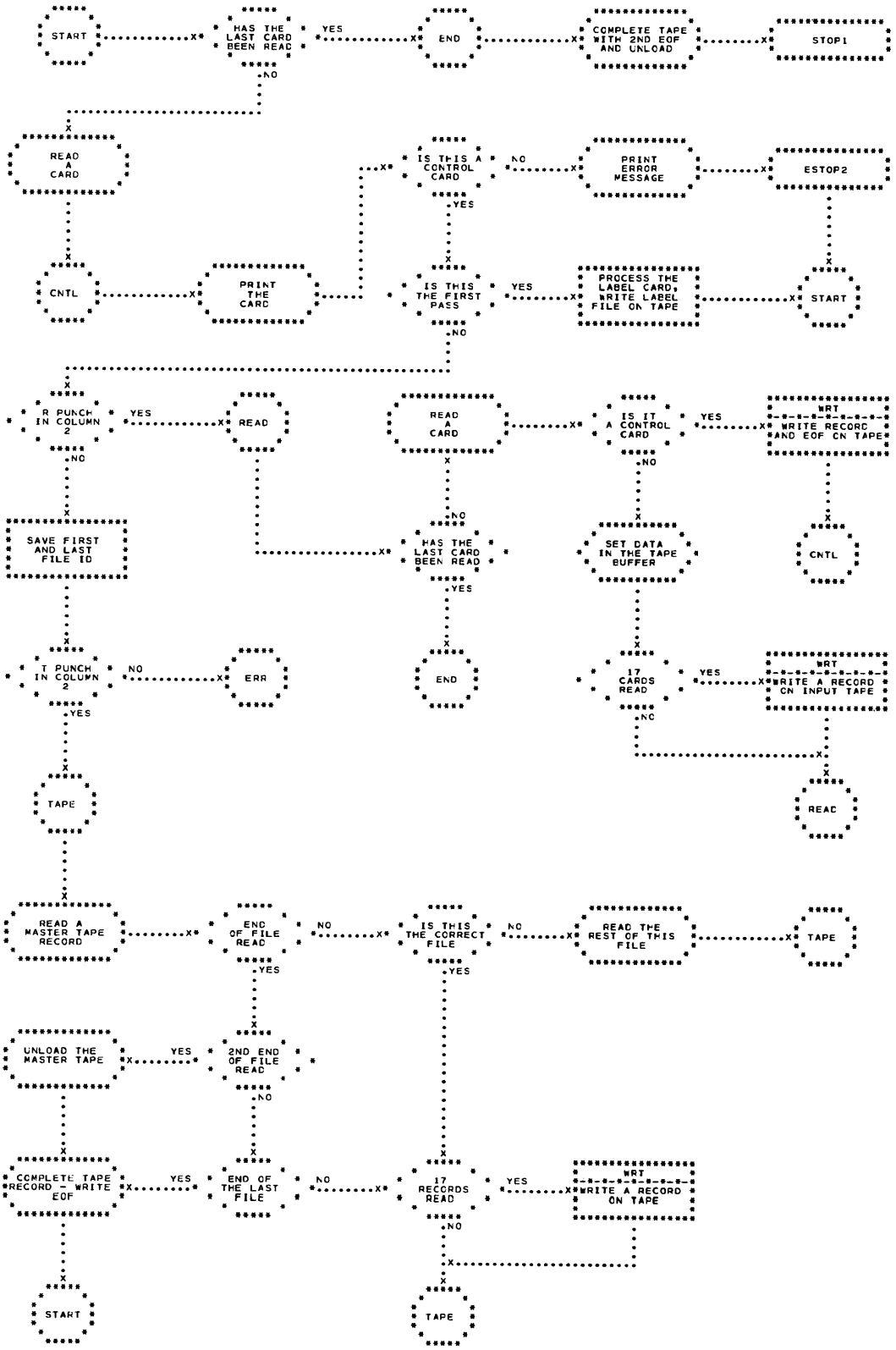


Figure 143. System Peripheral Input II

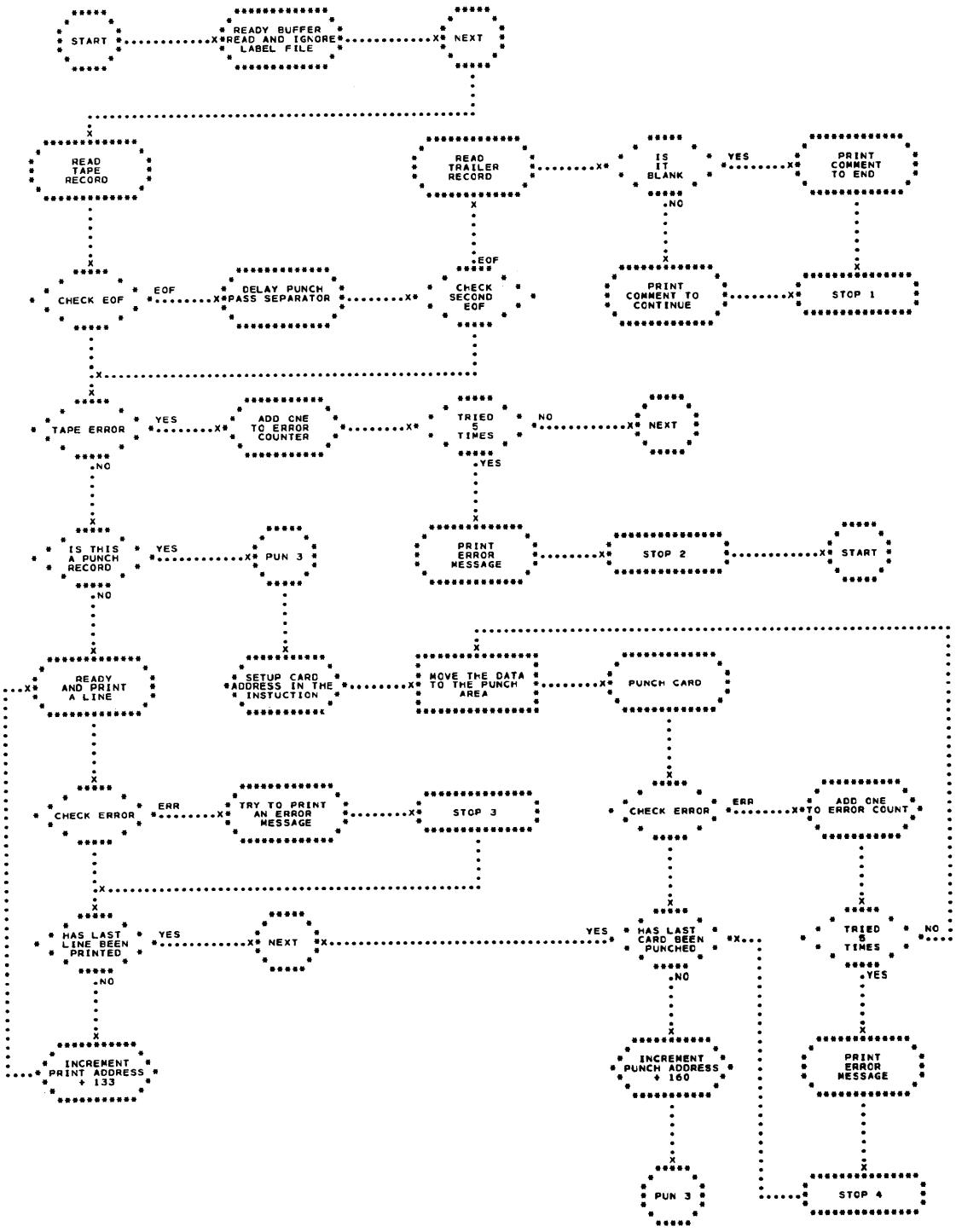


Figure 144. System Peripheral Output

CHANNEL AND UNIT STATUS TABLES

The channel and unit status tables reflect the operational status of all I-O channels and units logically attached to the system. The tables are constructed by the Initializer program at IPL time from IOCD cards defining the I-O configuration. Because MCP initiates all I-O and accepts all I-O interrupts for both MCP and the problem program, it is necessary for MCP to have complete information on the status of all channels and units. The channel and unit status tables provide this information in the form of specific indicators contained in the tables.

The channel and unit status tables will exist in two distinct forms, depending upon whether a channel has a single logical or physical unit (Figure 145), or multi-units (Figure 146). The form for any particular channel is determined by an indicator which is in a constant position for both types of tables. If the channel is a single-unit entity, the channel and unit status indicators are described in a single entry in the channel status tables. If a channel is a multi-unit entry, the channel status table entry will contain information pertaining to the channel, while a separate table, referred to by the channel status table, will contain information about each of the units in distinct entries in the table. An entry in these tables consists of one word.

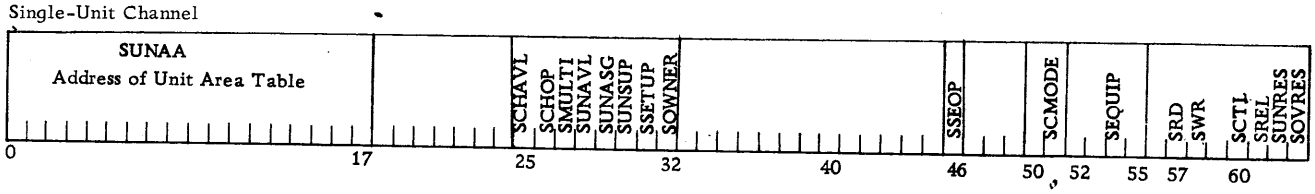


Figure 145. Channel Status Table Entry for Single-Unit Channel

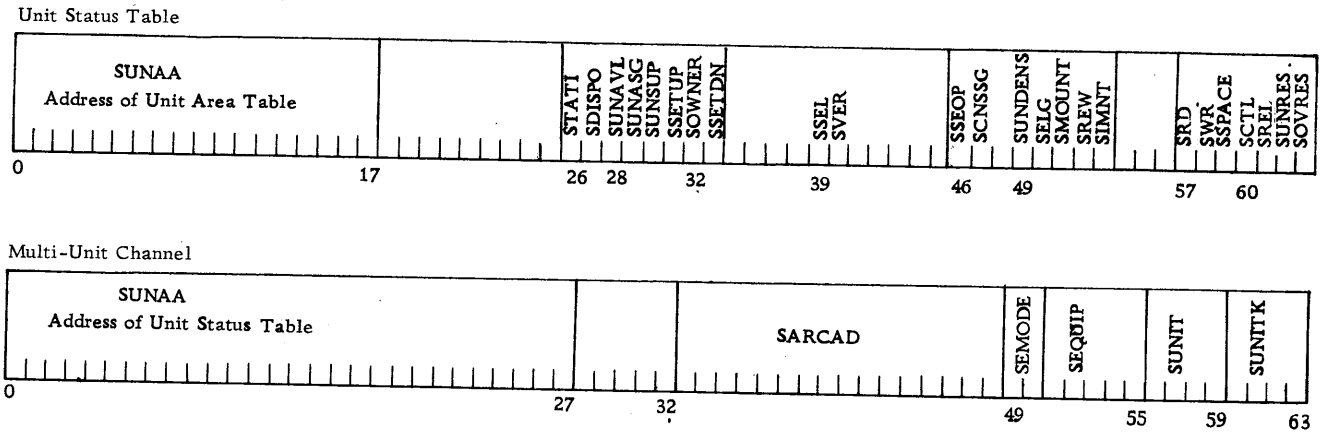


Figure 146. Channel Status and Unit Status Table Entries for Multi-Unit Channels

Channel Status Table Entry - Single Unit Channel

Bits	Symbol	Content
.00 - .17	SUNAA	This field contains the address of the unit area table (UAT) attached to the channel. If the unit has never been assigned, the entry will be zero. This field is set at IPL, and by I-O assignment, the I-O change command, and the special assignment package. The field is referred to by the identifier, the receptor, the I-O change command, the special assignment package, and I-O assignment.
.25	SCHAVL	0 - Channel is available 1 - Channel is not available A channel is available when it is physically attached to the exchange and is capable of being actuated by MCP. This indicator exists in the channel status table of either a single or multi-unit channel. It is set at IPL, and sometimes by the I-O change command. It is tested by the identifier (as an internal MCP check upon actuation), by special assignment, I-O assignment and I-O change.
.26	SCHOP	0 - Channel not operating 1 - Channel operating This indicator exists in the channel status table of either a single or multi-unit channel. It is set by the actuator routines and reset by the receptor. It is tested by the identifier, I-O change command and special assignment, Job Control **4** and UNASSIGN and the $FREE routine.
.27	SMULTI	0 - Single-unit channel 1 - Multi-unit channel This indicator exists in the channel status table of either a single or multi-unit channel. It is set at IPL. It is tested by most actuator routines, I-O change command, special assignment, UNASSIGN, I-O assignment and receptor.
.28	SUNAVL	0 - Unit is available 1 - Unit is not available A unit is available when it is physically attached to a channel and is capable of being actuated by MCP. This indicator exists in the channel status table for single unit channels and in the unit status table for multi-unit channels. It is set at IPL and sometimes by the I-O change command. It is tested by the identifier, I-O change, special assignment and I-O assignment.

Bits	Symbol	Content
.29	SUNASG	0 - Unit is assigned 1 - Unit is not assigned A unit is assigned if it is logically connected at present to either MCP or the currently operating problem program. This indicator exists in the channel status table of single unit channels, and in the unit status table of multi-unit channels. It is set by I-O assignment and sometimes by special assignment and the I-O change command. It is tested by the identifier, I-O assignment, I-O change command, special assignment and some actuator routines.
.30	SUNSUP	0 - Unit is not suppressed 1 - Unit is suppressed This indicator exists in the channel status table of single unit channels, and in the unit status tables of multi-unit channels. The unit suppressed indicator is set to 1 when an I-O interrupt has not been taken for a specific unit. It is set by the receptor, and tested by the identifier and the I-O change command.
.31	SSETUP	0 - Not Setup operation 1 - Setup operation This indicator exists in the channel status tables of single unit channels, and in the unit status tables of multi-unit channels. The setup indicator is turned on by the actuator whenever control is to be returned to the actuator for further processing of an I-O request. It is set by the actuator, and tested by the receptor and I-O change.
.32	SOWNER	0 - Problem program 1 - MCP This indicator exists in the channel status tables of a single unit channel, and in the unit status table of a multi-unit channel and shows what level of program is the current owner of a particular unit. It is set by I-O assignment, I-O change command, and special assignment, and is tested by the receptor, identifier, I-O change command and special assignment.
.46	SSEOP	0 - Allow EOP 1 - Suppress EOP This indicator exists in the channel status table of a single unit channel, and in the unit status table of a multi-unit channel and shows that the I-O operation has been requested in the SEOP mode. If the operation is terminated with an EOP, no indication will be given to

Bits	Symbol	Content
		the user. It is set by the identifier and tested by the receptor.
.50 - .51	SCMODE	00 - Illegal 01 - ECC, ODD parity 10 - EVEN parity 11 - NOECC, ODD parity This indicator exists in the channel status table of either a single or multi-unit channel. It is set initially at IPL to illegal (00), and subsequently by the actuator.
.52 - .55	SEQUIP	0 - Illegal 1 - Disk 2 - Console 3 - Card Reader 4 - Punch 5 - Printer 6 - None 7 - None 10 - Tape 11-17 - None This field exists in the channel status tables of either a single or multi-unit channel. It is set by the initializing program and it may be changed by the I-O change command for single-unit channels.
.57	SRD	0 - Non-read operation 1 - Read operation This indicator exists in the channel status tables of a single unit channel, and the unit status table of a multi-unit channel. The indicator is set by the actuator whenever a read operation, which is not a setup operation, is in progress. It is reset by the receptor. It may also be cleared by the I-O change command.
.58	SWR	0 - Non-write operation 1 - Write operation This indicator exists in the channel status table of a single unit channel, and the unit status table of a multi-unit channel. It is set by the actuator whenever a write operation, other than a setup operation, is in progress and is reset by the receptor. It may also be cleared by the I-O change command.
.60	SCTL	0 - Non-control operation 1 - Control operation This indicator is found in the channel status table of a single unit channel or the unit status table of a multi-unit channel. It is set by the actuator whenever a control operation other than a setup operation is initiated and is tested and reset by the receptor.

Bits	Symbol	Content
.61	SREL	0 - Non-release operation 1 - Release operation This indicator exists in the channel status table for a single unit channel and in the unit status table for a multi-unit channel. The release indicator is set by the actuator whenever a release pseudo-operation is requested. The indicator is tested by the receptor.
.62	SUNRES	0 - Not reserved 1 - Reserved This indicator exists in the channel status table of a single unit channel and in the unit status table of a multi-unit channel. It is a second level indicator for special situations in I-O assignment reservation of units. It has priority over SOVRES. It is set and tested by the I-O assignment program.
.63	SOVRES	0 - Not reserved 1 - Reserved This indicator exists in the channel status table of a single unit channel and in the unit status table of a multi-unit channel. It is used by I-O assignment to ascertain those units which have been preassigned in the overlapped mode of operation and is set and tested by I-O assignment.

Channel Status Table Entry - Multi-Unit Channel

Bits	Symbol	Content
.00 - .27	SUNNA	This field contains the address of the first entry in the unit status table for the channel.
.28 - .31	Not Used	
.32 - .49	SARCAD	This field exists only in the channel status table of a multi-unit channel. It contains the arc address of the last arc to be located on the disk to which the channel status table entry refers. It is set by the actuator whenever a locate or read or write of the disk is completed.
.50 - .55	Same as single unit channel.	
.56 - .58	SUNIT	0 - 7 indicates which logical or physical unit is presently selected for the specific channel to which the channel status entry refers. This field exists only in a multi-unit channel status table entry. It is set by the actuator locate routine.

Bits	Symbol	Content
.59 - .62	SUNITK	$0 - 17_8$ indicates the number of logical or physical unit status table entries attached to the channel to which the channel status entry refers. This field exists only in a multi-unit channel status table entry. It is set by the initializing program.
.63	Not Used	

Unit Status Table Entry - Multi-Unit Channel

Bits	Symbol	Content
.00 - .17	SUNAA	This field contains the address of the unit area table (UAT) attached to it. If the unit has never been assigned, the entry will be blank. The field is controlled at IPL, and by I-O assignment, the I-O change command, and the special assignment package. It is referred to by the I-O identifier, the receptor, the I-O change command, the special assignment package and I-O assignment.
.26	STATI	0 - Scratch tape 1 - Specific tape This indicator is found in the unit status table of a multi-unit channel. If a tape channel, this indicator shows whether a scratch tape is mounted or a specifically designated tape is mounted for the given unit. This indicator is set and tested by I-O assignment. It may be reset by I-O change.
.27	SDISPO	0 - Tape mounted 1 - No tape mounted This indicator exists in the unit status table of a multi-unit channel. It indicates whether a tape reel is mounted on a particular unit, or not mounted. It is set and tested by I-O assignment, UNLOAD, FREE, and IOCHANGE command.
.28 - .32	Same as in Single Unit Channel Status.	
.33	SSETDN	0 - Density never set 1 - Density set This indicator forces the density of a tape unit to be set the first time the unit is referenced after IPL.
.39	SSEL	0 - Selected 1 - Not selected This indicator exists in the unit status table of a multi-unit channel. It indicates which unit is presently located

Bits	Symbol	Content
		or selected to the channel. The select indicator is set originally by the initializing program and later by the actuator routines.
.40	SVER	0 - Verified 1 - Not verified This indicator is found in the unit status table of a multi-unit channel. It indicates whether a tape label has been verified on a particular unit or not. The verify indicator is turned on by I-O assignment and sometimes by the I-O change command and special assignment. It is tested and turned off by the actuator.
.46	SSEOP	0 - Allow EOP 1 - Suppress EOP This indicator exists in the channel status table of a single unit channel, and in the unit status table of a multi-unit channel. It shows that the I-O operation has been requested in the SEOP mode. If the operation is terminated with an EOP, no indication will be given to the user. The indicator is set by the identifier and tested by the receptor.
.47	SCNSSG	0 - No console signal 1 - Console signal This indicator exists in the unit status table of a multi-unit channel. It is set by the conceptor and is tested by the actuator READ routine.
.49	SUNDENS	0 - High 1 - Low This indicator exists in the unit status table of a multi-unit channel. It reflects the current density setting of that unit, if applicable. It is set by the initializing program to high density, and altered thereafter by the actuator. It is also tested by the actuator.
.50	SELG	0 - No erase long gap operation 1 - Erase long gap operation This indicator exists in the unit status table of a multi-unit channel. It indicates whether or not an erase long gap operation, which is not a setup operation, has been requested by an $ELG pseudo-op. This indicator is set by the actuator, and tested and reset by the actuator ($Write Routine Only).

Bits	Symbol	Content
.51	SMOUNT	0 - Not awaiting tape mounting 1 - Awaiting tape mounting signal This indicator exists in the unit status table of a multi-unit channel. It is set by the actuator, I-O assignment, I-O change command, and special assignment. It indicates that a channel signal can be expected from the reading of a tape unit after mounting of a reel. The indicator is tested and reset by the receptor, I-O change command, and the actuator.
.52	SREW	0 - Not rewind tape operation 1 - Rewind tape operation This indicator exists in the unit status table of a multi-unit channel. The indicator is set to 1 by the actuator whenever a rewind operation, other than a setup operation, is in progress, and is tested and reset by the receptor.
.53	SIMNT	0 - Not initial mount status 1 - Initial mount status This indicator is found in the unit status tables of a multi-unit channel. The indicator shows that the first mounting request has been made for program tape I-O request, and the channel signal has not been received indicating that the tape reel has been mounted. The indicator is set by the I-O assignment package, and sometimes by the I-O change and special assignment programs. The indicator is tested and reset by the receptor.
.57 - .58	Same as Single Channel Status.	
.59	SSPACE	0 - Non-space operation 1 - Space operation This indicator is found in the unit status table of a multi-unit channel. It indicates that a spacing operation, other than a setup operation, is in progress on a tape unit. The indicator is set by the actuator and tested and reset by the receptor.
.60 - .63	Same as Single Channel Status.	

THE UNIT AREA TABLE

The unit area tables contain information about assigned physical units. I-O assignment builds the tables and then the actuator and receptor store in and use significant information from the tables during the execution of I-O operations. There is one unit area table for each existing unit; its address can be obtained from either the I-O location table or the channel or unit status table to which it is attached. A unit area table consists of nine words (Figure 147), and its format is as follows:

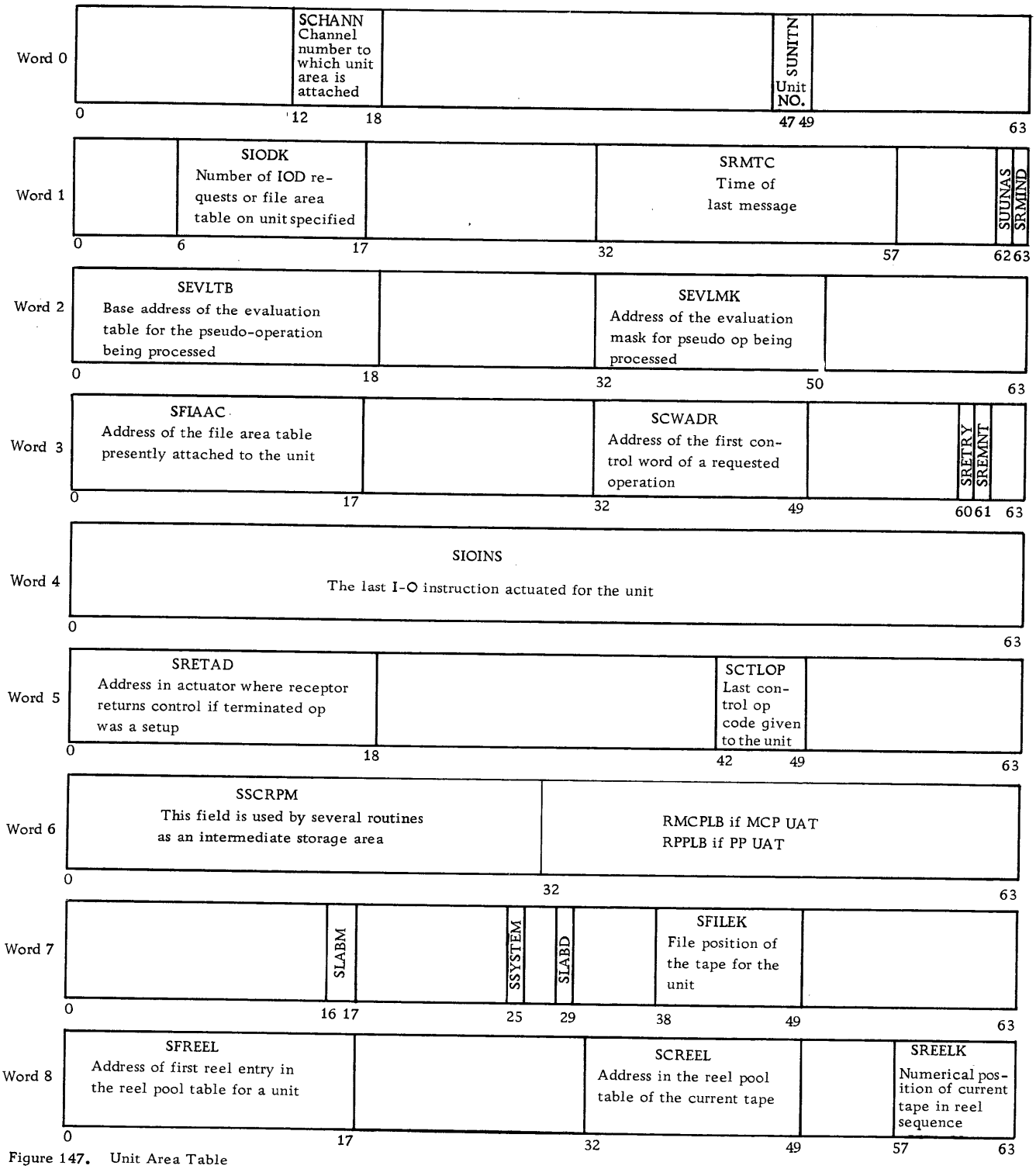


Figure 147. Unit Area Table

Bits	Symbol	Content
0.12 - 0.18	SCHANN	This field contains the binary number, located at a half word address, representing the channel number to which the unit area is attached. The field is set by I-O assignment and sometimes by I-O change and special assignment. It is used by the actuator, receptor and I-O change.
0.47 - 0.49	SUNITN	This field contains the binary number, located at a full word address, representing the unit number to which this unit area is attached. Setting and testing of this field is the same as for SCHANN.
1.06 - 1.17	SIODK	This field specifies the number of IOD requests or file area tables attached to the unit specified, as a binary number, located at a full word address. The field is set by I-O assignment.
1.32 - 1.57	SRMTC	Time of the last mount message.
1.62	SUUNAS	Unassign indicator 1 = already unassigned.
1.63	SRMIND	Remind bit: 0 - Mount reminder message not sent. 1 - Mount reminder message sent.
2.00 - 2.18	SEVLTB	This field specifies the base address of the evaluation table for the pseudo-operation currently being processed for the unit.
2.32 - 2.50	SEVLMK	This field contains the address of the evaluation mask for the pseudo-operation currently being processed for the unit. It is set and used by the actuator.
3.00 - 3.17	SFIAAC	This field specifies the address of the file area table presently attached to the unit. It is set and used by the actuator.
3.32 - 3.49	SCWADR	This field contains the address of the first control word of a requested operation. It is set by the control word checking routine in the actuator, and is used by the actuator.
3.60	SRETRY	I-O instruction retry control bit. If 1, retry.
3.61	SREMNT	This indicator is set by the tape label verification routine. It indicates to the verification routine whether or not a problem program tape has failed to verify and the operator has been informed. 0 - New tape not requested 1 - New tape requested

Bits	Symbol	Content
4.00 - 4.63	SIOINS	This field contains the last I-O instruction actuated for the unit. It is set and used by the actuator.
5.00 - 5.18	SRETAD	This field contains the address in the actuator to which the receptor will return control if the terminated operation was a setup operation. The field is set by the actuator and used by the receptor.
5.42 - 5.49	SCTLOP	This field contains the control operation code, in binary, for the last control operation given to the unit. It is set and used by the actuator.
6.00 - 6.31	SSCRPM	This field is used by several actuator routines as an intermediate storage media assigned expressly to the unit being acted upon.
6.32 - 6.63	RMCPLB (if MCP UAT) RPPLB (if PP UAT)	
7.16 - 7.17	SLABM	This field indicates the mode specified in the label of a system tape after said tape has been verified. The field is set by the verify routine and used by the actuator. 00 - Illegal 01 - ECC - ODD 10 - EVEN 11 - ODD - NOECC
7.25	SSYSTM	This indicator tells whether the tape presently mounted on the unit is a labelled tape, or not. It is set by I-O assignment and altered by the receptor when new tapes are mounted. It is tested by the receptor and by the verify routine. 0 - Non-labeled 1 - Labeled
7.29	SLABD	This indicator specifies the density found in the label after verification. It is set, tested, and used by the actuator. 0 - High 1 - Low
7.38 - 7.49	SFILEK	This field specifies the file position of the tape for the unit. If the tape is labelled, file zero is the label file. The file count will be -0 if before the label record and +0 if after the label record but before the label file mark. The field is set by I-O assignment and tested and altered by the actuator and receptor.
8.00 - 8.17	SFREEL	This field contains the address of the first reel entry in the reel pool table for a tape unit. It is set by I-O assignment and used and modified by the actuator and receptor.

Bits	Symbol	Content
8.32 - 8.49	SCREEL	This field contains the address in the reel pool table of the currently mounted tape. It is set by I-O assignment and altered by the actuator.
8.57 - 8.63	SREELK	This field indicates the numerical position of the currently mounted tape reel in the sequence of reels for the tape unit. It is set by I-O assignment and altered by the receptor.

THE FILE AREA TABLE

A file area table exists for each IOD card, containing the information from the IOD card, and also containing information obtained from an interrupt received for the specific file. The file area tables are constructed by the Move routine in I-O assignment just before the Go phase of a program. File area tables exist for the MCP IOD cards. A file area table consists of seven words (Figure 148), and its format is as follows:

Bits	Symbol	Content
0.00 - 0.17	SIODRN	This field contains the IOD reference number, in binary, for the file area table. It is set by I-O assignment, and used by the receptor.
0.32 - 0.50	SACTAD	This field contains the address of the calling sequence which caused the last actuation on this file. The field is set by the identifier and transmitted to the table of exits when an interrupt is unstacked.
1.09 - 1.13	SIOIND	This field contains the I-O indicators after the receptor has processed an interrupt. It is set by the receptor and transmitted to the table of exits when an interrupt is unstacked.
1.32 - 1.50	SINTAD	This field contains the address in the program at which interruption occurred for the I-O interrupt being unstacked. It is set by the receptor.
1.57	SEWAIL	This indicator enables the $WAIT routine to not wait again on a file which has been reactuated in a fixup. It is set and tested and reset by the $WAIT routine.
2.00 - 2.17	STOELO	This field contains the address of the table of exits presently attached to this file. It is set by I-O assignment, but may be altered by the $CHEX routine.
2.27 - 2.28	SFDEN	The file density is that density specified on the IOD card. It is set in the file area table by the I-O assignment routine and used by the actuator. 00 - Null 01 - High 10 - Illegal 11 - Low

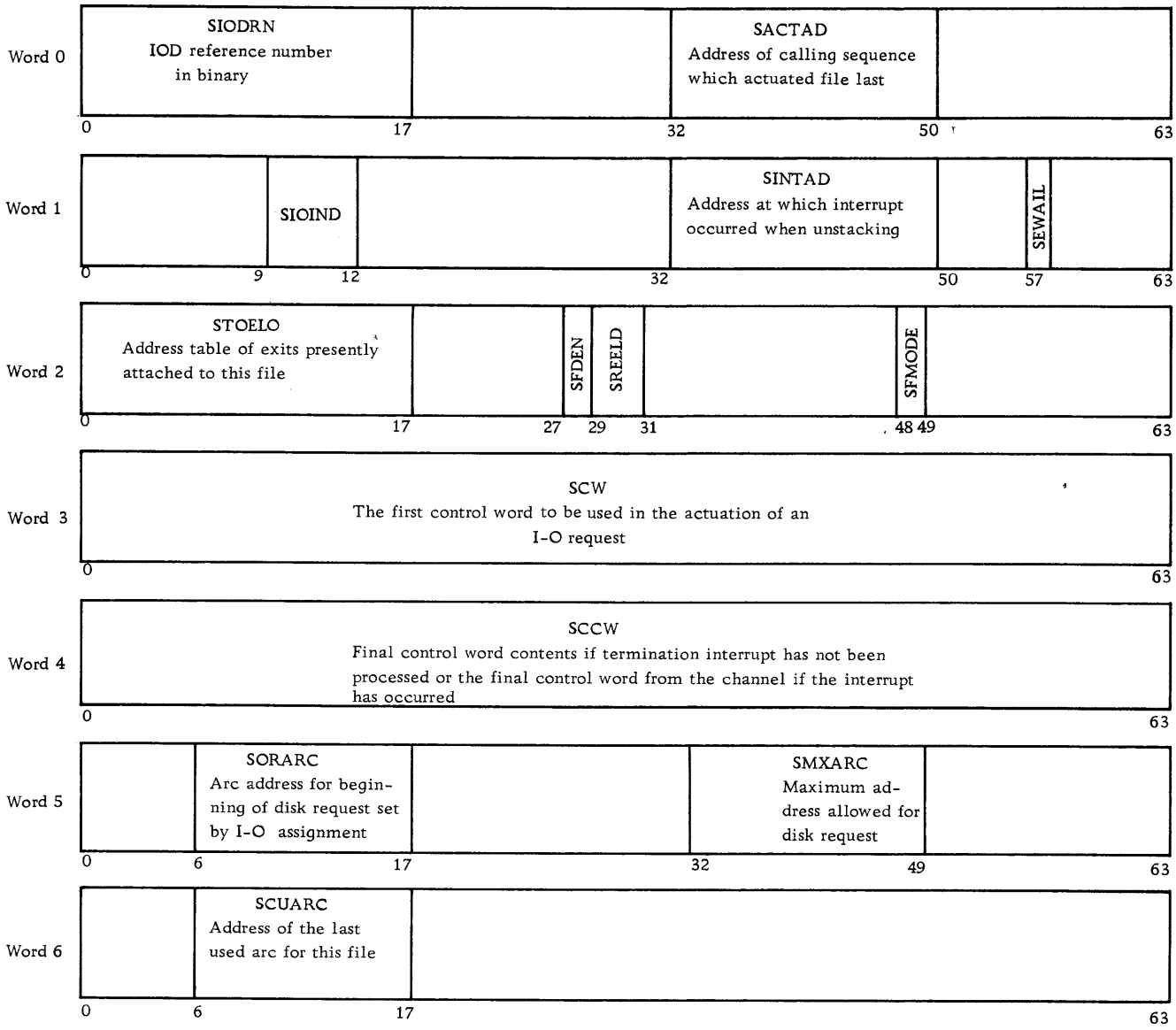


Figure 148. File Area Table

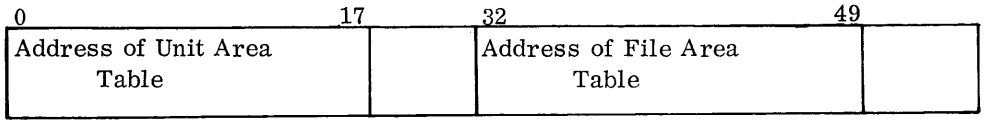
Bits	Symbol	Content
2.29 - 2.31	SREELD	This field contains the code for disposition of tape reels at $FREE time or at $EOJ time. It is set by the I-O assignment routine, and tested by $FREE. 000 - Do not save NSAVE 001 - Save if job complete CSAVE 010 - Save if job incomplete ISAVE 011 - Save in all cases SAVE
2.48 - 2.49	SFMODE	This field contains the mode specified on the IOD card. It is set by the I-O assignment routine and tested by the actuator.
3.00 - 3.63	SCW	This field contains the first control word to be used in the actuation of an I-O request. It is set by the control word checking routine and is used by the actuator.
4.00 - 4.63	SCCW	This field contains the first control word used in the actuation of an I-O request if the interrupt for the termination of that request has not been processed, or it contains the final control word contents if the interrupt has been processed. The field is set by the control word check routine or by the receptor. It is used by the $CCW routine.
5.00 - 5.17	SORARC	This field specifies the arc address for the beginning of the disk request setup by I-O assignment. It is tested by the actuator.
5.32 - 5.49	SMXARC	This field specifies the maximum allowable arc address for the disk request, set up by I-O assignment. It is tested by the actuator.
6.00 - 6.17	SCUARC	This field specifies the last used arc for this file. It is set by the locate routine or the receptor, and is tested by the actuator.

Bits not specified are not used. Symbols are defined relative to the location of the table e.g., SIODRN SYN(BU, 18), 0.0 .

THE SYMBOLIC I-O LOCATION TABLE

There are two symbolic I-O location tables: one for MCP, and one for PP. The MCP symbolic I-O location table is constructed in MCP core at IPL time by the initialization program from the MCP IOD cards. The PP symbolic I-O location table is constructed at job control time from the PP IOD cards.

The tables are identical in format and usage. There is one entry for each IOD reference number (IODRN) specified on the IOD cards, and the entry position is the same as the IODRN. The address of the unit area table assigned to the IODRN is entered in the VF of the first half word, and the address of the file area table assigned to the IODRN in the VF of the second half word.

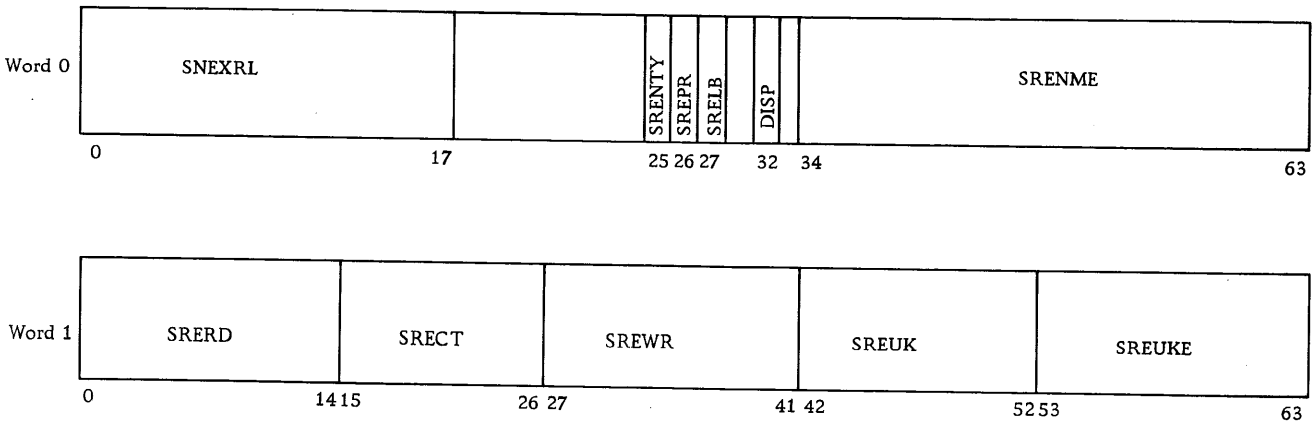


Format of Symbolic I-O Location Table

The base addresses of the two I-O location tables are contained in the value fields of the half words at SBAPP and SBAMCP, for PP and MCP respectively.

THE REEL POOL TABLE

The reel pool table consists of one full word entry for each reel which is requested by the problem program. The table is set up at job control. Its format is as follows:



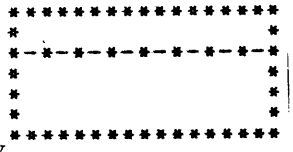
Bits	Symbol	Content
.00 - .17	SNEXRL	Address of the next reel table entry for the same unit. It is zero if no more reels are to be mounted.
.25	SRENTY	Entry in table indicator. 0 - No entry 1 - Entry
.26	SREPR	Reel protected indicator. 0 - not protected 1 - protected
.27	SRELB	Reel labeled indicator. 0 - not labeled 1 - labeled
0.32		Reel disposition indicator. 0 - not saved 1 - saved
.34 - .63	SRENME	Reel name in A6 code.
1.00 - 1.14	SRERD	Count of the read instructions issued to this reel.
1.15 - 1.26	SRECT	Count of the control instructions issued to this reel.
1.27 - 1.41	SREWR	Count of the write instructions issued to this reel.
1.42 - 1.52	SREUK	Count of the unit errors of this reel.
1.53 - 1.63	SREUKE	Count of the data errors of this reel.

The seven functional symbols used to represent logical groupings of instructions on the flow charts are as follows:

Processing

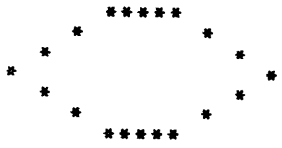
A brief statement of an operation or function is in the block. A processing block may have multiple entries but only one exit.

Subroutine



The primary function of this block is to represent a complex logical program unit without undue detail. The name of the program symbol, taken directly from the code, is printed in the upper section of the block, above the broken line. A brief description of the usage of the subroutine is printed in the lower section. A subroutine block may have multiple entries and multiple exits. The exits are labeled when necessary.

Decision



Brief statements of comparison, decision, choice or tests are printed in this block in the form of a question. A decision block may have multiple entries and has multiple exits. The exits are always labeled to show the paths of flow available for the various conditions labeled.

Modification

A brief statement of internal modification to program instructions is printed in this block. A modification block may have multiple entries but has only one exit.

Input/Output (I-O)

A brief statement of I-O function is printed in this block. An I-O block may have multiple entries and multiple exits. The exits are labeled when necessary.

Logical Connectors

Connectors are represented by a circle. All connectors used in the MCP flow charts have logical significance in terms of program flow. The names used are program symbols as found in branch addresses in the code.

A connector may be an entry connector, an exit connector, or both. Any program symbol can be used at most once in an entry connector. An entry connector has only one exit, and may or may not have entries. The symbol used in an entry connector is the symbol used to start the corresponding section of code.

An exit connector may have multiple entries. If it has an exit, it is also an entry connector and the symbol used will not be used elsewhere as an entry connector. The symbol used in an exit connector may be used in other exit connectors.

When a phrase such as ENTER ROUTINE, represented in the code by BR, 0.0($XR) is used in an exit connector, the actual branch point is computed (for instance, a branch table). When $RET appears as an exit connector, the pseudo-op is being used to determine subsequent flow. (See the Return Routine.)

Halt

A brief statement of the condition of the halt or an appropriate exit is printed in this block. A halt block may have multiple entries. A halt box with an exit implies external intervention (interrupt or operator) to utilize the exit.

The general flow on the charts proceeds to the right and/or down. The functional symbols are logically linked by the use of flow lines, represented by dotted lines terminating in arrowheads (printed as X) showing the direction of the flow. Arrowheads are always placed at the point of entry to a functional symbol and at junctions with major flow lines.

On some of the charts a program symbol from the code is printed to the left and above the functional symbol to help the reader relate the flow chart to the code. Sometimes a line of text is printed in the vicinity of a block as an aide to the reader. All symbols referred to in any of the blocks are taken directly from the program.

K2 and subsequent machines contain in the E-box a transistor register used as a floating point multiplier register ($MR). This register is not available on X-1 (LASL), the 7950, or K-1 (LRL).

The fraction and sign bits of $MR occupy the last 52 bits of the D register in the arithmetic unit. Its exponent occupies $R.48 through $R.59. The multiplier register has no address and can be referred to only through floating point instructions LMR, STM, and *+. The instruction LMR loads $MR in the same way as LFT loads $FT on the K-1 machine; similarly, *+ behaves in the same way as the corresponding instruction in K-1, but $MR instead of $FT will furnish the implied operand. Bit configurations for LMR and LFT are the same (10010), as are bit configurations for *+ and its corresponding instruction (01110). The floating point instruction Store Multiplier (STM) with bit configuration 11110, stores the contents of $MR into the full word specified by the effective address, and is added to K-2 and subsequent machines. On these machines LMR, *+, and STM are capable of being modified by modifier bits, as are other floating point instructions. These new instructions occupy only one level of the lookahead rather than two, as do the original corresponding instructions.

VFL instructions are unaltered, but $MR, aside from its shared $R bits, is part of the VFL accumulator and is on the path of data flow for some VFL instructions. Thus:

1. VFL fetch type instructions with offset less than 16 will alter the exponent bits of $MR. The corresponding store type instructions may store $MR exponent bits.
2. VFL load (L) and load with flag (LWF) automatically clears the $MR exponent to all zeros before the loading operation.
3. The fraction and sign byte parts of $MR are replaced whenever a VFL instruction calling for word boundary crossover is executed, because the D register is needed in all such cases to house the second full word temporarily.
4. Due to bit-sharing with $R, the $MR exponent will be altered by a VFL instruction which changes the sign of the accumulator.

MCP saves and restores $MR along with other low registers. It takes the precaution of resetting $OP in the event the machine has no multiplier register. Note that by using the $MR option, arithmetic time in matrix multiplication can be cut in half as compared with the corresponding $FT operation.

APPENDIX D - CONNECTOR INDEX

PART 1-MCP

Printed in U.S.A. C22-6728-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York