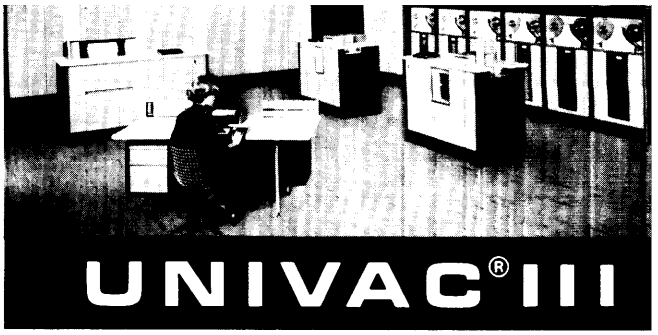


UNIVAC® III

GENERAL MANUAL

EXEC 8

OBTAIN JOB LOADER RERUN ALLOCATOR SUCO SYNCHRONIZER CONTINGENCY CONTROL

PROGRAMMER'S GUIDE

UP-4144

U3521

This manual is published by the UNIVAC Division in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC Systems developments. The UNIVAC Division will issue updating packages, utilizing primarily a page for page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The UNIVAC Division reserves the right to make such additions, corrections, and/or deletions as, in the judgment of the UNIVAC Division, are required by the development of its respective Systems.

PRINTED IN U.S.A.

UNIVAC III January 21, 1963

BOSS, U-3521

UPDATING PACKAGE A

CONTENTS:

The attached sheets are additions and changes to the subject manual.

Two new sections are included:

1. SECTION III OPERATIONAL CONTROL
2. SECTION VI PROCESSOR ERROR CONTROL

The index is modified to bring it up to date. All pages replace the existing pages. Pages that are removed should be destroyed.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

INTRODUCTION

The Business Oriented Systems Supervisor for the UNIVAC III (BOSS III) is a modular executive system of great flexibility. It is designed in sections so that a system may be "pieced" together to create the executive routine desired for any particular configuration. If required, the executive routine can be changed on a day-to-day basis. Only those sections absolutely necessary are maintained in memory while programs are operating.

Under the direction of BOSS III, programs are called from a Systems Tape, loaded into the memory of the computer, executed and completed. Concurrent operations are made possible by BOSS III.

The control of all type-ins and type-outs, other contingencies, processor errors, memory dumps and rerun procedures is included in BOSS III. Certain areas are not included in this first release but will appear in later additions.

Some of the specifications presented are interim and may be modified later, however, any programs coded using these specifications will be operable under the completed system. The interim procedures presented in this manual have been tested and are operable.

GENERAL

The Business Oriented Systems Supervisor for the UNIVAC III (BOSS III) is composed of sets of subroutines from which various combinations may be selected to form a particular BOSS III for a particular configuration. Within an executive system for a computer of such power as the UNIVAC III certain basic functions must be provided. These functions may be grouped into five major headings:

1. Operational Control
2. Synchronizer Control
3. Contingency Control
4. Processor Error Control
5. Memory Dump

These will be briefly and generally discussed in this section and expanded upon in detail in the sections that follow.

1. Operational Control

   Operational control provides the necessary routines to locate, load and execute programs either from a previously scheduled sequence of programs or through operator intervention. It is also concerned with the allocation of memory and input-output facilities for all programs and the initiation of programs when the program has been allocated and loaded. The function of removing a run and releasing input-output facilities when a program is ended for any reason is also included in operational control.

   Most of the subroutines which are a part of operational control are found in the Supervisor which is maintained on tape and only brought into the memory for use when a program is ended or a new program is called for, and for other reasons specified later.

2. Synchronizer Control

   Concurrent processing involves multi-program usage of input-output facilities as well as the central processor. Subroutines to accomplish

this sharing are contained in the synchronizer control routine. The tape synchronizer control routines are in memory at all times; other input-output synchronizers are loaded as required.

I/O orders are submitted to the executive system which determines the order of execution of all orders. Tape orders are executed according to a priority system. Since the main purpose is to maintain rated speeds on the slower peripheral units (card readers, punches, printers) such symbiont (peripheral) runs using these peripheral units are given to synchronizer control which attempts to fill the stand-by location and then returns to the interrupted program. Under certain conditions control is not immediately returned to the interrupted program. When the input-output order causing the interrupt was an order from a symbiont run to a printer, punch or reader, control may or may not be immediately returned to the interrupted program. If that symbiont program has previously given up control of the computer because the program was waiting for the completion of an order before it could proceed, then control is passed to that symbiont run and the interrupted program is held in abeyance until the symbiont program voluntarily gives up control.

In this manner, peripherals are maintained at rated speeds and the available computer time is shared by the programs running concurrently.

3. Contingency Control

Contingency control is in charge of processing operator-computer communications. It accepts and executes all message type-outs from programs in the computer whether recording running operational information or requesting operator intervention. It allows, through unrequested type-ins, operator intervention to change the operational state of the computer. All information typed into or out of the computer will appear on the console printer.

Contingency control also handles errors arising from arithmetic overflow, from the use of invalid operation codes and from clock power failure. (See SECTION V for specifications.)

Contingency control is part of the permanent executive system in memory at all times.

4. Processor Error Control

In a running environment, this function must be present at all times. It recognizes all processor errors and types out a diagnostic message.

5. Memory Dump

BOSS III contains a group of subroutines for memory dumps to be used in establishing rerun points and for memory dumps for informational purposes (useful in finding programming errors).

A repositioner routine is also included in the Supervisor to initiate a program from a rerun point.

Memory dump for rerun purposes exists for main programs only (tape-to-tape runs). Symbiont programs contain recovery procedures rather than rerun points.

Information necessary for BOSS III to maintain its internal files and tables is provided by the assembler or compiler that produces the object program. These communications and parameters are provided in the appropriate manual.

UNIVAC III COBOL PROGRAMMER'S GUIDE	U-3389
UNIVAC III FORTRAN PROGRAMMER'S GUIDE	U-3517
UNIVAC III UTMOST PROGRAMMER'S GUIDE	U-3520

III. OPERATIONAL CONTROL

A. General

Operational control is provided from a single execution system tape which will contain supervisory control routines and diagnostic aids in addition to the absolute programs which may be loaded by the Inter-Run Assignment Routine, SUCO, for object time execution. Control information made up from control cards during the Designation Run, DECO, is included with each absolute program. This control information is interpreted by SUCO to determine tape assignment, storage allocation, etc. The primary controlling medium during execution will be through the typewriter; the contingency control routine forms a substantial part of the controlling routines in core during execution. In addition to the typewriter routine, the basic input-output communication routine and the basic tape input-output routines will be in core, plus the necessary tape assignment and boot-strapping for the termination of a job. All coordination which can be accomplished between jobs has been placed in the supervisor. Upon the completion of a symbiont, the principal program will be dumped temporarily on tape, the supervisor loaded, and after completion of the supervisor action, the principal program will be reloaded and continued.

Progression of the principal programs from one program to another will be established at set-up time and carried in control information through the designation run. In addition, operator intervention will permit alteration of this schedule. Scheduling of symbionts will be under complete operator control.

B. BOOT

Bootstrap and System Tape Loader

1. General

BOOT provides a system tape bootstrap and a loader to load specified routines from the system tape.

2. Bootstrapping

BOOT is loaded from a system tape by hitting the load key when the computer is cleared and the system tape is rewound. This routine includes the system tape search routine. If the run key is struck BOOT proceeds to load the routine EXEC using the system search routine.

3. System Search

The system search is accomplished by scanning the system tape forward for a symbol block corresponding to the specified symbol. If an end-of-file is encountered before the symbol, the tape is rewound and search continued through the file a second time until either the program or the end-of-file is encountered. If the symbol is found, the corresponding program is loaded and executed if so indicated. If not, an error is indicated.

4. Register Usage

BOOT uses the AR's and index register 2 only.

5. To Load EXEC from Tape

Depress the following keys in sequence:

(a) REWIND

(b) CLEAR

(c) LOAD

Contingency STOP to set core to SLJ ERR

(d) RUN

6. To Load Binary Routine through Card Reader and Load EXEC

Depress the following keys in sequence:

(a) REWIND

(b) CLEAR

(c) LOAD

Contingency STOP to set core to SLJ ERR

(d) REQUEST

(e) RUN

7. To Load and Execute Specified Routine in Core from Typewriter

Depress the following keys in sequence:

(a) REWIND

(b) CLEAR

(c) LOAD

(d) RUN

(e) REQUEST

(f) Type in RXΔ routine name

(g) RELEASE

8. Calling Sequence

To Load and Execute Specified Routine from Tape

(a) LA 3, symbol desired

(b) J LODX

9. Calling Sequence

To Load Specified Routine from Tape but Return Control

(a) Load desired return in index register 1

(b) LA 3, symbol desired

(c) J LOAD

C. System Tape Format

1. General

A system tape is created by a DECO run. The first block on the system tape should be BOOT. BOOT will automatically call in EXEC, which should also be on the system tape.

The system tape is terminated by special end-of-file sentinels; searching is normally done by searching forward through the file until the desired name is found. If the desired name is not found on the second pass, the search program will spin.

A section of the system tape containing Job A would appear as:

2. Symbol Block - Job Identification

(a) SCAT word - 3, TCD

(b) Segment of three words

(1) Transfer address of previous program.

(1) JOBA, name of this job.

(c) SCAT word - 16, 0120

(d) Cover register information of previous program.

3. Preamble Block

   (a) SCAT word - signed negatively to make this block transparent to BOOT.

   (b) Control information for SUCO, such as tape assignment parameters.

4. Information Blocks

   As many as 19 pairs consisting of:

   (a) SCAT word - number of contiguous instructions, absolute location of first instruction.

   (b) Segment of contiguous instructions.

5. Perhaps Additional Symbol Blocks with Associated Information Blocks for Segments or Links of the Job

   See sections 2 and 4 above

6. Symbol Block - Transfer Information

   (a) SCAT word - 3, TCD

   (b) Segment

      (1) Transfer address of JOBA (or the last segment or link).

      (2) Name of next job physically on tape.

   (c) SCAT word - 16, 0120

   (d) Cover register information for JOBA (or the last segment or link).

D. Operator-Controlled Executive Operation

1. Loading Programs from Tape

This mode of operation permits individual principal programs to be called from the console and executed. No symbiont operation or tape-assignment will be provided. It is expected that this mode of operation will be used only for preliminary debugging of programs or for occasional runs of rarely used programs.

The following sequence will permit operator-controlled operation with the system tape mounted on UNISERVO 0:

(a) CLEAR

(b) REWIND

(c) LOAD

(d) STOP (if it is desired to reset core to SLJ ERR)

(e) RELEASE

(f) RUN

The system will now spin in a loop awaiting type-in. Programs may be called from the system tape by typing

RXΔNAME

where NAME is the name of the desired program. At the end of the program, an EOJ message is typed out and the system returns to the spin loop. If NAME is not found in two passes of the tape, the program will spin.

2. Loading Programs from Cards

If it is desired to load programs from the card reader, Step (e) in the above sequence should be REQUEST. This will call in the Binary Relocatable Loader which will load and execute binary decks.

E. SUCO Operation

1. General

Operation under Supervisor Control (SUCO) permits simultaneous operation of principal programs and symbionts, with automatic chaining of principal programs and automatic tape assignment. This is expected to be the normal mode of operation for production runs.

Principal programs will normally communicate with SUCO through control cards. An SLJ EOJ should be used to terminate each principal program.

2. Scheduling Principal Programs

Each principal program has the option of specifying the program which is to follow. This is done by means of a control card of the following format preceding the program:

NEXT NAME

where NAME is the name of the program to follow. If no NEXT card is included, SUCO will type out a message upon reaching EOJ in the current program, and spin in a STOP loop until the operator calls a new program. A principal program is called from the console in the same manner as a symbiont. The procedure is outlined below.

3. Memory Clearing

SUCO will normally clear the memory locations used by a principal program before loading. If one desires to preserve a segment of memory, a HOLD control card must be used. The format of this card is described under "SUCO Control Cards".

4. Initiation

The following sequence will call in SUCO and initiate automatic processing:

(a) CLEAR

(b) REWIND

(c) LOAD

(d) STOP (if core reset is desired)

(e) RUN

A message "WHATS NEXT" will be typed out, and the first program to be called should be typed in using the following message format:

RS△KILL△0△CALL△NAMΞ△2

SUCO will then type out tape assignment and accounting information for the called program, and load it in. Subsequent main programs will either be automatically called by NEXT control cards, or may be called from the console using the sequence above, if no NEXT card was provided.

An invalid type-in will produce an 'EH' message followed by "WHATS NEXT".

5. Calling Symbionts

Symbionts may be called by using the same type-in format as above, substituting the symbiont channel number for the 2. This sequence will interrupt the principal program and stop all I-O action while the symbiont is being loaded. Symbionts must not be called during the time that SUCO is loading a new principal program or another symbiont, but only during the running of a principal program.

So long as only principal programs are being run, no system DUMP tape need be specified. However, before a symbiont may be called from the console, a principal program containing a DUMP control card must have been called.

6. Running Symbionts Without a Principal Program

If, at any time after a principal program containing a DUMP card has been run, it is desired to continue running symbionts without a principal program, the "WHATS NEXT" message should be answered by REQUEST, GO (type-in), RELEASE. This will permit symbionts to run, with a spin loop in SUCO acting as principal program. A subsequent GO will cause an SLJ EOJ and allow running of principal programs to resume.

F. Control Cards for SUCO

These control cards are passed through the Designation Run, DECO, which makes up the Preamble Block for a program. All control cards have a 12-0-2 punch in column 1 followed by a space in column 2. (In the following control card formats, Δ indicates that there must be at least one blank space.)

First word = Operator; Final words = Operand.

1. NEXTΔNAME

This specifies the name of the program to be called at EOJ of the program containing the control card. The symbol NAME will be placed in Words 4 and 5 of the preamble on the absolute tape. SUCO will go to the typewriter for the next program if no NEXT card is present.

2. HOLDΔFIRST, LAST

This card specifies the first and last absolute locations of the core segment which is NOT to be reset before loading the program in which the card appears. If no card is present, the entire core segment within the program boundaries will be reset. Only one HOLD card may be included in any one program.

The first and last locations are loaded into Words 6 and 7, respectively, of the program preamble.

3. SYMΔX

   See "Symbiont Storage Allocation" (section III. H. 2. b.) for description. Only Symbionts use this card.

4. DEFΔ$S_1, S_2, \ldots$

   See "Allocation of Symbiont I–O Channels" (section III. H. 3. a.) Only Symbionts use this card.

G. Preamble Block

Control Information for SUCO

Immediately following the symbol block for a program must be a preamble block which has been automatically prepared by the Designation Run, DECO, laid out as follows:

Word

1	"1" for absolute, "2" for dynamically relocatable symbionts, "3" for symbionts requiring relocation in increments of 100.
2	First Location of program.
3	Number of cells required by program.
4 and 5	Symbol of Program to be called NEXT. "0" for symbionts, or if next main program is to be called from CONSOLE.
6 and 7	First and last words of core segment which is NOT to be reset before loading program. Zeros if entire segment occupied by program is to be reset.
8	(No. of Tape Assignment Parameter Entries following) X 4 = No. of words (Nt).
9	Tape Assignment Parameter Entries (4 words per entry) condensed from tape assignment parameter cards.

9 + Nt These entries are described under "Tape Assignment Parameters" (section III. I. 6.) below.

10 + Nt Number of Words of Undefined Symbol Table (Ns).

11 + Nt Undefined Symbols for Operator.

11 + Nt + Ns These entries are laid out as follows:

Symbol = 1-4 words of alpha, all but last minus.

Reference = 1 word containing address where symbol was referenced, same format as relocation reference words.

H. Symbiont Operational Control, Storage Assignment and I-O Allocation

1. Symbiont Operational Control

a. General

The role which the operator and the machine play is reversed with respect to symbionts. For symbionts the operator must initiate all action. He must request a termination and initialization of a symbiont; for tape alteration, opening, mounting, removing, etc., all actions are initiated by the operator rather than by the operating system. This reversal serves two purposes. One is to simplify the principal program's relocation and shorten tape assignment. The second purpose served by this reversal of the positions is that it gives the control primarily to the operator for the operation of symbionts as opposed to the scheduler; since it is felt that the scheduling of symbionts is a real-time problem, it is only the operator who can make intelligent decisions in this regard.

b. Symbiont Debugging

Although a symbiont is nominally not permitted to have debugging references, it may be debugged by running it without a principal program. In this circumstance, the symbiont is permitted to make debugging references since there would be no conflict. Typewriter intervention, however, is only possible in this instance when the symbiont is released; and a symbiont should be initially debugged as a principal program without release.

c. Beginning and Ending Symbionts

Symbionts are called and terminated only from the console. The message form is as follows:

RSΔKILLΔa,b,cΔCALLΔNAMEΔa

in which a, b, c are channel designators. This message indicates that the symbionts currently using those channels following KILL are to be terminated and that the program indicated by NAME is to be loaded into the space vacated and assigned Channel a. Principal programs may be called in the same way, using a channel designator of 2.

Both KILL and CALL must be entered. If there are no symbionts to be KILLed, an operand of 0 should be entered.

Symbionts must be written in such a fashion that the dominant channel release re-entry is also the initial entry for starting the symbiont, as well as the entry for acceptance of typed-in messages. The transfer card of a symbiont must specify the address of this entry point.

Symbionts are responsible for their own orderly termination upon reaching an EOF or EOJ condition. This should include typing a message to inform the operator that no further work remains, identifying the channels involved. The operator can then either restart the symbiont with additional input or clear it out by use of the KILL message.

2. Symbiont Storage Assignment

a. General

Symbionts are loaded at the top end of core. These symbionts may be located either statically, which is to say that their location is specified at designation time through a parameter card, or dynamically in memory. If a symbiont is located statically in memory, then the automatic symbiont memory allocation features of the supervisor are foregone and scheduling and memory area designation become the responsibility of operating personnel. If the symbiont is specified as a dynamically relocatable symbiont, then the system will provide automatic memory allocation for the symbiont at all times and will group the symbionts in contiguous areas at the top of available core. Both dynamically relocatable symbionts and statically relocatable symbionts can be in the machine simultaneously.

b. Storage Allocation

Symbionts may be of one of two types, absolute or dynamically relocatable. Of the latter, certain may be relocatable only in increments of 100 locations, due to the requirements of the punch and card reader buffers.

A SYM control card in the following form is used to indicate which type is desired:

SYMΔX

in which

X = R for relocatable

X = H for relocatable in increments of 100 only

X = A for absolute

All symbionts must be accompanied by a SYM control card. Absolute symbionts are loaded into whatever portion of upper memory is specified by the programmer, and are not relocated during execution. They should normally be written to occupy memory locations with the highest addresses practical, as no relocatable programs will be loaded above them.

Dynamically relocatable symbionts are stored in upper memory in the order of loading, and are relocated to pack from the top of memory down. To illustrate this, let A, B, and C be three symbionts. Let U represent the address of the highest cell in the relocatable area storage, and N_A, N_B, and N_C represent the number of cells required for the symbiont indicated in the subscript. Then, if A, B, and C are loaded in that order, they will occupy storage locations as follows:

A will occupy from cell $(U - (N_A - 1))$ to U

B will occupy from $(U - (N_A + N_B - 1))$ to $(U - N_A)$

C will occupy from $(U - (N_A + N_B + N_C - 1))$ to $(U - (N_A + N_B))$

When a new program is called, SUCO will determine whether sufficient vacant space exists in memory. If so, the relocatable symbionts currently in memory will be relocated to provide space at the proper point in memory and the program will be loaded.

If the new program is a principal program or a relocatable symbiont, relocation will be upwards. If it is an absolute symbiont, relocation will be downwards. In any case, the remaining vacant space in relocatable core will be above the highest location of the principal program and below the lowest address of the lowest relocatable symbiont. The figure at the end of this section shows the scheme for storage allocation in multiprogramming applications.

If there is not enough vacant space anywhere in memory, a message will be typed indicating how many additional cells are required and the program will go into a stop loop. The operator then has the option of either:

1. KILLing other programs to provide the needed space and reCALLing the desired program or

2. CALLing a program with smaller space requirements.

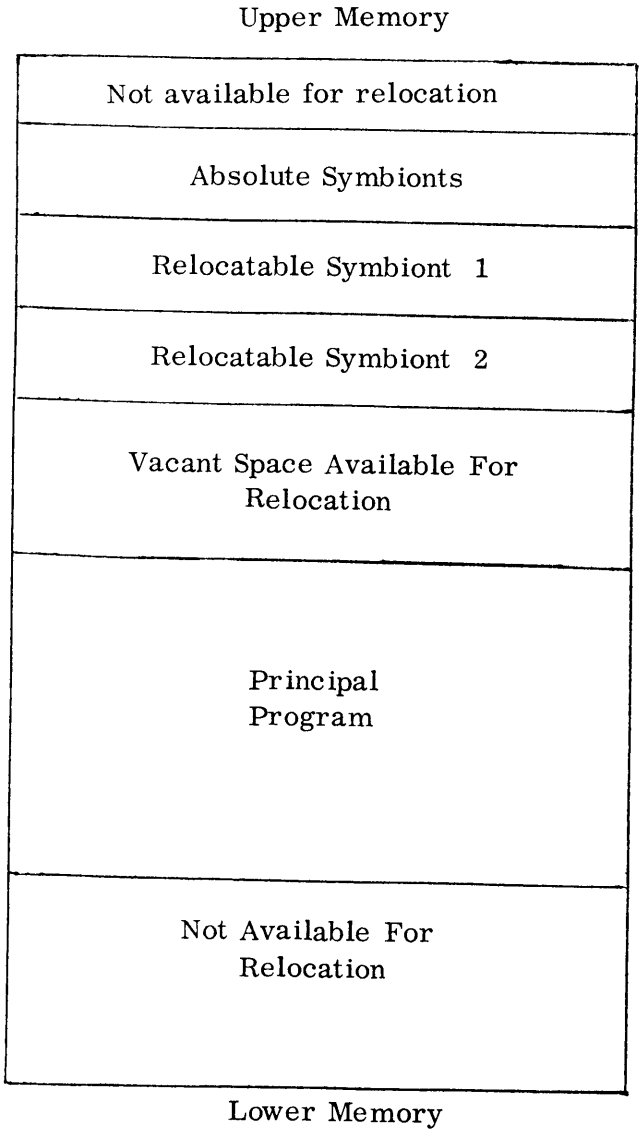
Typing GO will then permit execution of the typed-in message and another cycle of storage availability analysis.

If GO is entered without having first entered a KILL/CALL message, SUCO will remain in the stop loop operating as a principal program if a principal program EOJ condition exists. Otherwise, SUCO will exit and resume the principal program which was interrupted.

c. Dynamically Relocatable Symbionts - Programming Rules

In order for a symbiont to be dynamically relocatable, the only programming requirement is that all working storage usages be explicit with respect to any relocation indications. A particular example is a cell which is the object of an SLJ. This cell must be specified as a relocatable cell by placing in it a dummy value such as, for example, +$+00, which will both flag the location for documentation purposes and cause this location to be recognized by the assembler as being a relocatable location. It is not permissible to store in any cell of the symbiont both relocatable information and non-relocatable information.

A dynamically relocatable symbiont may potentially be relocated at any time that it is released. Since at the time that it is released the registers are not saved, there are no requirements with respect to register usage as to the type of information that they contain.


Upper Memory
Not available for relocation
Absolute Symbionts
Relocatable Symbiont 1
Relocatable Symbiont 2
Vacant Space Available For Relocation
Principal Program
Not Available For Relocation
Lower Memory

3. Allocation of Symbiont I-O Channels

a. General

In order to avoid conflicts in usage, symbionts should normally refer to tape units and I-O channels symbolically. The symbols may then be defined by the operator at execution time, in accordance with hardware availability.

Symbols may be indicated as operator-defined by a DEF control card of the following format:

$$\text{DEF}\Delta S_1, S_2, S_3, S_4, \ldots$$

in which S_1, S_2, etc., are otherwise undefined symbols.

This card will cause SUCO to generate a type-out of each symbol, following which the operator may type in the symbol definition. Such symbols must be in the form of 15-bit addresses.

It is suggested that a list of standard symbols for tape and I-O channels be adopted by each installation in order to minimize operator confusion.

It should be noted that this option is available only to symbionts. Main program symbols may not be operator-defined.

b. Multiple-Channel Symbionts

It is permissible for a symbiont to release on more than one channel. However, if this is done then the symbiont will be responsible for simulating a not-used status on channels other than the currently released channel by storing minus zero in the corresponding release entry MPRL + channel number.

I. Tape Operational Control and Assignment - UNISERVO III

1. General Descritpion

Tape assignment will be performed at execution time and between runs. The responsibility for tape assignment will be divided between two routines. One is a relocatable subroutine and the other will be part of the supervisor. The routine in the supervisor will be responsible for the analysis of tape parameter cards and cleaning up of tape assignments upon job completion. The relocatable routine will be responsible for preparation of "posting" messages for output files, handling of tape swapping, and of closing files.

All pertinent tape assignment information will be retained in a table in the communication region consisting of 16 words per tape channel pair. This table will be accessible for interrogation and manipulation by any routine so that it will be possible for programs other than the tape assignment routines to make tape assignment adjustments.

It is possible for a generalized program which accepts card images directly for control, to make distinct tape assignments for each run instead of relying upon a designation pass. This file assignment permits early mounting instructions for specified optional inputs such as library tapes.

2. Operator Messages

a. General

Except for operator intervention not covered here and system outguessing with free entries, all unit mounting and dismounting must be at the instigation of the typewriter. The system has been designed to permit early mounting instructions where possible by use of the SAVE instructions, and to be self-protecting via the dismount status. It permits operation to go from one run to another without any stop required to be sure all tape reels are mounted. A description of the messages and corresponding operator action follows.

For explanation of the term "alias", see Sect. III, p. 23 (e.).

b. "DISMOUNT unit"

The unit specified will be rewound with interlock and is to be dismounted. The unit specified should remain in "change tape" status until further instructions are received.

c. "MOUNT BLANK unit"

If the specified unit is not in a dismounted status it will be rewound with interlock. If already in a dismounted status, no I-O action will occur. In either case, the next reference to this unit will expect a blank reel to be on this unit.

d. "MOUNT alias-k ON unit"

If the specified unit is not in a dismounted status, it will be rewound with interlock. If already in a dismounted status, no I-O action will occur. In either case the next reference to this unit will expect an input reel to be on this unit.

e. "POST alias-k ON unit"

The operator should now affix a corresponding reel sticker to the specified UNISERVO. It is important that this be done at this time, inasmuch as no further typewriter reference with respect to alias-k will be made. Instructions to remove this reel will instead be implicit in a later DISMOUNT or MOUNT message.

f. "alias-k unit NOT CLOSED DISMOUNT"

This message will occur only if a run is terminated without closing a protected file. It would normally be expected that the results of the affected run would be invalid.

g. "alias ASSIGN file-entry file-entry DISAGREE"

This message will occur if the saved file-entry does not carry the same file alias as that specified on the ASSIGN parameter card. A subsequent wait will occur, whereupon the operator can determine whether to proceed.

h. "alias ASSIGN file-entry file-entry NOT SAVED"

Specified file-entry was not saved by previous job. Program will proceed as if no ASSIGN parameter had been given. Operator should intervene if corresponding required reels are not readily available.

i. "alias file entry ON unit ERRONEOUS REQUEST"

File-entry is already in use or else unit was not off-line. A subsequent delay for operator intervention will occur.

j. "unit OFF-LINE"

Specified unit has been set to OFF-LINE status and is not available to programs.

k. "file-entry HAS USAGE CONFLICT"

Conflicting parameter cards have occurred for the specified file-entry. A subsequent delay for operator intervention will occur.

l. "unit NOT READY"

System is waiting for specified unit to be ready so that input output may proceed. Specified unit may be rewinding or in the process of tape changing at the time of this message.

m. "file-entry SAVED NOT ASSIGNED"

A file entry which was saved by previous job was not assigned by current job. A subsequent delay will occur.

3. Procedures for TakingUNISERVOSOff Line

In the event that, due to mechanical failure or some other reason, a UNISERVO is unavailable, the operator should take it off line. This may be done as follows:

Type in LIΔTPSWICH

This will cause SUCO to pause before the next tape assignment phase and type out an inquiry. The operator should then type in

RSΔn1ΔOFFΔUSEΔn2 and Enter GO

in which n 1 is the two-digit number of the UNISERVO to be taken off-line, and n2 is the two-digit number of the UNISERVO to be substituted for it.

SUCO will then

(a) Find the file entries to which n 1 and n2 are assigned.

(b) Exchange the servo numbers.

(c) Set the file entry receiving n2 ON LINE on symbiont channel 2 (main program) and the file entry receiving n 1 OFF LINE.

When an off-line UNISERVO becomes available again, it may be placed on line by the same procedure used for taking it off, except that the type-in should read

RSΔn1ΔON

This will place the file entry now containing n 1 ON-LINE on symbiont channel 2 (main program).

When a file entry has been taken off line by the above procedure, the operator should avoid using that file entry in defining symbolic tape files for symbionts.

If a main program attempts to use the off-line file, a message will be generated and SUCO will wait for the operator to either type in a unit "ON" message, or to interchange units with an "OFF USE" message.

4. Permanent Assignment Routine

a. General

The permanent assignment routine (COSTA), which is relocatable, accepts calling sequences from the user, types out pertinent messages and performs a limited amount of I-O action for rewinding and interlocking tapes. It accepts the following calling sequences:

(1) To open output file:

```
SLJ     OPEN
 +      file entry
```

(2) To close a file:

```
SLJ     CLOSE
 +      file entry
```

(3) To swap input reels:

```
SLJ     SWAP
 +      file entry
```

In each of the above calling sequences, file entry specifies a location in the tape assignment table. The word in the calling sequence must not have bits except in Positions 1 - 10.

b. Open Routine

The open routine will produce a POST message if an output reel is to be saved.

c. Close Routine

The close routine will rewind the specified reel and, if protected, produce a DISMOUNT message. Unless the file is saved, it will set the symbiont channel designation to "free".

d. Swap Routine

(1) Input

The swap input routine first determines whether an alternate reel has been specified. If an alternate reel has been specified, the unit numbers will be inter-changed between the file entry and its alternate. If the next reel does not correspond to a prescribed maximum (via file parameter card), then a MOUNT message will be produced and the current reel rewound with interlock. If the next reel does correspond to the prescribed maximum then the current reel will be rewound with interlock, a DISMOUNT message will be produced, and the file entry symbiont channel will be set to "free". If no alternate has been specified, a simple MOUNT message and rewind with interlock will be produced. In any case, the reel number will be incremented by one.

(2) Output

The swap output routine causes the current reel to be rewound with interlock and a MOUNT SCRATCH message to be produced. If an alternate reel reference is indicated, the unit will be interchanged. A POST message will then be produced for the proper unit.

e. File Alias Table

Corresponding to each file entry there is a two-word file alias entry in the file alias table. The first six characters of this entry determine the alias and the last two contain the current reel count.

f. Usage

The item advance routines (high level I-O) will contain their own interface with the tape assignment routines. Programs which require tape mounting, dismounting, or swapping must provide their own interface with the permanent assignment routine. Symbionts may not communicate directly with SWAP, OPEN, and CLOSE routines. Since SWAP may not be used, symbionts may not specify alternate entries.

5. Inter-Run Assignment Routine (SUCO)

a. General

The inter-run assignment routine cleans up after the previous job, canonizes the assignments, and interprets the assignment cards for the next job.

b. Clean-Up

Each file entry used by a symbiont which is being terminated is examined. The only file entries which should still remain assigned are unprotected files, unused input alternates, and saved files, unless the terminating program failed to close all of its files, in which case messages would be produced for any protected files still remaining assigned.

Each file entry assigned to the affected symbiont channel is examined in sequence and (a) if it is an input alternate, the dismount bit is set and the entry is released; (b) if the entry is not protected, the entry is simply released; (c) if protected, a NOT CLOSED DISMOUNT message is produced and the unit is rewound with interlock; and (d) if protected and referenced by input alternate, then the alternate is rewound with interlock and a DISMOUNT message is produced.

c. ASSIGN Processing

The ASSIGN parameters are processed first.

The old assignment table is saved and all of the ASSIGN's are processed in order of receipt. Following the completion of the processing of the ASSIGN entries, the remaining file entries are transferred from the old assignment table to the new one.

d. Canonization

Canonization is accomplished following the processing of the ASSIGN entries by examining all of the then free or off-line entries and making any interchanges which will permit a unit to fall into its canonical location. Dismount status is retained. Canonical references are accomplished by a fixed table in the supervisor which can be altered as an installation option.

e. INPUT, OUTPUT, INEX, SCRACH Processing

These parameters are processed as a group. Particular attention is made to any possible conflicting usage, and appropriate messages are produced.

f. SAVE, ALT Processing

SAVE and ALT parameters may require information from previous parameters and are processed in the last group.

g. Summarization

Summarization permits the operator to be aware of which physical units are in use and which are free.

6. Tape Assignment Parameter Cards

a. General

Tape assignment parameter cards are placed with the

beginning parameter information for a run. They are condensed by the designation run and written on the system tape as part of the run preamble. During the initialization of a run by the supervisor, they are examined and appropriate action is taken.

The tape assignment parameter card format is:

Columns	
1 - 6	Alias
7	Blank
8 - 9	File Number (k)
10	Blank
11- 16	Operation
17	Blank
18...	Assignment numbers right justified in Columns 20, 24, 28, 32, 36,....

The file-alias has no logical attachment to any symbols generated by a program and is carried as a mnemonic device only. Its sole employment is on tape assignment paramenter cards and on correspondingly generated tape mounting, posting, and dismounting instructions.

The file number specifies the entry in the tape assignment table. A description of the permissible tape assignment operations follows.

The function of the assignment numbers depends upon the operation involved and is described in the description of these operations. It may be a physical unit number, a file number, or an estimate of the number of reels. All numbers are decimal.

b. ASSIGN

alias k ASSIGN k2, k3

Assigns file entry k2 from the previous run to file entry k3 of the current run. This is accomplished by interchanging the unit numbers between the two file entries. A check is made to see if the previous alias agrees with the alias on the ASSIGN card, and, if not, an error message is produced. File number (k) is always blank (ΔΔ) in this card.

c. INPUT

alias k INPUT n

Describes file k as being a protected input file. Causes beginning-of-job mounting instructions and end-of-job dismounting instructions. If reels are not in dismounted status at the beginning of the run it produces rewind-with-interlock instructions. Sets the 'input' bit. If n is non-zero, it specifies the expected number of reels, thereby permitting an early release of the alternate, if any. An incorrect n will not cause an error.

d. INEX

alias k INEX n

Describes file k as being an unprotected input file. Causes beginning-of-job mounting instructions only. If reels are not in "dismounted" status at the beginning of the run it produces appropriate rewind-with-interlock instructions. Sets the "input" bit. If n is non-zero, it specifies the expected number of reels, thereby permitting early release of the alternate, if any. An incorrect n will not cause an error.

e. OUTPUT

alias k OUTPUT

Describes file k as being a protected output file. Causes assurance of scratch tapes at beginning-of-job and appropriate end-of-job dismounting instructions. Tests to see if reel is dismounted and, if so, produces MOUNT BLANK message.

f. SCRACH

alias k SCRACH

Describes file k as being a scratch reel. Causes assurance of scratch at beginning of job, and produces MOUNT SCRATCH message at beginning of job if file is dismounted.

g. ALT

alias k1 ALT k2, k3,....

Describes k1 as being an alternate to files k2, k3,....,and sets the alternate reference bit in entry k2, k3,..... . If k2 is an INPUT file, then there should only be the entry k2 in the list. If k2 is an input reel, then a MOUNT message will be produced and the unit rewound with interlock if not dismounted. If k2 is an output reel, then a MOUNT SCRATCH message will be produced if the unit is "dismounted".

h. SAVE

alias k SAVE k1

Sets the "save" bit in file entry k1 and thereby causes the file to be carried over to the next run. If file k1 has not been described as a SCRACH,

INPUT, INEX or OUTPUT file, it causes carryover anyway. If the file is not in use, it causes MOUNT message and rewind-with-interlock, if appropriate. As in ASSIGN, the file number (k) in this card is always blank (ΔΔ).

i. DUMP

alias k DUMP

Specifies the file entry for the system dump tape.

j. DIAG

alias k DIAG

Specifies the file entry for the system diagnostic tape.

7. Tape Assignment Table

Central control for tape assignment lies in the contents of the tape assignment table. Each word of this table completely defines the status of a file entry, insofar as it affects tape assignment. All tape assignment functions are accomplished by interrogating or manipulating the contents of this table. The first entry in the tape assignment table at 0200 will normally be assigned to the system tape file. The format of these words follows:

bits	function
sign	Not used
24 - 21	Logical unit number
20	Not used
19	Protected
18	Dismounted unit
17	Input
16	Alternate reference
15	Saved
14 - 11	Not used
10 - 7	Symbiont channel number
6 - 1	m field

The logical unit number will be the actual value which will appear in the I-O function specification word.

A file which is described as protected is to be removed after use. It may pertain either to an input reel, in which case the write-permissive ring would be expected to be missing at the time of mounting, or to an output reel, in which case the operator would be expected to remove the reel and extract the ring upon completion of usage. In either case, the file assignment program will rewind with interlock when closing or swapping the specified reel.

The dismounted unit bit is set whenever a reel is rewound with interlock without immediate operator instructions for replacing the reel. This permits the supervisor to avoid redundant rewind with interlock when issuing mounting instructions, if any, for the next job.

The "input" bit is set if the file is an input file with input reels to be mounted.

The alternate reference bit is set if the file entry is referenced by an alternate reel parameter. This bit is interrogated during tape swapping; the alternate reel is substituted if so indicated by this bit.

The "saved" bit is set if the file is to be retained on the computer for the following job. It causes dismounting to be prevented and "protect" information to be remembered.

The symbiont channel field is used to describe the symbiont with which the file is associated. "Free" tapes are assigned a channel number of 0. A channel number of 2 is used for principal programs. Off-line tapes are assigned a channel number of 1.

The m field is normally used in conjunction with the alternate drive indication to specify the file number to be used as an alternate drive. In the case of output drives, more than one file word can be chained to a single alternate.

If an alternate drive is used for input, then the reel count specified is placed in the m field of the alternate reel.

IV. SYNCHRONIZER CONTROL

A. Basic Dispatchers - Calling Sequences All routines use the AR's and index registers 1 and 2.

1. Card Reader

a. Request and Verify

	SLJ	CDRQ
+1.	+	location of current card stored here on ready return
+2.	. . .	error returned with indicators in AR4
+3.	. . .	not-ready return
+4.	. . .	ready return

Function specifications are prepared automatically by the basic dispatcher for this channel only. Entry to request implies release of previous card buffer and commitment of five cards to the reader. Because of the real-time operation of the card reader, the timing and error-verification functions which normally have a separate calling sequence on other channels are incorporated into the request calling sequence for the card reader channel only. Symbiont programs use the not-ready return to release.

b. Errors

Cards are normally directed to stackers 1 and 2, alternating every 500 cards. All cards in the reader at the time of an error or fault are directed to stacker zero. When a request is made for the buffer that would have contained the error card, the error return is taken by the request subroutine. Bits in AR4 are set on in the same position as used in the TIO instructions and may be interrogated with the same logic to determine the cause of error. Operator communication and restart are determined by the requesting program.

c. Mode

A 1 in Bit 18 of CDQF (request communication word) in the synchronizer communication region specifies translation. When mode is changed, four images will subsequently be supplied in the previous mode. The dispatcher is initially loaded for the translated mode.

d. Release

```
          SLJ     *$+1
+1        +       CDRL
+2        +       save
+3        +       index register 3 value
+4        . . .   return
```

Upon return, an entry to the request subroutine will produce an immediate "ready" exit. Save is the label of a nine word area within the program releasing for holding the environment of a program interrupted (released to and return from). The nine word area contains CC, LEG, AR8, AR4, AR2, AR1, IR1, IR2, and IR3, in this order. LEG represents the low, equal, greater indicators.

2. Card Punch

a. Request

Load desired function specification in AR8 (address within the function specification must be a multiple of 64)

```
          SLJ     PCRQ
+1        . . .   deferred return
+2        . . .   accepted return
```

If accepted, another request may be placed immediately. If deferred, no more requests may be placed until the program receives a "ready" return from the verify subroutine. Symbionts should release at this point. Principal programs may wait by looping through the verifier. If a request is made following a "deferred" return without intervening verification for ready, the request subroutine will wait until the next interrupt on the punch synchronizer, process the request, and return with a "deferred" exit.

b. Verify

load function specification to be verified in AR8

	SLJ	PCVF
+1	. . .	error return with indicators in AR4
+2	. . .	not-ready return
+3	. . .	ready return

If punching at full speed, two functions will initially be accepted and the third will produce the deferred return from the request subroutine. For simplicity of error handling, the earliest outstanding request should be the one verified although any previously-requested function may be used; a ready return implies completion of all requests prior to the one verified. Symbionts will not encounter the "not ready" exit if they release on a deferred request. Principal programs may wait for completion of a given action by making the not-ready return a jump to the SLJ. Data errors and faults produce the error return. The earliest outstanding request has caused an error or fault and no subsequent requests have been processed. The bits in AR4 are set on the same positions as used in the TIO instructions and may be interrogated with the same logic to determine the cause of error.

For data error, the basic dispatcher has directed the error card to stacker 1 on the assumption that normal punch usage will direct cards to stacker zero. The disposition of the card currently at the post-punch station will depend on the next function requested. Repunching of error cards, operator communication, etc., is determined by the punch user.

c. Errors

The basic dispatcher will direct a card incurring a data error interrupt to stacker 1 and preserve the I-O indicators for interrogation by the user at the error return from the verifier subroutine. All other action is determined by the user.

d. Release

```
          SLJ     *$+1
+1        +       PCRL
+2        +       save
+3        +       index register 3 value
+4        . . .   return
```

Upon return, an entry to the verifier will produce an immediate "ready" return. The save area must be a reservation of nine words.

3. Printer

a. Request

load desired function specification in AR8

```
          SLJ     PRRQ
+1        . . .   deferred return
+2        . . .   accept return
```

See description above under Card Punch

b. Verify

		load function specification to be verified in AR8
	SLJ	PRVF
+1	. . .	error return with indicators in AR4
+2	. . .	not-ready return
+3	. . .	ready return

The description above, under Card Punch, applies with the following difference: the basic dispatcher takes no action whatsoever in processing a print error. Operator communication and reprinting action are determined by the user.

c. Errors

The basic dispatcher will reset the print synchronizer and collect the I-0 indicators for interrogation by the user at the error return from the verifier subroutine. All other action is determined by the user.

d. Release

	SLJ	*$+1
+1	+	PRRL
+2	+	save
+3	+	index register 3 value
+4	. . .	return

Upon return, an entry to the verifier will produce an immediate "ready" return.

4. Tape Read - UNISERVO III

a. Request

load desired function specification in AR8

	SLJ	RTRQ
+1	. . .	deferred return
+2	. . .	accepted return

If accepted, another request may be placed immediately. If deferred, no request should be placed until the program receives a "ready" return from the verify subroutine. Both principal and symbiont program normally wait by looping through the verifier. Symbionts do not normally release their tape synchronizer. Re-entry to the request subroutine following a "deferred" exit without intervening verification for ready will cause the request subroutine to wait until the next tape-read interrupt servicing the dispatcher priority class requested. It will then process and return via the "deferred" exit.

b. Verify

load desired function specification in AR 8

	SLJ	RTVF
+1	. . .	not ready return
+2	. . .	ready return

When reading tape at full speed in the absence of higher-priority requests from other users of the tape-read synchronizer, two function specifications will initially be accepted, and the third will produce the deferred return from the request subroutine. Although any previously requested function may be verified, for simplicity of

higher-level dispatching the earliest outstanding request should be the one verified.

Programs will normally wait when encountering the "not ready" return by looping back through the verify subroutine. Since the tape synchronizers are shared by up to four classes of programs, provision is made for the function to be entered on either a request or a demand basis. A request is indicated by a positive sign bit on the function specification as it stands in its reservation word. The request may be converted to a demand at any time (normally at the "not ready" verifier exit) by reversing the sign to minus. This does not involve re-entry to the request subroutine. However, interrupts should be momentarily prevented during the reversal of the sign. A demand by one user overrides all other normal requests but defers to a simultaneous demand by the user of higher priority.

c. Errors

The basic dispatcher provides standard error procedure for tape reading. This includes automatic re-tries for type B errors. For faults, the read function specification is repeatedly re-issued until the condition is corrected by the operator or terminated by a type in.

d. Release

	SLJ	RTRL
+1	+	save
+2	. . .	return with completed function in AR4

Return is made on every tape-read interrupt. Since the synchronizer may be shared by up to four programs, the completed function specification is supplied upon return from a release so that the releasing program may

determine whether this interrupt is the one for which it is waiting. This can be determined by comparing the completed function with the earliest outstanding request. When the two match, the release is completed. When they do not, the program should release again.

Normally only one program may release on the tape-read synchronizer. To permit more, a special routine would be needed to associate the completed functions with the correct initiating programs.

e. Contents of MAC

To permit efficient operation of the basic tape dispatcher, the read channel memory address counter is not normally stored. If its contents are essential to a programs operation (as in a generalized tape dump, for example), the basic tape dispatcher can be directed to store the channel by issuing the following instructions for every tape-read request linkage:

```
LA      8, (function specification)
SAN     8, RTSC (any negative value is sufficient)
SLJ     RTRQ
```

This sets a switch in the tape-read request subroutine which causes dispatching to proceed in the start-stop mode and causes the channel to be stored in a special table within the dispatcher. This table is referenced by the label WTTT and consists of 16 entries, corresponding to servo numbers 0 through 15. Correct information will be found in the table only if the using program places one basic request at a time and always verifies it before making another. This non-buffered technique combined with the start-stop operation of the input servos reduces the efficiency of the "store channel" operation considerably and is recommended only

when information concerning the data on a tape cannot be obtained in any other way.

Such multiple symbionts may require the "store channel" operation, the switch RTSC can only be restored between jobs by supervisory control. The restoring instruction is:

```
SZ          RTSC
```

f. Load-Point Test

A function specification for a load-point test (bits 17, 18, 19, and 20 all on) may be requested through the basic read-tape dispatcher. If error indicators are turned on as a result of the issuance of the load-point test, the indicator bits will be collected into a word in the same bit positions used by the TIO instruction and stored in the table WTTT (the same table used for the "store channel" operation). This table consists of 16 entries, corresponding to servo addresses 0 through 15. The error flag word, if any, will be found (after the request has been verified in the usual manner) in the table entry corresponding to the servo number contained in the re-request. Since the table entries are not reset by the dispatcher, it is recommended that the requesting program zero-out the appropriate table entry word prior to issuing the load-point test, in order to distinguish error-free results (tape at load-point). An intermediate level subroutine is available to issue a load-point test and analyze the results (see Support III).

5. Tape Write - UNISERVO III

a. Request

```
                        load desired function specification in AR8

          SLJ           WTRQ

+1        . . .         deferred return

+2        . . .         accepted return
```

b. Verify

	SLJ	WTVF
+1	. . .	not-ready return
+2	. . .	end-of-tape return
+3	. . .	normal ready return

A successfully completed function at the end of tape produces the end-of-tape ready return. This is to notify any higher-level dispatcher or user program of the end-of-tape condition on the unit specified in the function verified. See Tape Read for the remaining description.

c. Errors

The basic dispatcher provides standard error procedures for tape writing. This includes re-writing of information on Error A conditions. For faults, the write function specification is repeatedly re-issued until the condition is corrected by the operator or terminated by a type-in.

d. Release

	SLJ	WTRL
+1	+	save
+2	. . .	return with completed function in AR4

Description as above for Tape Read

6. Symbiont Tape I-O - UNISERVO III

a. Request and Verify

In order to prevent interferring with principal program tape input/output, special indirect tape request and verify references must be used by symbionts. For fixed symbiont channel assignments, the linkage should

be of the form:

```
SLJ     *RTSRQ+CHNO (in lieu of SLJ  RTRQ)
SLJ     *RTSVF+CHNO (in lieu of SLJ  RTVF)
SLJ     *WTSRQ+CHNO (in lieu of SLJ  WTRQ)
SLJ     *WTSVF+CHNO (in lieu of SLJ  WTVF)
```

in which CHNO is the number of the channel upon which the symbiont is releasing.

If the channel numbers is not to be defined at assembly time, special double-reference, indirect address words must be used instead of the addresses above. This can be done in the following manner:

```
RDREQ           +*RTSRQ
        RES     -1
                +*CHNO
```

in which CHNO is to be externally defined as well as RTSRQ. An SLJ *RDREQ would now give the correct external linkage for a symbiont.

B. Basic Interrupt Analyzer

A basic interrupt analyzer services all I-O interrupts and determines which synchronizer requires control. Associated with each synchronizer is a basic dispatcher. The basic interrupt analyzer transfers control to the appropriate dispatcher for suitable processing of any particular interrupt.

C. Basic Dispatchers - General Information

1. Introduction

The basic dispatchers perform the necessary error checking, take varying degrees of corrective action if needed, and attempt to load the standby location with a new I-O function before returning control to the interrupted program. For interrupt servicing, the basic dispatchers operate independent (asynchronous) of program using the central processor. Communication and synchronization with these dispatchers by programs using synchronizer control is described in detail below.

2. Use of the Central Processor

The basic dispatchers are designed to accomodate a multi-programming use of the central processor wherein the principal program uses tapes exclusively and shares (through interrupts) part of its time with one or more smaller programs whose processing cycle is determined by the speed of one of the pieces of peripheral equipment. Any program other than a principal program is called a symbiont. These smaller, symbiont programs (card-to-tape, tape-to-print, tape-to-punch) relinquish control whenever the peripheral unit they are using is not yet ready to accept a new I-O function, a technique hereafter called releasing. This permits the principal program to retain control of the central processor for all but a minimal amount of the peripheral processing cycle.

3. Use of the Synchronizers

One program only (either principal or symbiont) of a group of programs sharing the central processor may use a specific general-purpose synchronizer with its associated dispatcher. Thus, if the principal program requires a card reader, a card-to-tape symbiont using the same card reader may not be run at the same time as the principal program.

The present tape synchronizer dispatchers allow four programs to use the tape-read synchronizer, and four programs (not necessarily those using the tape-read synchronizer) to use the tape-write synchronizer.

A synchronizer communication region within the dispatcher for each synchronizer contains information (request or reservation words which are I-O function specifications) which is used to control loading of the appropriate standby location. Each general-purpose synchronizer communication region contains one word of such information which is automatically assigned to the one program using a particular general-purpose

synchronizer. The tape synchronizer dispatchers maintain a synchronizer communication region of four such reservation words which are currently assigned the following priorities:

Tape reading

Priority 1.	Tape-to-punch symbiont
Priority 2.	Tape-to-printer symbiont
Priority 3.	Principal program
Priority 4.	Unassigned

Tape writing

Priority 1.	Card-to-tape symbiont
Priority 2.	Principal program
Priority 3.	Unassigned
Priority 4.	Unassigned

The I-O requests of symbionts are given higher priority (initiated first) than principal program I-O requests, since the symbionts are attempting to maintain maximum speed on the peripheral units they are using. To prevent penalizing the principal program, symbiont programs are required to release (relinquish control) to their peripheral unit basic dispatcher whenever they find their peripheral unit synchronizer busy. The basic dispatcher then will switch control away from the symbiont. This restores control for the major portion of the processing cycle to the principal program. A principal program never releases, and thereby retains primary control of the central processor.

4. Dispatcher Communication

The basic dispatcher for each synchronizer maintains a communication region is unindexed locations in low order memory (below 2000_8). A list of base locations for the various dispatchers is maintained as part of the general I-O communication region to permit the dispatcher themselves to be self-locating. All the synchronous I-O subroutines ("synchronous" refers to those parts of the dispatchers directly communicated to by programs) use the four arithmetic registers and index registers 1 and 2. Information in these registers and the comparison indicator settings at the time of program communication with the I-O subroutines will not be saved. On the other hand, the asychronous portions of the I-O system ("asynchronous" refers to those parts of the dispatcher entered upon hardware I-O hardware I-O interrupt) save and restore all registers and indicators used.

Additionally, the dispatchers maintain storage within themselves for the function specification currently active and the function specification next to be loaded into standby (reservation word).

The basic element of communication between an I-O synchronizer dispatcher and a program using the dispatcher's associated synchronizer is a request word (I-O function specification) which may be of any desired configuration except all-zero. The validity of function specification from a hardware point of view is not checked by the dispatchers. Therefore, if processor or channel errors are to be avoided, care should be taken in the fabrication of function specifications in programs using synchronizer control. The dispatchers do insert an interrupt bit (bit 16) in all function specifications received from requesting programs.

If a requesting program fails to have an interrupt bit in a requested function specification, the bit inserted by a dispatcher causes the requested function specification and the executed function specification to have different configurations. This

usually causes a premature ready return from the verify subroutine which inturn causes problems in the requesting program. Therefore, function specifications submitted to the dispatchers should always have an interrupt bit.

D. Dispatching - General

1. Request

To initiate or request I-O action, a function specification word for the desired synchronizer activity is loaded into AR8 and a calling sequence (subroutine linkage) written which begins with an SLJ to the addressed request entry point of the appropriate dispatcher. (See Part A of this section for specifications). As a consequence of the dispatching method used, the request subroutine, after processing the request, exits to one of two return address lines in the calling sequence - - accept or deferral.

If the synchronizer is not active at the time of the request, the function specification word is loaded into the standby location for immediate initiation. In this case, the request subroutine will return to the requesting program via the requesting program's acceptance return line. A new function specification can and should be requested for the same synchronizer immediately in order to maintain maximum I-O speed.

On UNISERVO-III channels, if the synchronizer is currently operating but the standby-interlock indicator is off, a request will be loaded into standby in order to permit the servos to operate in the "non-stop" mode. This request will be initiated immediately at the next interrupt, while the basic dispatcher processes the results of the interrupt. In this case, as in the case where the channel is not active, the request subroutine will return to the requesting program via the acceptance return line.

A deferred return (see below) will be taken for UNISERVO-III's only when the standby interlock indicator is on at the time of the request.

When synchronizers other than UNISERVO III synchronizers are currently active, the requested function is placed in the reservation word for loading into the standby location at the next interrupt. For those synchronizers which interrupt upon initiation of a function specification, the function specification is loaded into the standby location as well as the reservation word in order to cause the next interrupt. The request subroutine then returns control to the requesting program via the requesting program's deferral return line indicating that maximum I-O speed has been attained. No new function specifications should be requested after a deferred return until an entry to the verify subroutine results in a ready return. The verify subroutine determines if successful completion of the currently active function specification has occurred. If a new function specification is requested after a deferred return, the request subroutine will wait until present action on the synchronizer is completed (loop internally), and a processing delay occurs until the next interrupt. The request subroutine then loads standby with the request in the reservation word, places the new request into the reservation word, and exits to the requesting program's deferred return line.

Since the deferred return occurs when the synchronizer is busy, a symbiont program uses the deferred return as the signal to release (see below). A principal program which attempts to operate the synchronizer at full speed uses the deferred return as the signal to verify its earliest outstanding request. A principal program which does not attempt to operate the synchronizer at full speed may ignore the deferred return.

2. Verify

The verify subroutine is used to determine whether a requested function has been executed. Entrance to the verifier is normally made by the using program to check the completion of a previously requested function specification. The function specification to be verified is loaded into AR8. A ready return from the verifier indicates that the input or output function has been completed. A not ready return will normally indicate to a program

that it should loop back to verify until a ready return is indicated.

In a conventional buffering system, verification should be made normally prior to supplying data from an input buffer and prior to supplying data to an output buffer.

A symbiont program which uses the deferred request return as a signal to release will never encounter the not ready return from the verifier on its peripheral unit dispatcher.

3. Errors

Tape synchronizer dispatchers will attempt to correct errors. Other dispatchers will return error indicators via an error return line to a program verifying a function specification on which an error has occurred. All error correction procedures for peripheral units other than tape are the responsibility of routines written to use these units. Tape synchronizer dispatcher error correction has been explained in Part A of this section.

4. Releasing

Symbiont programs are required to release, normally, upon finding the peripheral unit synchronizer they are using busy. This release is made to the busy peripheral unit synchronizer's dispatcher. The release is initiated when an entry to the request subroutine by a symbiont produces a deferred return, or when an entry to the verify subroutine by a symbiont produces a not ready return. Rather than looping through the verify subroutine unproductively for a time equivalent to the peripheral unit cycle and then receiving a ready reutrn, a symbiont executes entry to the release subroutine of the peripheral unit found busy. Control of the central processor is returned to the principal program and other symbionts until the next interrupt on the released synchronizer. Re-entry to the symbiont is then made by synchronizer control through the synchronizer dispatcher. At this time an entry to the verify subroutine by the symbiont will produce an immediate ready return, and the symbiont program can execute another processing cycle.

Each symbiont must provide a nine-word area, within itself, to hold the environment of any program interrupted by the symbiont's released synchronizer. The label of the first line of the nine-word hold area must be provided in all release calling sequences contained in the symbiont program. The contents of the arithmetic registers, the comparison indicators, and index registers 1, 2, and 3 (basic environment) of the program interrupted will automatically be stored in the symbiont's hold area when control returns to the symbiont from a release. This environment will automatically be restored to the proper registers and indicator when the symbiont subsequently releases. If a symbiont uses any registers or indicators other than those automatically saved, the symbiont must store their contents and status when control returns to the symbiont and must restore them prior to any subsequent release. A symbiont must also save its own environment before it releases since none of its registers or indicators will be saved. On return from a release, a symbiont must restore its cover index registers (those index registers which modify 10 bit addresses in instructions). Cover index registers may be pre-stored as constants when the symbiont is coded. Index register 3 will be restored automatically with respect to the symbiont at the time of return from a release. Therefore, the release-return coding should be covered by index register 3. On return from a release on a tape channel the following coding to restore index register 3 should be used:

```
SLJ     IOLX
 +      index register 3 value
. . .
```

IOLX should be equated to an absolute 10 bit address as determined from the executive routine listing (IOLX EQU 0461).

E. Dispatching - Card Reader

To maintain its rated speed, the card reader must be operated as a real-time device. Since the reader is not clutched, function specifications must be available without fail at each card reader synchronizer interrupt so that the position of cards at the stations within the reader may be accounted for.

On all other units, the function specifications are prepared by the using program and are communicated to the basic dispatchers in response to processing demands. This variable timing is insufficient to meet the card reader's real-time demand for new functions. The basic dispatcher for the card reader accordingly incorporates a buffering function as well.

A request for a card image does not supply the basic dispatcher with a function, but, instead, receives from it, when ready, the location of the internal buffer in which the current card image is stored. The verifier technique used to coordinate the activity of other synchronizers is not applicable to the card reader. However, since an internal buffer may or may not be ready when needed, ready and not ready returns are associated with the request subroutine on the card reader only.

The basic dispatcher for the card reader maintains six buffers. One is the active read-in area at a memory location whose address is a multiple of 64. Card images are moved as read to the other five buffers in rotation. Of the five remaining buffers, one is normally in the hands of the using program, and the other four must be available to receive cards already committed to the reader. A request for a card implies: (1) that the buffer containing the card previously requested is released and available to receive a new image, and (2) that four additional cards following the one requested are committed to the reader. This implies that four dummy cards should follow the end of a deck, and that when the reading mode is changed from translated to untranslated, after cards have been committed to the reader and vice versa, the first four requests in the new mode will produce images in the previous mode.

F. Timing and Space Requirements of the Basic I-O Routines

The following figures represent the timing and space requirements of the input-output interrupt routines for the five basic channels. Future modifications and refinements may change these specifications moderately, but the basic range is expected to remain fairly stable.

1. Card reader:

Size:	516 words (includes 240 words of buffers)
Interrupt:	1020 us when image present
	600 us when no image present
Request:	216 us ready
	444 us not ready and reader inactive
	168 us not ready and reader active

2. Punch:

Size:	164 words
Interrupt:	488 us
Request:	188 us deferred
	232 us accepted
Verify:	176 us ready
	152 us min. not ready
	184 us max. not ready
	168 us avrg. not ready
	152 us error

3. Printer:

Size:	146 words
Interrupt:	488 us
Request:	188 us deferred
	232 us accepted
Verify:	160 us ready
	140 us min. not ready
	152 us max. not ready
	146 us avrg. not ready
	152 us error

4. Tape write:

Size:	230 words
Interrupt:	492 us minimum (class 1 priority)
	788 us maximum (no activity)
Request:	180 us deferred
	212 us accepted
Verify:	384 us ready
	244 us not ready

5. Tape read:

Size:	325 words (includes routines used jointly with tape-write)
Interrupt:	556 us minimum (class 1 priority)
	848 us maximum (no activity)
Request:	192 us deferred
	224 us accepted

Verify: 348 us ready

200 us not ready

6. Release (all channels):

216 us

V. CONTINGENCY CONTROL

A. General

The contingency control routine is a most important function of the executive system inasmuch as it will provide the only means of operator control. The contingency control routine is designed to operate in parallel with other programs. One message at a time may be typed in or typed out on the console keyboard typewriter under control of the contingency control routine. When the typewriter is busy, additional message requests will have to be resubmitted until acceptance by the contingency control routine is possible. The contingency control routine does, however, have capability to communicate buffering information to other routines which may desire to buffer a large number of lines of information at a time.

The contingency control routine in itself provides only diagnostic operator control (listed in sections V. B-C). Functional operator control such as orders to space paper, start or stop symbionts, etc., are communicated by the keyboard through contingency control to the symbiont or principal program involved, or to other routines in the executive system. Interpretation of the message is then accomplished by the program receiving the message. The contingency control routine is synchronized so that diagnostic operator control reflects only the principal program.

B. Programmer Control

1. Typewriter Output

A program may type out a message on the typewriter in the following manner:

a. Load AR1 with the starting address of the message to be typed out in the form of an indirect address word using index register 1.

b. Cause an overflow by adding two appropriate constants together.

c. Follow the line in which overflow will occur by a line containing the octal constant 037777. This sequence which causes a contingency interrupt is recognized by the contingency control routine as a planned overflow. It is the only kind of planned overflow handled by the contingency control routine. If the typewriter is not busy the entire message is transcribed to the typewriter buffer area in the typewriter routine, typing is commenced and control is returned to the requesting program at the address of the second line following the 037777 constant. The remainder of the message is typed (one character at each contingency interrupt) by the contingency control routine concurrent with subsequent operation of the requesting program. A carriage return character as the final character of a message indicates the end of the message to the contingency control routine. If, however, the typewriter is busy, then control is returned to the requesting program at the address of the first line following the 037777 constant. In this case, the contingency control routine makes no attempt to type out the message. At this point the requesting program could postpone the request for a typeout or could loop back through the request routine. The following sample code could be used to type out a message:

```
LA    1,(LOC,1)          a.  load AR1 with starting
                             address of message to be
                             typed-out.  Index register 1
                             is used as a counter by the
                             contingency control routine
                             to scan the message.

LA    2,(077777777)      b.  force overflow
BA    2,(077777777)

+     037777             c.  octal constant

J     $-4                d.  message not accepted, then
                             loop through through re-
                             quest routine

. . . . . .              e.  message accepted.  This
                             line could be the start of a
                             test loop if an operator
                             typed-in is expected.
```

2. Program Acceptance of an Operator Type-in

In order to be able to accept operator type-ins, a program will be given a one character identification. Symbionts (programs other than the principal program) will be given the channel number of the peripheral unit they are using as the one character identification. Principal programs will be given any alphabetic character identification excluding those used for format codes and system control (see part C 5b of this section). This character identification and the starting address (exit line) of the program's subroutine for accepting operator type-ins will be punched into a parameter card for loading into table TAB2 of the contingency control routine.

The format of the one word entry into table TAB2 for each program given the capability of operator type-in acceptance will be as follows:

Bits	24-19	one character identification
Bits	15-1	starting address of type-in acceptance subroutine

An operator type-in may be accepted by a program through the use of the typewriter operator R as the first character typed-in after a keyboard request. The character typed-in following the R will be the one character identification followed by the typed-in message plus an extra leading blank. The contingency control routine will load the type-in into its typewriter line buffer one character at a time. The first character of the typewriter line buffer is left blank by the contingency control routine. Since an extra leading blank is typed-in by the operator, this assures that the message itself will start in the second word of the typewriter line buffer. The first word will contain blank, R, one character identification, and blank in that order. The typewriter line buffer area is capable of holding one printed teletype length line. Each typed-in character is accepted by interrupting the program which has control of the central processor. After accepting each typed-in character through interrupt the contingency control routine will return control to the interrupted program. When the contingency control routine finally receives a keyboard release, it scans table TAB2 for the appropriate one character identification. Upon finding the one character identification, the contingency control routine tests the contents of the location specified by the associated type-in acceptance subroutine starting address (exit line) for zero. If the contents of the location specified by the starting address is zero, then the contingency control routine executes an SLJ to the starting address of the program's type-in acceptance subroutine. At the time of this SLJ the contents of index register 2 will contain the beginning address of the typewriter line buffer. If the contents

of the location specified by the starting address is not zero, then the message is bypassed by the contingency control routine. This technique prevents information from being typed-in to a program which is not ready to accept such information which in effect prevents unwanted operator intervention. It should be especially noted that when and if the contingency control routine does an SLJ through a program's type-in acceptance subroutine the following condition exist:

(a) Interrupt is prevented

(b) The program's environment is not restored

Therefore the subroutine will be required to load its cover index registers, process the message, and exit by executing a J indirectly to the starting address (exit line) of the subroutine as quickly as possible. Preventing interrupt for any length of time may have devastating effects. The recommended usage of the program subroutine is move the typewriter line buffer contents into a program area, set the necessary program switches, and then exit. The contingency control routine will then restore interrupt and the interrupted program's environment and return control to the interrupted program. A program requiring an operator type-in will set the starting address (exit line) of its type-in acceptance subroutine to zero. It will then normally type-out a message requesting the type-in and loop in a test loop, which tests the contents of the location specified by the starting address (exit line) of the type-in acceptance subroutine, or an appropriate switch setting. Each time control is returned to the program's test loop after a contingency interrupt, the program will test either the contents of the location specified by the starting address (exit line) of its subroutine or an appropriate switch setting. If the contents of the location specified by the starting address is not zero or the appropriate switch is set, the typed-in message is available and the program can process. If one or the other condition is not met, the program continues looping in its test loop.

3. Unplanned Overflow or Illegal Operation Code

A contingency interrupt due to an unplanned overflow or an illegal operation code in a program will cause the contingency control routine to type-out the following message:

XXXXX CONTING. 77777Δ00Δ00Δ0000Y

where 77777 indicates in octal notation the address of the location following the line in which the contingency interrupt occurred. Y represents the indicator set. Following the typeout, the contingency control routine will loop in a stop loop:

J $

C. Initiate Operator Control - Keyboard

1. Load System and Reset Core

To load the executive system and reset core memory, depress the following keys in sequence:

a. REWIND

b. CLEAR

c. LOAD

d. STOP

e. RUN

This will reset the system and reload EXEC. It will re-establish canonical tape assignments and will automatically reload supervisory control. Depressing STOP will cause core memory outside of the EXEC to be set to SLJ ERROR. Any transfer by a program to a location outside of itself which contains SLJ ERROR will cause an immediate jump to an error routine in the EXEC.

2. Load System without Reset Core

To load the executive system and leave core memory outside of the EXEC intact, depress the following keys in sequence:

a. REWIND

b. CLEAR

c. LOAD

d. RUN

This will reload the EXEC only.

3. Load System and Load Binary Cards

To load the executive system and load binary cards through a card reader, depress the following keys in sequence:

a. REWIND

b. CLEAR

c. LOAD

d. REQUEST

e. STOP to reset core memory

f. RUN

In this case the on line relocatable loader is employed and conventions of this loader should be followed. The binary cards loaded can now be used as a simple self-contained program for non-standard routines, to make temporary system modifications, etc. For example, a cell in LODX could be overlayed so that control would transfer to the patch loader following the loading of each subsequent program segment from the system tape, facilitating checkout of very large multiple-pass programs.

D. Exec Operator Control-Keyboard

Operator control is provided by the following:

1. Contingency Stop

Depressing this key creates an interrupt which causes the contingency control routine to type-out the message:

*△77777

where 77777 indicates the address of the next location to be executed of the program interrupted. The contingency control routine then does a spin in a stop loop:

J $

2. Keyboard Request

Depressing this key will activate the typewriter to accept an operator type-in under control of the contingency control routine. If an error is made during type-in, the message may be retyped by depressing the request key again.

3. Keyboard Release

Depressing this key signals through interrupt the end of message type-in to the contingency control routine. It should be noted that while the contingency control routine is accepting a type in (keyboard is active), programs within the computer cannot type out.

4. Carriage Return

The carriage return key should not be used since carriage return is used as an internal control function (end of message type-out) by the contingency control routine.

5. Operation Control Characters

The first character typed-in on a message controls message interpretation. The characters and their interpretation are listed below:

a. C (Location Counter) (Control Counter)

displays on the console typewriter in octal notation the current value of interrupted program's location counter.

b. ↄβΔ77777 (Display)

displays on the console typewriter according to format β the contents of the location indicated in octal notation by the address 77777.

Format Codes are:

A	Δ-AAAA	Alphanumeric
C	Δ-17Δ37Δ77777	Complete Indirect Address Word in octal notation
D	Δ-999999	Decimal
F	Δ-17Δ37Δ37Δ1777	Field Select Word in octal notation
I	Δ-17Δ77Δ17Δ1777	Instruction in octal notation
O	Δ-77777777	Octal

c. EIΔ-17Δ77Δ17Δ1777 (Execute Instruction)

restores the interrupted program's environment and executes the instruction typed-in in octal notation.

d. G (GO ON)

permits continuance of a program stopped (looping) in a stop loop J $ by returning control to the location following the stop loop. Also breaks out of the contingency control routine stop loop and allows return to the interrupted program.

e. JΔΔ77777 (Jump)

causes control to be transferred to the location specified in octal notation by the address 77777.

f. Lβ Δ77777Δ. . . . . (Load)

loads into the location specified in octal notation by the address 77777 according to format β the typed-in information following the address 77777.

Format Codes are:

A	Δ-AAAA	Alphanumeric
C	Δ-17Δ37Δ77777	Complete Address Word in octal notation
D	Δ-999999	Decimal
F	Δ-17Δ37Δ37Δ1777	Field Select Word in octal notation
I	Δ-17Δ77Δ17Δ1777	Instruction in octal notation
O	Δ-77777777	Octal

g. SkΔ. . .

directs type-in message to symbiont on channel k.

h. MβΔ77777Δ77777 (Memory Print Out)

prints memory between the specified address limits in octal notation 77777Δ77777 according to the format β. If one or more lines of the memory print out contain the same word as the last word in the previous line in all locations being printed out, the lines are omitted and represented by a single line of periods. The following dump formats are available:

I	Instruction
A	Alphanumeric
D	Decimal
O	Octal
B	Both Octal and Instruction

i. RXΔ. . . . . (Request)

transfers control to routine X with the type-in message in the typewriter line buffer. Routine X is in the contingency control routine and is used to locate and load programs on the systems tape. For example, the type-in RXΔ ACCO locates and loads the assembler-compiler control program.

j. TΔΔ. . . . . (Transfer)

transfers control to the location specified in the contingency control communication location TCD. The location in TCD is normally the transfer address of the last program loaded from the system tape. The type-in message will be in the typewriter line buffer and may be used to communicate operating instructions to the program transferred to.

6. Invalid Characters

If the first character typed-in on a message is not a valid operation control character, the contingency control routine types the whole message out on the typewriter followed by the type-out XX. Control is then returned to the interrupted program. If the second character typed-in on a R message is not a valid format code or the affected starting address is not ready ($\neq 0$), the contingency control routine types out the whole message followed by the type-out YY. As before, control is then returned to the interrupted program.

7. Typewriter Page Ejection

It is not necessary to take the typewriter off-line to eject a page. Typewriter page ejection may be accomplished simply by depressing the keyboard (form advance) release. The form advance release may be depressed more than once if a longer ejection is desired.

E. Contingency Interrupt Interpretation and Typing Control

When a contingency interrupt occurs, the contingency routine tests the contingency indicators. Tests are made to determine if a keyboard release, a keyboard request, a contingency stop or a typed character interrupt occurred. If none of the indicators tested is set, then the interrupt is assumed to be due to an overflow or an illegal operation code. In any case the total environment of the interrupted program is saved and the previous contingency control routine environment is restored.

1. Keyboard Request

If the contingency interrupt is due to a keyboard request, the contingency control routine determines if a message is now in the process of being typed out. When the contingency

control routine is in the process of typing out a message, it sets a switch within itself and postpones the keyboard request until the completion of the type-out. After the message type-out is completed or the typewriter has been found to be inactive, the contingency control routine will do the following:

a) Clear out the typewriter line buffer (zeroize the core area used).

b) Return the typwriter carriage by outputing a carriage return character to the typewriter.

c) Go into a typewriter input made of operation (active keyboard).

2. Keyboard Release

If the contingency interrupt is due to a keyboard release, the contingency control routine assumes that the last character of a previously requested message has been typed-in. Contingency control will determine whether at this point I-O interrupt has been prevented or whether control is now within a symbiont. If not, the contingency control routine will transfer control to its message interpretation routine. If so, it will set appropriate switches in I-O synchronizer control so as to regain control as soon as neither of the above conditions prevail. Upon regaining control, it will cause a dummy contingency interrupt and will then transfer control to its message interpretation routine.

3. Typewriter Interrupt

If the contingency interrupt is due to a character type-in, the contingency control routine accepts the character, stacks the character in the typewriter line buffer, and issues an echoing type-out of the character on the typewriter. If the contingency interrupt is due to a character being typed-out, the contingency control routine examines the character to determine if the character is an echo of a type-in or a carriage return. The

contingency control routine will reactivate the keyboard if the character type-out is an echo of a type-in. A carriage return character indicates end of message type-out to the contingency control routine. If the contingency interrupt is caused by any character other than an echo character or a carriage return being typed-out, the contingency control routine outputs the next character of the typewriter message submitted by a program.

4. Contingency Stop

If the contingency interrupt is due to a contingency stop then the contingency control routine transfers control to its contingency stop routine. The contingency stop routine functions in a manner similar to the keyboard request routine except that an * is typed out and switches are set to simulate an immediate keyboard release. This will cause a blank image except for the first character and a consequent entry into a J $ loop awaiting operator intervention. This loop is known as the stop loop.

5. Overflow or Illegal Operation Interrupt

If the contingency interrupt is due to an overflow or illegal operation code, the contingency control routine examines the location following the location that caused the interrupt. If the following location contains +037777, control is transferred to the requested type-out routine. In all other cases, the interrupt is considered an error; the contingency control routine types out an error message indicating the address of the error location and then loops in its stop loop J $.

6. Typewriter Routine Exiting

The last action of the contingency control routine before exiting to the interrupted program is to reset the contingency indicator that initiated the entry to the contingency control routine. In the case of message type-in to a symbiont, the contingency indicator is not reset when the control is passed to the symbiont. The symbiont tests the indicator and resets it.

7. Register Save and Restoration

Upon entry through interrupt, the contingency control routine saves the contents of all index registers and arithmetic registers, the settings of the comparison indicators, and the contents of cell 22 (location counter contents at time of interrupt). Upon exiting to the interrupted program, the contingency control routine restores these registers and settings. During the course of a type-out or type-in, the contingency control routine depends upon the preservation of the arithmetic registers and the first five index registers between interrupts. For this reason, each time the contingency control routine exits to the interrupted program, these registers are saved and upon interrupt these registers are restored by the contingency control routine. The locations at which the registers and indicators are saved can be found from an assembler listing of the contingency control routine.

F. Typed-in Message Interpretation and Control

The basic message interpretation performed by the contingency control routine is on the first two characters in each typed-in message. The first character is defined as the principal operation and the second character is defined as the operation modifier. The interpretation of each of these two characters is determined by a table look up. The interpretation of the operation modifier is primarily on the basis of the principal operation. However, operation modifiers are shared among all the principal operations so that it is not possible for two different principal operations to have different operation modifiers defined by the same character. The principal operation entry in the operation table consists of one word. The sign bit of this word indicates whether this operation is to be performed within (plus sign) or outside (minus sign) of the contingency control routine environment. Bits 24-19 specify the principal operation character and are used as the basis for the table look up.

Bit 17 indicates whether there will and should be a five-digit octal address as part of the typed-in message to be converted to binary. Bit 16 indicates whether there will and should be a typed-in word to be converted to binary. If the designated operation is to be performed without the contingency control environment, the operation is placed within the contingency control routine and is executed with the interrupted program's environment restored, e.g. EI (see part C 5c of this section).

VI. PROCESSOR ERROR CONTROL

A. General

Processor error interrupts will be handled by a processor error routine in EXEC. A type-out of the following format will be initiated by the processor error routine:

uuuuuΔPROCΔERRΔvvvvvΔOOΔOOΔyyyyy

where

uuuuu = location in processor error routine

vvvvv = location of interrupt (contents of 020)

yyyyy = processor error indicators

After initiating the type-out the processor error routine will go into a J $ loop. Memory dumps and other diagnostics must be initiated by the operator through the keyboard if possible.

B. Processor Error Indicators

00001	Instruction access
00002	Operand access
00003	UNISERVO III Basic Write Access
00004	UNISERVO III Basic Read Access
00005	General Purpose Channel 1 Access
00006	General Purpose Channel 2 Access
00007	General Purpose Channel 3 Access
00010	General Purpose Channel 4 Access
00011	General Purpose Channel 5 Access
00012	General Purpose Channel 6 Access
00013	General Purpose Channel 7 Access
00014	General Purpose Channel 8 Access
00015	UNISERVO II Access
00016	UNISERVO III Additional Write Access
00017	UNISERVO III Additional Read Access
00020	Modulo 3 check on Instruction
00040	Modulo 3 check on Operand
00100	Adder Error Check

BOSS

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

U3521