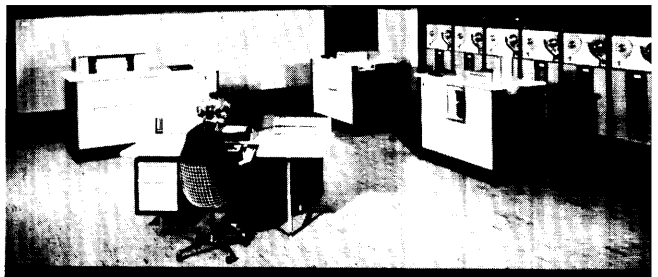
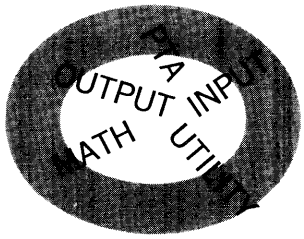


UNIVAC® III

TECHNICAL BULLETIN

SCOPE

Reference Manual



REPORT

UNIVAC III SUPPORT

INDEX

INDEX

INDEX

GENERAL INTRODUCTION

SUPPORT III is a dynamic expanding library of routines and subroutines designed to facilitate the efficient utilization of UNIVAC III. The SUPPORT III library falls into six categories: Input, Output, Utility, Program Testing Aids, Mathematical, and Miscellaneous. All of the routines in the library are integrated with other UNIVAC III programs such as ALMOST, UTMOST, COBOL, BOSS III, FORTRAN IV for UNIVAC III and SORT III.

As other routines become available, they will be incorporated in this manual.

ON-LINE BINARY CARD LOADER

A. Purpose:

Provide a simple, compact binary card absolute loader for the on-line reader. This loader is used to load ALMOST assembly output.

B. Method:

Cards are loaded with interrupt prevented into the locations specified on each card. Low-speed card reading of 175 cards per minute is employed.

C. Restrictions:

Card reader will only handle up to 24 words per card of object information.

D. Memory Space:

Program uses the first 200 octal words of storage, including the card image which is located at 100 octal.

E. Input Card Form:

Input card form is standard UIII binary.

F. Operating Procedure:

Hit general clear. Load one card and hit run.

BINARY CARD FORMATS

The output of an ALMOST assembly is a deck of 80 column punched cards. The cards are punched in binary, for loading into the UNIVAC III using the binary loader in untranslated mode.

In an 80 column untranslated card a word is 4 columns wide 6 rows long (4x6). Word 1 occupies columns 1-4, rows 12-3; Word 2, columns 1-4, rows 4-9; Word 3, columns 5-8, rows 12-3; Word 4, columns 4-9; etc. In this fashion the upper half of a card contains the odd numbered words (1, 3, 5, 7, ... 39), the lower half of the card contains the even numbered words (2, 4, 6, ... 40).

ALMOST produces two types of cards: Data Cards (the data which the assembler produces--instructions, constants, etc.) and a Transfer Card. The Data Cards need not be loaded in any special order since each card carries the address of the first data word in the first word on the card. (See Data Card Format following.)

A Data card may contain up to and including 28 words.

The first word contains the address where the first data word will go in memory.

Word 2 is of special format to cause the card to have even parity.

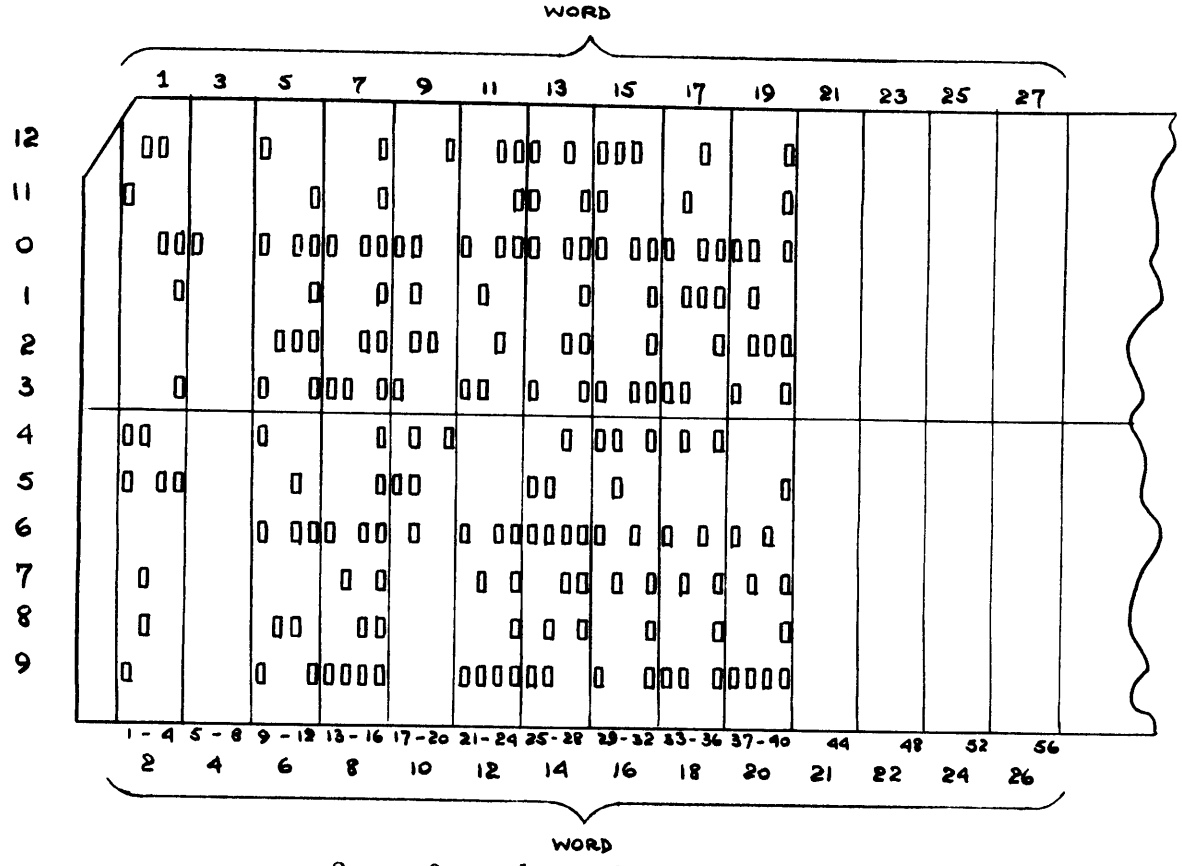
Word 3 contains the sign bits for the data words; a blank for +, a punch for -.

Word 4 is always blank.

Words 5-28 contain the data words.

A Transfer Card contains the starting address of a program (where control is transferred after the program has been loaded into memory) and the index registers specified in the ALMOST USE statement with the contents they should contain. The binary loader loads the program into memory, loads the index registers specified in the transfer card with the amounts given and transfers control to the address specified in word 1 of the Transfer Card.

DATA CARD



WORD 1

2	0	1	0	5	5	4
0 0 0 0 1 0	0 0 0	0 0 1	0 0 0 1 0 1	1 0 1 1 0 0		
3 2 1 0 11 12	3 2 1	0 11 12	3 2 1 0 11 12	3 2 1 0 11 12		

Col. 1 Col. 2 Col. 3 Col. 4

Bits 1 - 15 Address of First Data Word

10554_8

(see sample output listing)

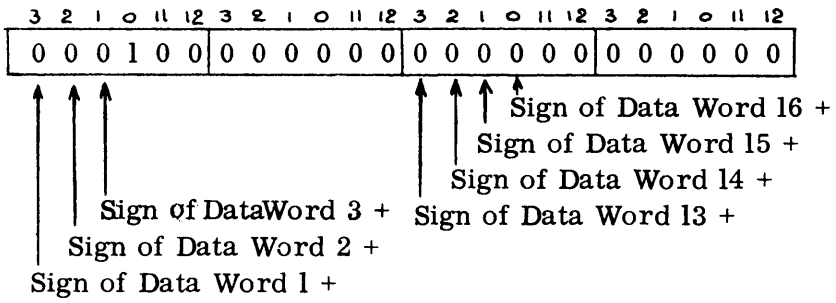
Bits 16 - 24

Count of number of Data Words ($C \leq 24$)

WORD 2 'Exclusive Or' of all other words on the card (causing the card to have even parity)

Col. 1	Col. 2	Col. 3	Col. 4
1 0 0 0 1 1	0 1 1 0 0 1	0 0 0 0 1 0	0 0 0 0 1 0
9 8 7 6 5 4	9 8 7 6 5 4	9 8 7 6 5 4	9 8 7 6 5 4

WORD 3 Col. 5 Col. 6 Col. 7 Col. 8



Signs of Data Words:

bit 24 = Sign of Data Word 1
bit 23 = Sign of Data Word 2

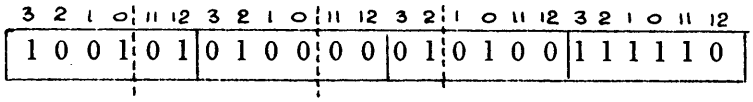
.
.
.
.

etc.

WORD 4 Always Blank

WORDS 5 - 28 The Data Words

Col. 9 Col. 10 Col. 11 Col. 12



On output listing this is instruction word

Bits

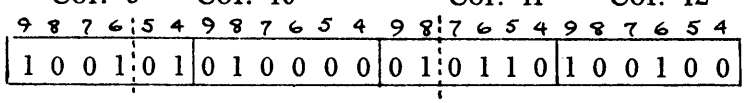
24 - 21 IR 9 = 11_8

20 - 15 OP = 24_8 = BA

14 - 11 AR = 01_8 = arithmetic register 4

10 - 1 10 bit address = 0476_8

WORD 6 Col. 9 Col. 10 Col. 11 Col. 12



This is an instruction word

Bits

24 - 21 IR 9 = 11

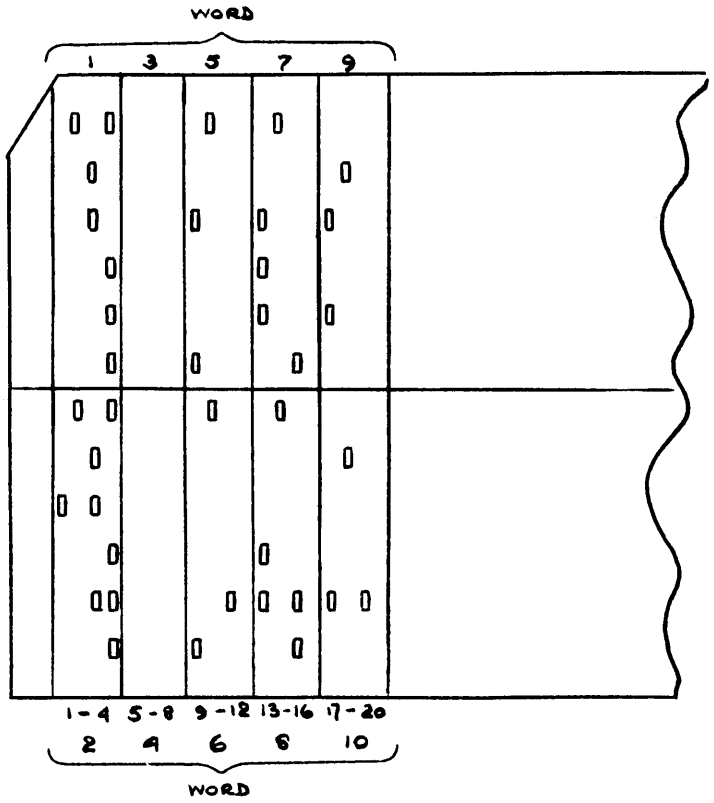
20 - 15 OP $= 24_8$ = BA

14 - 11 AR $= 01_8$ = arithmetic register 4

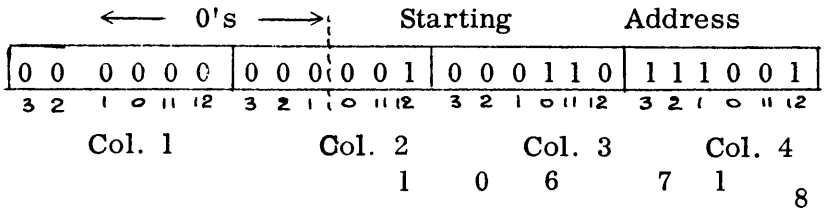
10 - 1 10 bit address = 0644_8

etc.

TRANSFER CARD

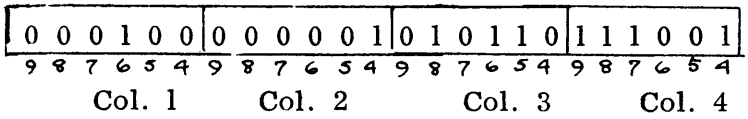


WORD 1



WORD 2

Exclusive or (Logical Difference) of all other words on the card (causing the card to have even parity).



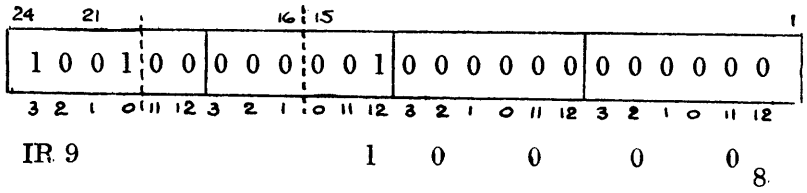
WORDS 3 and 4 Blanks

WORDS 5 - Up

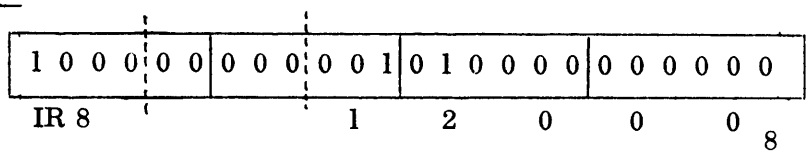
Index Load Words:

Bits 24 - 21 Index to be loaded
20 - 16 Zeros
15 - 1 Amount to be loaded

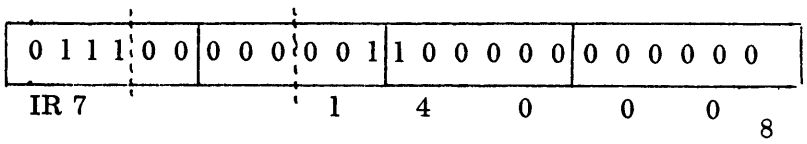
WORD 5



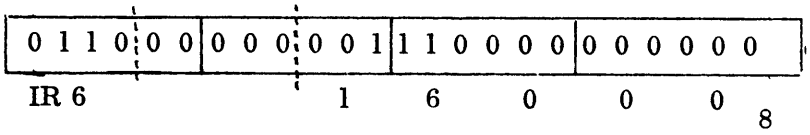
WORD 6 - 10



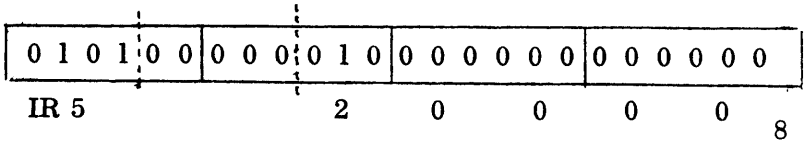
WORD 7



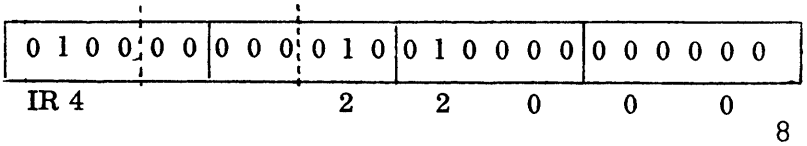
WORD 8



WORD 9



WORD 10



COMPOSITE CARD LOADER

A. Purpose:

Provide a loader, moderate in scope, which will load absolute instructions or data in a variety of octal, decimal, and alphabetic formats, using the on-line 80-column card reader.

B. Method:

Card formats are recognized by absence of punching in characteristic columns. A reading speed of 350 cards per minute is employed.

C. Restrictions:

Blind determination of card form is made and no check is made by the loader as to the propriety of the contents of the card. Requires blank card at end of deck if no cards follow.

D. Mapping:

Current version of the program is at 1400 - 1677 octal. Card images are at 100_8 and the read subroutine is at 1716 octal.

E. Input Card Forms:

(See end pages Ch. 5.)

F. Operating Procedure:

If program is used to make patches to a standard absolute binary deck, then the transfer card from the absolute deck should be removed and the composite loader substituted in its place. Behind this should be placed the composite cards to be loaded and the index load instructions in composite card format, followed by a composite transfer card.

G1. General Formats

Let 1, 3, 7, and 9 represent digits less than or equal to the written number (e.g., 3 may represent 0, 1, 2, or 3). Let A represent any character. Following are the six word formats accepted by the loader:

	0									1									2	
Col.#	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	
	7	7	7	7	7		-	1	7		7	7		1	7		1	7	7	7	instruction format
	7	7	7	7	7		-	1	7		3	7	-	3	7		1	7	7	7	field select word format
	7	7	7	7	7		-	1	7						3	7	7	7	7	7	indirect address word format (col. 15 need not be punched)
	7	7	7	7	7		-	7	7	7	7	7	7	7							octal data format
	7	7	7	7	7		-	9	9	9	9	9	9								decimal data format
	7	7	7	7	7		-	A	A	A	A										alphanumeric data format

2. Description of Format

Instruction Word

1 - 5	location
7	sign
8 - 9	index (4 bits)
11 - 12	operation (6 bits)
14 - 15	A field (4 bits)
17 - 20	M field (10 bits)

Indirect Address Word

1 - 5	location
7	sign
8 - 9	index (4 bits)
16 - 20	address (15 bits into bits 1 - 15)
	bits 16 - 20 are cleared

2. Description of Format (Cont'd.)

Field Select Word	
1 - 5	location
7	sign
8 - 9	index (4 bits)
11 - 12	left bit in octal excess three (5 bits)
13	-
14 - 15	right bit in octal excess three (5 bits)
17 - 20	M field (10 bits)

Octal Word	
1 - 5	location
7	sign
8 - 15	octal representation of word

Decimal Word	
1 - 5	location
7	sign
8 - 13	decimal representations of word

Character Word	
1 - 5	location
7	sign
8 - 11	character representation of word

3. Index Loading and Transfer Cards

a. Index Card

If the location (columns 1-5) is greater than zero and less than 00020, the data is loaded in cell zero and the corresponding index register is loaded from cell zero.

b. Transfer Card

If the location (Columns 1-5) is zero or blank, the data is loaded in cell zero and program control is transferred by means of a J * 0 instruction (Indirect Addressing).

CARD READER ROUTINE 1.0003

A. Purpose:

To maintain a flow of cards through the Card Reader at its rated speed of 700 cards per minute, making the images available to the program as required in either translated or untranslated format.

B. Method:

To maintain its rated speed, the Card Reader is operated as a real-time device. Since the reader is not clutched, function specifications are made available at each interrupt so that the position of cards at the stations within the reader will be accounted for by the reader program. The basic dispatcher for the card reader (synchronize control routine for the reader) maintains six buffer areas within the memory of the computer. One is the active read-in area and of the remaining five, one is used to hold the card currently available to the worker program, and the other four available for the cards already committed to the reader. A "request" for a card implies that the buffer area containing the card image previously requested is now released and available for a new image and that four cards have been committed to the reader following the one currently being requested.

C. Restrictions:

The commitment of four cards implies that four blank cards should follow the end of a card deck and also that, when a change from translated to untranslated mode or vice versa is made, the succeeding four card images will be made available in the previous mode. All ARs and IRs 1 and 2 must be preserved by the worker program if desired, or made available to the reader routine at the time control is transferred to the Card Routine from the worker program.

D. Programming Procedures:

1. Card Request: To request a card image, the programmer should provide the following packet of coding:

```
n       SLJ   *CDRQ    Execute Card Request
n+1     (         )    Starting Location of Card Image provided
                       on ready return
n+2                    Not Ready Return Exit
n+3                    Ready Return Exit
```

Location "n" contains an SLJ instruction indirectly addressing the Card Request subroutine. If an image is not available, control will be returned to location "n+2", and the programmer provision should be made in this location for this eventuality. If an image is available, control will be returned to location "n+3" and the base address of the card image will be provided in location "n+1".

2. Release: To release control of the processor to an alternate program when an image is not available, the programmer should provide the following packet of coding:

```
p       SLJ       *CDRL
p+1     (Return entrance)
```

When control is transferred to location "p", the Release exit, the processor will be released to another program. When an image is available, control will be returned to location "p+1" for further card processing. An entry to the Request subroutine on Return will produce an immediate "Ready" exit.

3. Translation Mode: To specify an untranslated image, a zero should be placed in bit 18 of CDQF (Card Request Communication word) in the input/output communication region. When the mode is changed, four images will be supplied in the previous mode.

4. Stacker Selection: Cards are normally directed to stackers 1 and 2, alternating every 900 cards. Cards incurring an error or fault are directed to stacker 0.

5. Error Recovery: When an error is detected, the card is sent to stacker 0. The operator notifies the routine by a type-in when and whether to retry reading the card. The basic dispatcher will return to the interrupted environment while waiting for the type-in.

NOTE 1: The Card Reader routine utilizes all arithmetic registers and Index Registers 1 and 2. These must be preserved by the programmer if it is necessary to retain their contents.

NOTE 2: In utilizing the Card Reader Routine with the ALMOST assembly system, standard EQU cards should be placed ahead of the ALMOST symbolic deck which is to be assembled. In this manner, the labels for the Card Reader Routine will be provided.

INTERMEDIATE TAPE HANDLING ROUTINE

A. PURPOSE

To provide a set of tape handling routines at a block level of communication.

B. METHOD

1. Structure of the Intermediate Tape Handling Routines

The Intermediate Tape Handling Routines permit the manipulation of UNIVAC III tapes on the block level. This permits the following functions: Block Read Forward, Block Read Backward, Scatter Read Forward, Scatter Read Backward, Write, Rewind after Reading, Rewind after Writing and Position Tape (skip a given number of blocks with no data transmission).

The checking for labels, end-of-file sentinels, end-of-reel sentinels and by-pass sentinels is left to the user as is the implementation of item advance.

All of the subroutines which perform the above mentioned functions specify a Symbolic Tape Unit Reference. This Symbolic Tape Unit Reference is the label or numeric address of a location which contains in bits 24-21 the logical tape unit number. When BOSS III is loaded, locations octal 0200 through octal 0217 contain, in order, the logical tape unit numbers 0 through 15. The user may equate any label he desires with these locations.

The levels of the Intermediate Tape Handling System are:

a. Intermediate Level Tape Handling Routines

b. Basic Request and Verify Routines

c. Basic Interrupt Dispatchers

The use of the Intermediate Level Tape Handling Subroutines is discussed below. The basic request and verify routine and the basic interrupt dispatchers are described in the BOSS III manual.

2. Tape Formats

This level of tape input-output communication can accept any tape formats. The processing of the contents of a tape is completely at the discretion of the user as described above. It is recommended, however, that the tape formats as described in the section on Tape Input-Output Item Handling in this manual be used for processing at the block level.

3. General Procedures

If the user of the Intermediate Tape Handling Routine wishes to perform simple buffering, two alternating areas may be employed for this purpose by specifying a verification cycle of 2. This causes the return from a request for a tape operation, with a specified reserve word R, to be delayed until the previous request with the same reserve word specified has been completed.

Matching requests with returns is accomplished by reference to the reserve word R within the Intermediate Level Tape Handling Routine. The matching is done by storing the function specification word in the reserve word R before transmitting it to the basic dispatcher. A unique reserve word must be provided for each tape unit referred to by the user at the intermediate level. Each reserve word must have an initial value of zero. Swapping of tapes and of input-output areas may be accomplished either by modifying the calling sequence or by providing different calling sequences.

If the user wishes to use a demand read method of processing tape, waiting until each tape action is completed before returning to the calling program, a verification cycle of 1 should be used. No reserve word R is required in this case.

A verification cycle of 0 will cause the specified tape action to be initiated. Control will then be immediately given back to the calling program which must determine for itself when completion of the tape action has occurred by direct communication with the appropriate basic tape dispatcher. This is accomplished by executing a verify calling sequence as described in the BOSS III manual.

Verification cycles greater than 2 will operate as though they were 2.

C. CODING PROCEDURES

1. Block Read Forward

n	SLJ	RDBF
n+1	+	Symbolic Tape Unit Reference
n+2	+	Address of first word of read-in area
n+3	+	R, V
n+4	...	Return Point

V is the code for the verification cycle desired: 0, 1 or 2. R is the address of the Reserve word if V is 2. Otherwise R should ~~be zero~~.

2. Block Read Backward

n	SLJ	RDBB
n+1	+	Symbolic Tape Unit Reference
n+2	+	Address of last word of read-in area
n+3	+	R, V
n+4	...	Return Point

3. Scatter Read Forward

n	SLJ	RDSF
n+1	+	Symbolic Tape Unit Reference
n+2	+	Address of first word of SCAT control list
n+3	+	R, V
n+4	...	Return Point

4. Scatter Read Backward

n	SLJ	RDSB
n+1	+	Symbolic Tape Unit Reference
n+2	+	Address of last word of SCAT control list
n+3	+	R, V
n+4	...	Return Point

5. Write (Gather or Block)

n	SLJ	WRIT
n+1	+	Symbolic Tape Unit Reference
n+2	+	Address of first word of write-out area or address of first word of SCAT control list
n+3	+	Output block word count (Zero for gather write)
n+4	+	R, V
n+5	...	End-of-tape Return Point
n+6	...	Normal Return Point

A block write will be simulated if a word count is specified. This word count may not exceed 4096. A word count of zero indicates a gather write is desired. In this case the third word of the calling sequence is then assumed to be the address of the first word of a SCAT control list provided by the user. The end-of-tape return is used to indicate that the end of tape window on tape was reached on the specified tape unit during the previous tape function. Verification cycles and the use of a reserve word are used as described above for tape reading.

6. Rewind after Reading

n	SLJ	RDRW
n+1	±	Symbolic Tape Unit Reference
n+2	...	Return Point

Normally, return to the calling program will not be made until the rewind has been initiated by the specified tape unit. The return will be made immediately after the request has been made without waiting to verify initiation if a 1-bit is placed in position 16 of the second word in the calling sequence. If the second word in the calling sequence, the symbolic tape unit reference line, is negative, the tape will be rewound with interlock. If it is positive, the tape will be rewound without interlock.

7. Rewind after Writing

n	SLJ	WRRW
n+1	±	Symbolic Tape Unit Reference
n+2	...	Return Point

8. Position Tape

n	SLJ	TPOS
n+1	+	Symbolic Tape Unit Reference
n+2	±	Number of Blocks to be Skipped
n+3	...	Return Point

The subroutine skips the number of blocks indicated. A plus sign on the third word in the calling sequence, the number of blocks to be skipped, indicates skipping in a forward direction; a minus sign on this word indicates skipping in a backward direction.

TAPE INPUT-OUTPUT ITEM HANDLING

A. Purpose

To provide a set of tape input-output item handling routines.

B. Method

1. Structure of the Item Handling Subroutines

File description tables constitute the highest logical level within the tape Input-Output system. Entries in this table are either defined as constants by the user or compiled from given paramenters by a special subroutine. These tables are interpreted and the information in the files they represent is processed by a group of subroutines which perform the functions customarily associated with item handling operation: open, close, read, write and with the UNIVAC III system, write-read. The item handling operators in turn communicate to lower-level routines which act as a file dispatcher. The file dispatcher maintains a queue of requests generated by the item handling operators and coordinates these requests with the request-and-verify mechanism of the basic interrupt system. This coordination is accomplished through the intermediate level (block handling) tape input-output package which consists of the following block handling functions: read, write, overwrite, position and rewind. The levels of the item handling input-output system are thus seen to be:

a. File description table entries
b. Item handling operators
c. File dispatcher subroutines
d. Intermediate level tape handling subroutines
e. Basic request and verify routines
f. Basic interrupt dispatchers

The file description table entries and the item handling operators are discussed below. The use of the intermediate level tape handling subroutines, in which the user must provide his own item advance routine, is described in another section of this manual. The basic request and verify routines and the basic interrupt dispatchers are described in the BOSS III manual. (Section IV, Synchronizer Control).

2. Tape Formats: Labels, Data Blocks, and Sentinels

a. General

The input-output file structure produces and accepts tapes whose format follows the conventions described below. Tapes produced by the Item Handling System will contain the standard labels, sentinels and flags described below. Tapes read by the Item Handling System should conform to the standard format, unless the block option is exercised. If an input tape contains the proper flags and sentinels, the standard file system will process items of any fixed length and blocks containing a variable number of items provided that the specified maxima for item length and number of items per block are not exceeded.

b. Label Block

(1) Label Block Processing

If a data tape is labeled, the first block must be the label block. The presence of an address as one of the entries in the file description table (which is described below) indicates whether or not the file is labeled. If such an address is present, the first block of each reel of an input file will be read with a block-read tape order into the last twelve words of the file description table for that file, and a subroutine linkage will be made to the address specified. The subroutine located at the address given is assumed to be either a standard or special label-checking program which will verify the contents of the label just read.

For output files, the subroutine linkage will be made at the beginning of each reel, and the last twelve words of the file description table for that file will be written as a block upon return from the output label subroutine.

If the file is not labeled, a zero address is entered in the file description table as the location of the label checking subroutine. The first block on each reel of the file is then assumed to be a data block in standard format as described below.

If input labels are present but are not to be checked, a label

subroutine must still be provided. It can simply return control to the input-output routine without processing or checking the label.

(2) Label Block Layout

Word	Content	Information
0	-00000000	Label flag
1	AAAA	First four alphanumeric characters of file label
2	ddmmyy	Date: day, month and year in decimal characters
3	00rrr	File reel number in decimal characters
4		These words are available for use by the individual installation
5		
6		
7		
8		
9		
10	AAAA	Last four alphanumeric characters of file label
11	-00000000	Label flag

c. Data Blocks

(1) General

Data Blocks consists of one or more logical items of fixed length and two data descriptor words, one at each end of the block. The data block and data descriptor word

formats are shown and described below.

(2) Data Block Layout

Segment separators occur on tape between the initial data descriptor word and the first item of the block, between items within the block, and between the last item and the terminal data descriptor word. When writing, the data descriptor words are automatically prepared by the file dispatcher. When reading, the first data descriptor word encountered (depending on the direction the tape is being read) is placed in the item descriptor word position of the first and last item buffer area.

The data descriptor words are principally used as position markers for restart purposes. It should be noted that if the block mode is being used, the block will consist of two data descriptor words and the data area, which is considered as one item by the item processing routine. In this case there will be only two segment separators on tape for this block. If a block mode of operation is desired, it is recommended that the intermediate tape handling routines be utilized. They are described in another section of the manual.

(a) Data Descriptor Words

Data descriptor words consist of a one word marker at each end of a block. This marker is composed of two parts: the channel increment and the block number. The channel increment is located in the upper half of the data descriptor word and will be the number of items in this block in the case of a scatter read-gather write system or the number of words in the block in the case of a block-read block-write system.

The block number is in the lower half of the marker word and will be the true block number for this reel, modulo 4096. Both of these entries in the data descriptor word will be expressed in pure binary, 12 bits in each entry. The sign of the data descriptor word is positive.

(b) Items

The data area within the block may consist of one or more items. The number of items per block and the item size must not exceed the maximum size as stated in the file description table. The item size also must not exceed 511 words. The output produced by this system will consist of fixed size items and fixed size blocks. An exception in block size may occur preceding sentinel blocks or bypass blocks, where short blocks may occur.

(c) Item Chain Words and Item Descriptor Words

Preceding each item area are two words, the item chain word and the item descriptor word. The chain word contains the address of the next available input or output item area in the buffer pool. The item descriptor word is used to control and analyze tape format. The data descriptor word for a block is placed in the item descriptor word of the first and last item area.

The item descriptor words in the other item areas used for this block will not be used. Under normal usage, the user does not need to make reference to the item chain word or the item descriptor word. The item address given the user by this system will be the address of the actual data item. It should be noted that neither item descriptor words nor item chain words appear on tape.

(3) Data Block Processing

(a) Item Handling

When a request for an input or output item advance has been honored, the starting address of the next item buffer area will be found in the first location of the file description table associated with that file. The user can then load an index register with this address or can access the data indirectly using the label of the file description table as the operand.

When an FRD (read) operator is given, the current item area will be released back to the pool. When

an FRW (write) operator is given, the item whose starting address is in the first location of the file description table associated with that file is placed on the output list for that file. The item area will be released when the tape write operation involving that item area is completed. When a FWRD (write-read) operation is given, the item whose starting address is given in the first location of the input file description table specified, is placed on the output list for the specified output file. The specified input file will then be advanced as described above. When a write tape operation is completed, the item areas involved will be returned to the pool for that file.

(b) Block Option

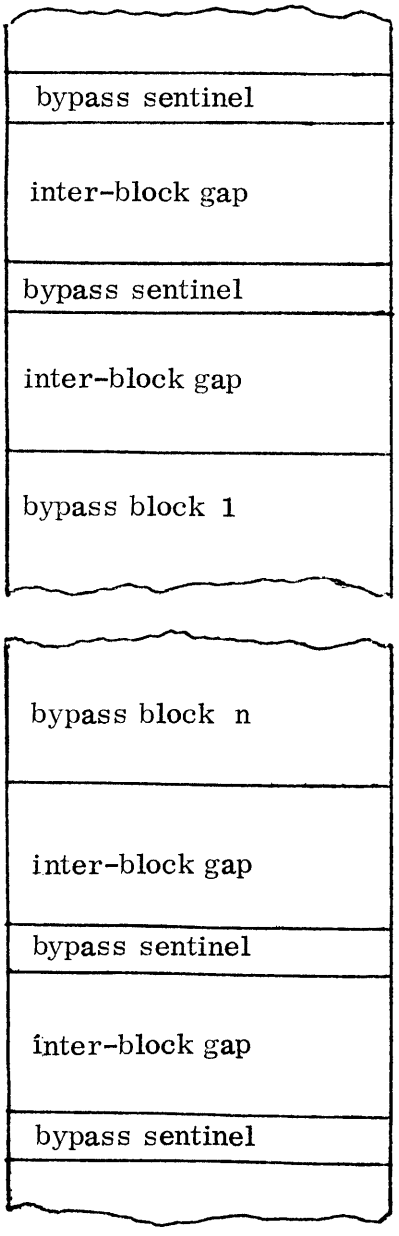
An input tape file, which does not contain standard labels, flags and sentinels, may be read by exercising the block option. Such a file should use a pool in which the maximum block size is given as the size of the item. The blocks on this file will be processed in the block mode, and the use of read and write operators will provide the user with the address of the first word of each block. Analysis of the contents of the block then becomes the responsibility of the user. Sentinel checking, label checking and item advance must be provided by the user when using the block option.

d. Sentinels

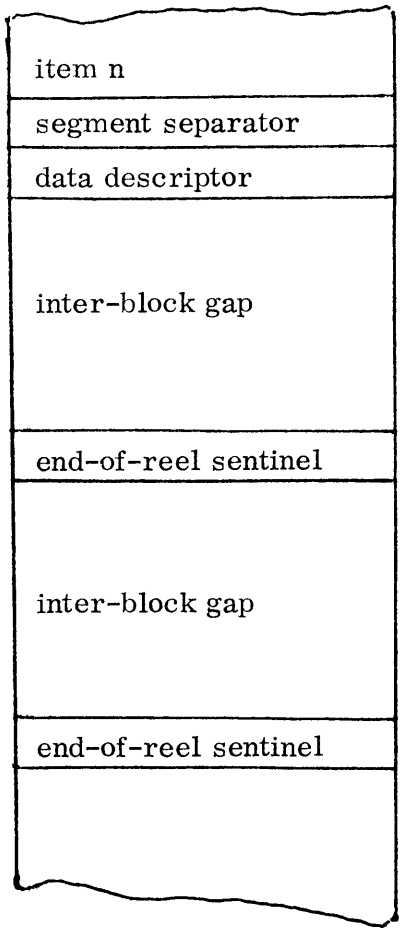
Three types of sentinels exist in this system. They are the end-of-file sentinel, the end-of-reel sentinel and the bypass sentinel. The end-of-file and the end-of-reel sentinels each consist of a one word block. The high order 2 bits and sign of this one word are a special flag signifying what kind of a sentinel it is. The low order 22 bits of this word will be a block count of all of the label blocks, data blocks and sentinel blocks in this reel to this point including the sentinel block.

Bypass sentinels are used to indicate that a portion of the data on a tape does not pertain to the file which contains that tape. Two bypass sentinels will appear before and after the block or blocks containing this extraneous information. These blocks are ignored

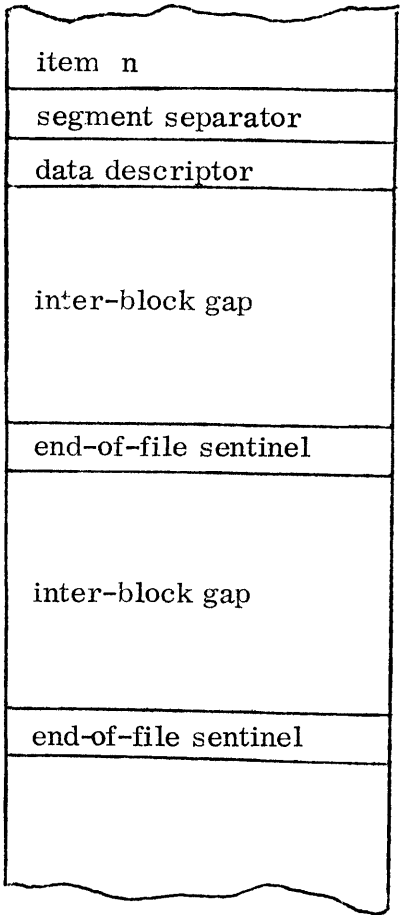
when encountered by the item handling routine and the information contained in them will not be given to the user. These bypassed blocks are generally used for memory dumps.



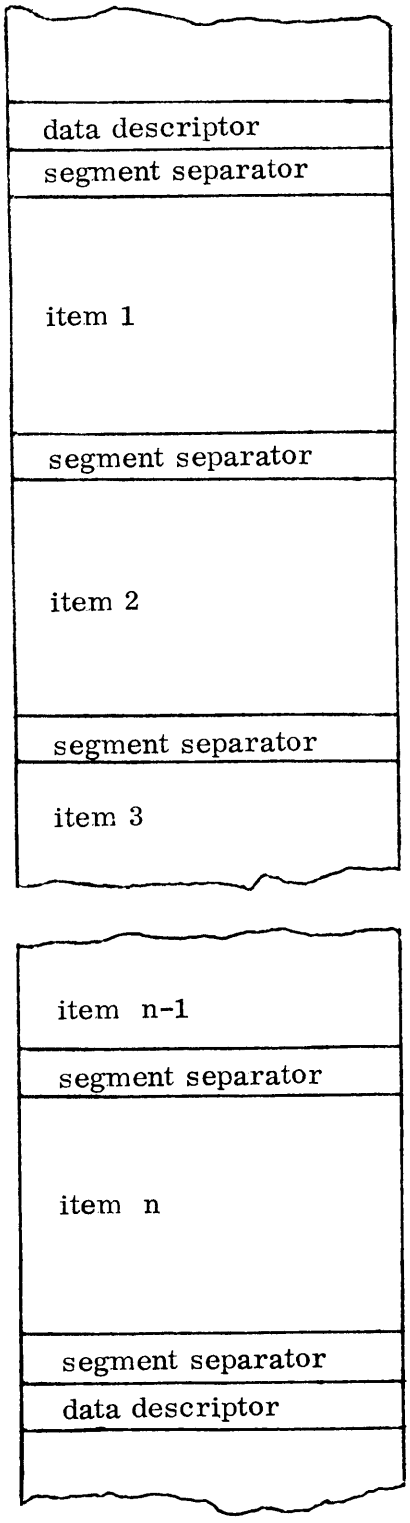
This diagram shows how bypass sentinels are used to indicate that a portion of a tape contains information which does not pertain to the file which contains that tape.



This diagram shows the end of a block followed by two end-of-reel sentinels.

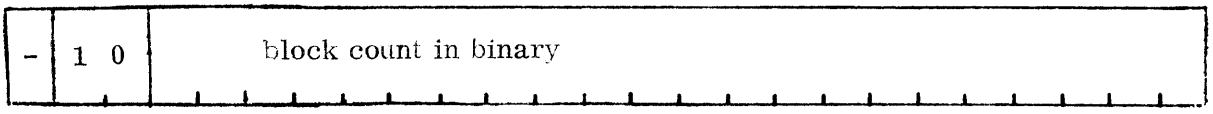


This diagram shows the end of a block followed by two end-of-file sentinels.

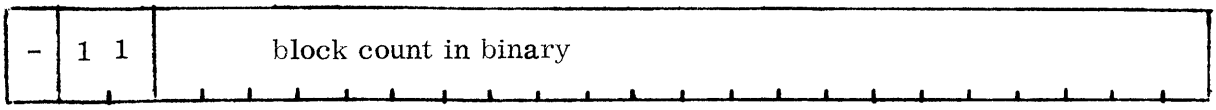


This diagram shows a data block containing n items with associated data descriptors and segment separators.

data descriptor



end-of-reel sentinel



end-of-file sentinel

bypass sentinel

3. File Description Table

a. General

The characteristics of each file to be processed by a program utilizing the Item Handling Subroutines must be provided by the user. These characteristics are described in a table called the File Description Table. This table consists of 28 words in the case of a labeled file and 16 words in the case of an unlabeled file. The entries in this table marked by an asterisk must be supplied by the user. The other entries are used internally by the Item Handling Subroutine. The values of the entries supplied by the user are dynamically alterable and may be changed whenever the file is not open. The label of the first entry in the table will be the "file label" as used in the calling sequences described in the section on file operators.

b. File Description Table

File Label	
	current item location
1	maximum item length*
2	maximum number of items per block*
3	pool control label*
4	location of current item chain word
5	location of last item
6	block position
7	item position counter
8	dispatching factor*
9	status indicators*
10	symbolic tape unit reference*
11	block dispatch count
12	eight character
13	label identification (two words)*
14	current reel number (decimal)
15	address of label check routine
16-27	twelve word area for label

The entries marked by an asterisk must be supplied by the user.

The following bit positions in word File +9 have the specified functions as status indicators:

1	usage	zero=input, 1 = output
2*	block option	zero=no, 1=yes
3*	input end notice	zero=last reel only; 1=every reel
4*	output end notice	zero=none; 1=every reel
5	open for output	zero=no; 1=yes
6	open for input	zero=no; 1=yes
7	not used	
8	~~dispatching rate~~ Not Used	~~zero=less than two blocks~~ ~~1=two blocks~~

The bit positions marked by an asterisk must be supplied by the user. It should be noted that an entry of +0 would be normal.

c. Detailed Description of the File Description table Entries

File: The current item location is maintained in the least significant 15 bits of the first word in the file description table, the file label word. This permits the user to address the current data by loading an index register with the contents of the file label word.

The first read operation on an input file establishes the first item location. Subsequent read operations provide new data locations. Each read operation releases the item area used on the previous read. The input-close operation releases the last item area used and any remaining unused areas. It should be noted that the open input operation does not provide a data location.

The open operation on an output file establishes the address of an available (empty) item area. Each subsequent write operation assigns the previously supplied item area to an output block and makes available the location of a new, empty item area. The close operation on an output file releases the last unused item area to the pool.

The write-read operation assigns the item area currently available for that input file, to the block in preparation by the output file. A new input data location is then made available. The empty item area location specified in the output file label word is left undisturbed by the read-write operation. The two files used by a write-read operation must share the same pool.

File+1: Item length is the length in words of the largest item in the file. Normally, all items are the same length. For scatter-gather mode, no item may exceed 511 words.

File + 2: For input, this parameter represents the number of items in the largest block in the file. Normally, all blocks contain the same number of items. For output, this parameter determines the standard number of items to be placed in a block.

File + 3: The pool used by the file is specified here. Files with similar item length normally share the same pool. Files with widely divergent item lengths should specify separate pools. The number of item areas required in a pool is dependent on block size and the dispatching factor for input, and on block size only for output. See below under File + 8.

File + 4: For input, the file system attempts to maintain a number of items in advance of processing requirements. These unprocessed items are chained together through item chain words. The location of the chain word for the current item is maintained at this location in the file table and is used by each read operation to locate the next item to be made available to the user. For output, this word contains the address of the chain word for the first item in the output block being prepared. Each write operation adds a new item to the chain. The output file dispatcher extracts the specified number of items from the beginning of the chain to prepare an output block (using this word in the file table to locate the first unwritten item).

File + 5: When the input file dispatcher has successfully read in a new block for the file, it chains the item area locations assigned to the new block to the list of item area locations for items previously read. The dispatcher uses this word of the file table to locate the end of the chain of previously read items. Each write operation causes the currently assigned output item area to be added to the end of the chain of output items. The write operation uses this word of the file table to locate the end of the chain.

File + 6: The block counter indicates how many blocks on the current reel of the file have been read or written. The counter is used primarily for repositioning purposes.

File + 7: The item counter indicates how many items of the current block have been processed.

File + 8: For input usage, the dispatching factor represents the number of items in excess of one block to be used for advance reading, and, in effect, specifies how many item areas in the pool are to be allocated to the file. A request to read a block is made whenever the number of available item areas associated with the file equals or exceeds the number of items per block. The end-of-reel conventions limit the number of advance reads to two blocks; hence, the value specified as the dispatching factor will automatically be limited by the "OPEN" operation to a maximum of twice the number of items per block.

The entry is not used for an output file. Instead, the dispatching factor (the amount of buffering) is determined by the size of the pool used by the file. Thus, for output, the entry should be +0. The minimum number of item areas in an output pool must be equivalent to the number of items per block plus one, for each output file sharing the pool. Any item areas in excess of this number will permit the stacking of output items in the pool for later dispatching. Dispatching will be allowed to proceed at a natural rate unless the pool no longer has any free item areas for new output, under which circumstance the previously stacked output will be forcibly dispatched.

The size of a pool can thus be calculated as follows: For each input file sharing the pool, I_m + D buffer areas (where I_m is the maximum number of items per block for the file and D is the dispatching factor). For each output file sharing the pool, I_m + 1 buffer areas is minimum (where I_m is the maximum number of items per block).

To increase output efficiency, an addition I_m to 2 I_m item areas per output file is suggested.

File + 9: This word contains a number of 1-bit indicators which supply miscellaneous information about the file, as follows:

Bit 1 indicates whether the file is currently in use as input or as output.

Bit 2 indicates whether the block option is to be used in preparing the input-output dispatching for the file. (See above for a description of the block option.)

Bit 3 indicates whether the read operation should yield an end-of-file return at the end of every reel or only at the end of the last reel of the file.

Bit 4 indicates whether the write operation should yield an end-of-reel return or not.

Bit 5 is set on when the file is opened for output.

Bit 6 is set on when the file is opened for input.

Bit 8 is set on when the input dispatching factor has its maximum value of twice the number of items per block. Under these conditions the file will normally be reading two full blocks ahead, a condition which must be accounted for at end of reel or file.

The sign bit is used for input files to indicate the direction in which the file is being read. The sign is negative for backward operations.

Bits 2, 3 and 4 are set on by the compiler or user of the file. The other bits are set and interrogated during the running of the program.

File +10: This parameter specifies the symbolic tape unit reference and is normally a reference to a word on the table maintained by the tape assignment system. The tape unit number must be found in bits 24-21 of the referenced word.

File +11: This word is used as a running counter to determine when a request should be made to dispatch an input or output block. An input request is made when this counter exceeds the items-per-block (ipb) count. The counter is initialized with the sum of the dispatching factor and the ipb count, is reduced by the ipb count whenever a request is made, and is increased by one for each item read. Adjustment for short blocks is made at each block flag by using the item counter at File + 7. An output request is made when this counter equals the ipb count. The counter is initialized at zero, is increased by one for each item written, and is reset at each request.

File +12 and +13: The file identification appears as eight alphanumeric characters in these two words.

File +14: The current reel number is maintained here in decimal digit format. An open operation sets this count to zero, and internal operations at the beginning of each reel advance it by one. At the time of linkage to the input label-check routine, the count in this word should match the reel count in the input label. This word should be used as the reel count for an output label.

File +15: The presence of a nonzero address in this word indicates that the file is labeled and that the nonzero address is that of the appropriate label-checking routine. If the word is binary zero, it is assumed the file is unlabeled.

File + 16 to File +27: If the file is labeled, these twelve words are used as the label area. If the file is not labeled, these twelve words may be omitted.

d. Example of a File Description Table

In this example a file description table is shown for an input file with the following parameters:

23 words per item; 40 items per block; pool control label: IOPOOL; 35 item areas for buffering in addition to the first block; symbolic tape unit reference: A; label identification: MASTERAA; starting address of label checking routine: LBLCHECK.

INFILEA	+	0
+1	+	23
+2	+	40
+3	+	IOPOOL
+4	+	0
+5	+	0
+6	+	0
+7	+	0
+8	+	35
+9	+	0
+10	+	A
+11	+	0
+12	+	'MAST'
+13	+	'ERAA'
+14	+	0
+15	+	LBLCHECK
+16	+	0
+17	+	0
+18	+	0
+19	+	0
+20	+	0
+21	+	0
+22	+	0
+23	+	0
+24	+	0
+25	+	0
+26	+	0
+27	+	0

C. Coding Procedures

1. Construction of an input-output item area pool

Files containing items of similar size may use a common memory area or pool to share their individual item areas. The pool area may either be constructed to static specifications or dynamically constructed through the use of a special subroutine which is provided as part of the input-output routines. The pool control label is attached to the first of three words known as the pool control words which should have the following format after the pool has been constructed:

Pool Control Table	+	address of next buffer area (initially first buffer area)
+ 1	+	address of last buffer area
+ 2	+	length of individual item area in pool (not including the item chain word or the item description word)

The length of each individual item area must be equivalent to that of the largest item to share in the pool. The format of the buffer area is shown below. It should be noted that the size of a buffer area is the length of the item area plus two words, the item chain word and the item description word. The item chain word contains the address of the next available buffer area in the pool. The item description word is used to control and analyze tape format. Available item areas are chained together via the item chain words using the pool control words. The last item area will have a chain word of zero.

a. Dynamic Construction

Dynamic construction of a pool is accomplished with the following subroutine linkage:

SLJ	BUFC
+	address of first word in pool area
+	number of words allocated to pool
+	pool control label

In this case the pool control words should be defined as:

Pool Control Label	+ 0
+ 1	+ 0
+ 2	+ length of individual item areas in pool (not including the item description or the item chain word)

C. Coding Procedures

1. Input-Output Area Pool

a. General

Each file using the Tape Input-Output Item Handling Routine is required to have a memory area or pool specified, which will be used for buffering purposes. Files containing items of the same or similar length should specify the same pool area.

A pool consists of a series of buffer areas. The total size of the pool must accommodate the specified buffering for all of the files using this pool. It is required that these buffer areas be initially chained together. This is accomplished through the use of a special subroutine provided as part of the input-output routines. This routine will insert the proper chain word into each buffer area.

The size of each buffer area is the length of the largest item plus two words, the item chain word and the item descriptor. The item chain word contains the address of the next available buffer area in the pool. The item descriptor word is used to control and analyze tape format. The length of each individual item area must be as large as the largest item to share the pool. The last buffer area in a chain will have a chain word which contains zero.

Three words known as pool control words are associated with each pool. A Pool Control Label is attached to the first of these. The pool control words will contain the following after the pool has been initialized:

Pool Control Label	+	address of next buffer area (initially the first buffer area)
+1	+	address of last buffer area
+2	+	length of individual item areas in pool not including the item chain word or the item descriptor word

The number of words allocated to the pool is calculated by multiplying the item size plus 2, times the number of items of buffering desired, i.e.,

$$n\,(i+2) = s$$

where n = number of items, i = item size, and s = pool size.

b. Static Construction

Static construction of a pool is accomplished by placing the proper item chain word (the address of the next buffer area) in the first word of each buffer area.

The dynamic construction method, using the BUFC subroutine as described above, is recommended.

2. Item Handling Operators

a. Open Input Forward

```
SLJ     FOIF
 ±      File
```

Execution of this subroutine initializes the specified file description table for reading forward. If the file is labeled, the label is read and the label-checking routine is called. The sign of the calling sequence specifies whether the file is to be rewound prior to opening (-) or not (+).

b. Open Input Backward

```
SLJ     FOIB
 +      File
```

Execution of this subroutine initializes the specified file description table for reading backward. The tape is assumed to be positioned correctly (ending sentinels will be ignored when encountered). No label checking or rewind option is incorporated in the subroutine.

c. Open Output

```
SLJ     FOPO
 +      File
```

Execution of this subroutine initializes the specified file description table for writing. The sign of the calling sequence indicates whether the file should be rewound (-) or not (+). If the file is labeled, the label subroutine will be called, and the label block will be written when control is returned. The location of an empty buffer will be made available in the first word of the file description table upon return.

b. Coding Procedures

(1) Object time initialization of a pool is accomplished with the following subroutine linkage:

```
SLJ    BUFC
 +     address of first word in pool area (pool name)
 +     number of words allocated to pool (pool size)
 +     Pool Control Label
```

(2) The pool control words should be coded as follows:

```
Pool Control Label   +   0
      +1             +   0
      +2             +   length of individual item areas in pool
                         (not including the item descriptor or
                         the item chain word)
```

(3) The pool area itself may be specified in the following manner:

```
pool name      RES      pool size  (s, as defined above)
```

2. Item Handling Operators

a. Open Input Forward

```
SLJ    FOIF
 ±     File
```

Execution of this subroutine initializes the specified file description table for reading forward. If the file is labeled, the label is read and the label-checking routine is called. The sign of the calling sequence specifies whether the file is to be rewound prior to opening (-) or not (+). It should be noted that the open input operation does not provide a data location.

b. Open Input Backward

```
SLJ    FOIB
 +     File
```

Execution of this subroutine initializes the specified file description table for reading backward. The tape is assumed to be positioned correctly (ending sentinels will be ignored when encountered). No label checking or rewind option is incorporated in the subroutine.

d. Read

```
SLJ     FRD
 +      File
...     End of file (or reel) return
...     Normal return
```

Execution of this subroutine causes the location of the next item of the file to be placed in the first word of the file description table. The buffer containing the previous item is released to the pool. Detection of an end-of-file flag (or end-of-reel if such option is specified) causes an appropriate return. The end-of-reel option, if taken, requires subsequent use of the "close input reel" subroutine.

e. Write

```
SLJ     FWR
 +      File
...     End-of-reel return if specified
...     Normal return
```

Execution of this subroutine causes the location of the item currently stored in the first word of the file description table to be added to the chain of items ready for output. The location of a new item is procured from the pool and placed in the first word of the file description table.

If specified, an end-of-reel return will be made when such condition is detected. At such a return, all output previously stacked in the pool will have been written out. This permits a limited number of blocks to be written by a closing label routine, or a limited number of additional items to be inserted in the output string. The "close output reel" subroutine, which must subsequently be used in this case, will dispatch any further output items stacked in the pool and will write two end-of-reel sentinels. Approximately 25 feet of tape remain on the reel at the time of end of reel return to accommodate any closing blocks.

f. Write-Read

```
SLJ     FWRD
 +      Input file
 +      Output file
...     End return with code in AR8
...     Normal return
```

Execution of this subroutine causes the location of the item currently stored in the first word of the input file description table to be added to the chain of items ready for dispatching in the output file. The location of a new input item is made available, as in the "read" operation; the output buffer location in the first word of the output file description table is left undisturbed. Input or output end notice will be given as specified in the respective file description tables. The code supplied in AR8 upon return will be 1 for input end, 2 for output end, and 3 should both occur simultaneously.

g. Close Input Reel

```
SLJ     FCIR
 +      File
```

If the end-of-reel return option is elected for an input file, the close input reel subroutine must be executed in order to advance to the next reel of the file. The subroutine executes a rewind and swaps tapes.

h. Close Input File

```
SLJ     FCIF
 ±      File
```

Execution of this subroutine causes all outstanding item buffers to be released to their pool. Tape swapping is suspended, and the sign of the calling sequence indicates whether the tape is rewound (-) or not (+). The file description table is stabilized and may subsequently be re-opened.

i. Close Output Reel

```
SLJ     FCOR
 +      File
```

Execution of this subroutine causes all output currently stacked in the pool to be dispatched and two end-of reel sentinels to be written. The subroutine executes a rewind and swaps tapes.

j. Close Output File

```
SLJ     FCOF
 ±      File
```

Execution of this subroutine causes the last unused buffer to be released to the pool, all output currently stacked in the pool to be dispatched, and two end-of-file sentinels to be written. Tape swapping is suspended, and the sign of the calling sequence indicates whether the tape is to be rewound (-) or not (+). The file description table is stabilized and may subsequently be reopened.

k. General Close Reel

```
SLJ     FCLR
 +      File
```

This subroutine performs the input and output reel closing functions described above, determining by analysis of the file description table whether an input function or an output function is required. The routine isdesigned for use in programs which do not know at the time the calling sequence is programmed which function the file table will be performing at execution time.

l. General Close File

```
SLJ     FCLF
 +      File
```

This subroutine performs the input and output file closing functions described above, determining by analysis of the file description table whether an input function or an output function is required.

TAPE INPUT-OUTPUT VARIABLE SIZE ITEM HANDLING

A. Purpose

To provide a set of tape input-output variable size item handling routines.

B. Method

1. Structure of the variable size item handling routines

File description tables constitute the highest logical level within the tape Input-Output system. Entries in this table are either defined as constants by the user or compiled from given parameters by a special subroutine. These tables are interpreted and the information in the files they represent is processed by a group of subroutines which perform the functions customarily associated with item handling operation: open, close, read and write. The item handling operators in turn communicate to lower-level routines which act as a file dispatcher. The file dispatcher maintains a queue of requests generated by the item handling operators and coordinates these requests with the request-and-verify mechanism of the basic interrupt system. This coordination is accomplished through the intermediate level (block handling) tape input-output package which consists of the following block handling functions: read, write, overwrite, position and rewind. The levels of the item handling input-output system are thus seen to be:

a. File description table entries
b. Item handling operators
c. File dispatcher subroutines
d. Intermediate level tape handling subroutines
e. Basic request and verify routines
f. Basic interrupt dispatchers

The file description table entries and the item handling operators are discussed below. The use of the intermediate level tape handling subroutines in which the user must provide his own item advance routine, is described in another section of this manual. The basic request and verify routines and the basic interrupt dispatchers are described in the BOSS III manual. (Section IV, Synchronizer Control).

2. Tape Formats: Labels, Data Blocks, and Sentinels

a. General

The input-output file system produces and accepts tapes whose format follows the conventions described below. Tapes produced by the Variable Size Item Handling System will contain the standard labels, sentinels and flags described below. Tapes read by the system must conform to the standard format.

b. Label Block

(1) Label Block Processing

If a data tape is labeled, the first block must be the label block. The presence of an address as one of the entries in the file description table (which is described below) indicates whether or not the file is labeled. If such address is present, the first block of each reel of an input file will be read with a block-read tape order into the last twelve words of the file description table for that file, and a subroutine linkage will be made to the address specified. The subroutine located at the address given is assumed to be either a standard or special label-checking program which will verify the contents of the label just read. For output files, the subroutine linkage will be made at the beginning of each reel, and the last twelve words of the file description table for that file will be written as a block upon return from the out-put label subroutine.

If the file is not labeled, a zero address is entered in the file description table as the location of the label checking routine. The first block on each reel of the file is then assumed to be a data block in standard format, as described below.

If input labels are present but are not to be checked, a label subroutine must still be provided. It can simply return control to the input-output routine without processing or checking the label.

(2) Label Block Layout

Word	Content	Information
0	-00000000	Label flag
1	AAAA	First four alphanumeric characters of file label
2	ddmmyy	Date: day, month and year in decimal characters
3	00rrr	File reel number in decimal characters
4		These words are available for use by the individual installation.
5		
6		
7		
8		
9		
10	AAAA	Last four alphanumeric characters of file label
11	-00000000	Label flag

c. Data Blocks

(1) General

Data Blocks consist of one or more logical items of variable length (the maximum item length must be equal to or less than the block length) and two data descriptor words, one at each end of the block. The data block and data descriptor word formats are described below. Each item is preceded and followed by an item descriptor word, as shown. This item descriptor need not be considered by the programmer to be a part of the item. These words are placed in the block by the handling routine on output and are expected

to be present on input. Thus, the input format required must be generated by the use of the Variable Size Item Output Handling routines.

(2) Data Block Layout

Segment separators occur on tape between the initial data descriptor word and the first item of the block, and between the last item of the block and the terminal data descriptor word. When writing, the data descriptor words are automatically prepared by the file dispatcher. When reading, the first data descriptor encountered (depending on the direction the tape is being read) is accounted for as a file position marker in order to facilitate restart. The item descriptor words are used to present the address of the current item in word zero of the file description table on input, and are created from ~~words zero and one on~~ output.

(a) Data Descriptor Words

Data descriptor words consist of a one word marker at each end of a block, consisting of the channel increment and the block number. The channel increment is located in the upper half of the data descriptor word and specifies the number of words in the block. The block number is in the lower half of the word and specifies the block number within the reel, modulo 4096. Both of these fields are 12 bit binary numbers. The sign of data descriptor words is positive.

(b) Items

The data area within the block consists of one or more items. The item size may vary from one word to ~~4096 (maximum block size)~~ and is always an integral number of words.

(c) Item Descriptor Words

Preceding each item is an item descriptor word which is used to control and analyze the tape format. This word is not considered to be part of the item for addressing

and processing purposes. On output, the user need only indicate the item length (in words) as word one of the file description table in the call to the output handling routines. Should the user wish to access the item descriptor, it is found in the address previous to that given as the current item address in the file description table.

The item descriptor is composed of two fields, the length of the previous item as a binary value in the most significant 12 bits and, the length of the next item as a binary value in least significant 12 bits of the word. The first item descriptor of a block has the length of the previous item as zero. The last item descriptor of a block has the length of the next item as zero. Under normal usage the user does not need to make reference to the item descriptor. The item address given the user by this system will be the address of the first word of actual data in the item.

(3) Data Block Processing

(a) Item Handling

When a request for an input item advance has been honored, the starting address of the next item area available will be found in the first location of the file description table associated with that file. The user can then load an index register with this address or can access the data indirectly using the label of the file description table as the operand.

When a VRD (Variable item ReaD) operator is given, the current item area will be released back to the pool. When a VWR (Variable item WRite operator is given, the item whose starting address is in ~~word zero~~ of the file description table associated with that file is moved to an output block area for that file. The length of the item moved to output is determined by ~~word one~~ of the file description table. The block area will be released when the tape write operation involving that block area is completed.

d. Sentinels

Three types of sentinels exist in this system. There are the end-of-file sentinel, the end-of-reel sentinel and the bypass sentinel. The end-of-file and the end-of-reel sentinels each consist of a one word block. The high order 2 bits and sign of this one word are a flag signifying the type of sentinel. The low order 22 bits of this word are a block count for all of the label blocks, data blocks and sentinel blocks in the reel to this point, including the sentinel block.

Bypass sentinels are used to indicate that a portion of the data on a tape does not pertain to the file which contains that tape. Two bypass sentinels will appear before and after the block or blocks containing this extraneous information. These blocks are ignored when encountered by the item handling routine and the information contained in them will not be given to the user, or counted for block count purposes. These bypassed blocks are generally used for memory dumps.

3. File Description Table

a. General

The characteristics of each file to be processed by a program utilizing the Item Handling Subroutines must be provided by the user. These characteristics are described in a table called the File Description Table. This table consists of 28 words in the case of a labeled file and 16 words in the case of an unlabeled file. The entries in this table, which must be supplied by the user, are marked by an asterisk in the following table. The other entries are used internally by the Variable Size Item Handling Subroutine. The values of the entries supplied by the user are dynamically alterable and may be changed whenever the file is not open. The label of the first entry in the table will be the "file label" as used in the calling sequences described in the section on file operators.

b. File Description Table

File label	current data location
+ 1	item length
+ 2	maximum block length*
+ 3	pool label*
+ 4	location of current block chain word
+ 5	location of last block chain word
+ 6	block position counter
+ 7	~~item position counter~~
+ 8	dispatching factor*
+ 9	status indicators*
+ 10	symbolic tape unit reference*
+ 11	block dispatch count
+ 12	eight character label
+ 13	identification (two words)*
+ 14	current reel number (decimal)
+ 15	address of label check routine*
+ 16-27	twelve word area for label

The following bit positions in word File + 9 have the specified functions as status indicators:

The bit positions indicated by an asterisk must be supplied by the user.

1	usage	zero=input, 1= output
2	block option	this bit is ignored by the Variable Size Item Handling routines
3*	input end notice	zero=last reel only; 1=every reel
4*	output end notice	zero=none; 1=every reel
5	open for output	zero=no; 1=yes
6	open for input	zero=no; 1=yes
7	not used	
8	dispatching rate	zero=less than one block 1=two blocks

c. Detailed description of the File Description Table Entries

File: The current data location is supplied by the input routines in the least significant 15 bits of the file label word, permitting a program to locate the current data by indirectly addressing the file label, or by loading this word in an index register used in referencing the item. The first read operations on an input file establishes the first data location. Subsequent read operations produce new data locations. Each read releases the item areas used on the previous read, and the input-close operation releases the last item area. Each write operation moves the output item specified by the user in the first two words of the file description table to the output buffer area, i.e., each time the user wishes to send an item to output, he places the address of the item in the first word of the file description table, the length of the item in the second word (File+1), and calls the write operation. The close operation on an output file releases the last block area to the pool.

File + 1: Item length is the length in words of the current item to be sent to output. The maximum item length is equal to the block length. This entry is not specified by the user for input files.

File+2: For both input and output, this entry is the maximum block size in words.

File + 3: The location of the pool control table used by the file is specified here. Files with similar block length normally share the same pool. Files with widely divergent block lengths should specify separate pools. Input and output files must not share the same pool. The number of areas required in a pool is dependent on the dispatching factor for input, and on block size only for output. See below under File + 8.

File + 4: For input, the file system attempts to maintain a number of items in advance of processing requirements. The block area locations of these unprocessed items are chained together through chain words. The location of the first unprocessed block is maintained at this location in the file table and is used by each read operation to locate the next block to be made available to the user.

File + 5: When the input file dispatcher has successfully read in a new block for the file, it chains the block area location assigned to the new block to the list of block area locations for blocks previously read. The dispatcher uses this word of the file table to locate the end of the chain of previously read blocks. Each write operation causes the currently assigned output area to be added to the end of the chain of output blocks. The write operation uses this word of the file table to locate the end of the chain.

File + 6: The block counter indicates how many blocks on the current reel of the file have been read or written. The counter is used primarily for repositioning purposes.

File + 7: The item counter indicates how many items of the current block have been processed.

File+ 8: For input usage, the dispatching factor represents the number of blocks in excess of one block to be used for

advance reading, and, in effect, specifies how many areas in the pool are to be allocated to the file. The end-of-reel conventions limit the number of advance reads to two blocks; hence the value specified as the dispatching factor will automatically be limited by the "OPEN" operation to a maximum of two blocks.

For output usage, the dispatching factor is not explicitly stated but implied by the size of the pool used by the file. The minimum number of block areas in an output pool must be equivalent to the number of files sharing the pool. Any areas in excess of this number will permit the stacking of output blocks in the pool for later dispatching. Dispatching will be allowed to proceed at a natural rate unless the pool no longer has any free areas for new output, under which circumstance the previously-stacked output will be forcibly dispatched.

File+9: This word contains a number of 1-bit indicators which supply miscellaneous information about the file, as follows:

Bit 1 indicates whether the file is currently in use as input or as output.

Bit 2 is not used by the Variable Item Size Handling Routine.

Bit 3 indicates whether the read operation should yield an end-of-file return at the end of every reel or only at the end of the last reel of the file.

Bit 4 indicates whether the write operation should yield an end-of-reel return or not.

Bit 5 is set on when the file is opened for output.

Bit 6 is set on when the file is opened for input.

Bits 3 and 4 are set on by the compiler or user of the file. The other bits are set and interrogated during the running of the program.

File+10: This parameter specifies the symbolic servo unit

reference and is normally a reference to a word on the table maintained by the tape assignment system. The servo unit number must be found in bits 24-21 of the referenced word.

File+11: This word is used as a counter in order to determine when a request should be made to dispatch an input or output block.

File+12: and +13: The file identification appears as eight alphanumeric characters in these two words.

File+14: The current reel number is maintained here in decimal format. An open operation sets this count to zero, and internal operations at the beginning of each reel advance it by one. At any label exit, the count in this word should match the reel count in an input label, and is used as the reel count for an output label.

File+15: The presence of a nonzero address in this word indicates that the file is labeled and that the nonzero address is that of the appropriate label-checking routine.

File+16 to File+27: If the file is labeled, these twelve words are used as the label area. If the file is not labeled, these twelve words may be omitted.

d. Example of a File Description Table

In this example a file description table is shown for an input file with the following parameters:

400 words per block; pool labels: INPOOL; 1 block area for buffering in addition to the first block; symbolic tape unit reference: A; label identification: MASTERAA; starting address of label checking routine: LBLCHECK.

```
INFILEA  +     0
         +     0
         +   400
         +  INPOOL
         +     0
         +     0
         +     0
         +     0
         +     1
         +     0
         +     A
         +     0
         +  'MAST'
         +  'ERAA'
         +     0
         +  LBLCHECK
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
         +     0
```

C. Coding Procedures

1. Construction of an input-output area pool

Files containing blocks of similar length may use a common memory area or pool. The pool may either be constructed by the user to static specifications, or dynamically constructed through the use of a subroutine provided as a part of the input-output routines. The label of a pool is attached to the first of three words known as the pool control words, which have the following format:

Pool Label	+	address of first buffer area
	+	address of last buffer area
	+	length of the individual buffer area in this pool

The length of each block buffer area must be equivalent to that of the largest block area in the pool. Preceding each buffer area is a chain word in which the address of the next buffer area is located. The chain word of the last buffer area is zero. Dynamic construction of a pool is accomplished by coding the following subroutine linkage:

SLJ	VBC
+	address of the first word in the area to become the pool
+	number of words to be allocated to the pool
+	pool label

In this case the pool control words are defined as follows:

pool label	+	0
	+	0
	+	length of individual block areas in this pool

2. Item handling operators

a. Open Input Forward

SLJ	VOIF
±	File

Execution of this subroutine initializes the specified file description table for reading forward. If the file is labeled, the label is read and the label-checking routine is called. The

sign of the file label indicates whether the file is to be rewound prior to opening (-) or not (+).

b. Open Input Backward

```
SLJ    VOIB
 +     File
```

Execution of this subroutine initializes the specified file description table for reading backward. The tape is assumed to be positioned correctly (ending sentinels will be ignored when encountered). No label checking or rewind option is incorporated in the subroutine.

c. Open Output

```
SLJ    VOPO
 ±     File
```

Execution of this subroutine initializes the specified file description table for writing. The sign of the calling sequence indicates whether the file should be rewound (-) or not (+). If the file is labeled, the label subroutine will be called, and the label block will be written when control is returned.

d. Variable Size Item Read

```
SLJ    VRD
 +     File
...    End of file (or reel) return
...    Normal return
```

Execution of this subroutine causes the location of the next item of the file to be placed in the first word of the file description table. The buffer containing the previous item is released to the pool. Detection of an end-of-file flag (or end-of-reel if such option is specified) causes an appropriate return. The end-of-reel option, if taken, requires subsequent use of the "close input reel" subroutine.

e. Variable Size Item Write

```
SLJ    VWR
 +     File
...    End-of-reel return if specified
...    Normal return
```

Execution of this subroutine causes the item whose address is stored in word zero of the file description to be moved to the output block area. The number of words moved is controlled by the item length stored in word one of the file description table. When the current item to be moved to output will not fit within the current output block area, considering maximum block size, the current block area is dispatched and the current item is moved to the next available block area. If specified, an end-of-reel return will be made when such condition is detected. At such return, all output previously stacked in the pool will have been written out. This permits a limited number of blocks to be written by a closing label routine, or a limited number of additional items to be inserted in the output string. The "close output reel" subroutine, which must subsequently be used in this case, will dispatch any further output items stacked in the pool and will write two end-of-reel sentinels. Approximately 25 feet of tape remain on the reel at the time of end of reel return to accomodate any closing blocks.

f. Close Input Reel

```
SLJ    VCIR
 +     File
```

If the end-of-reel return option is elected for an input file, the close input reel subroutine must be executed in order to advance to the next reel of the file. The subroutine executes a rewind and swaps tapes.

g. Close Input File

```
SLJ    VCIF
 ±     File
```

Execution of this subroutine causes all outstanding item buffers

to be released to their pool. Tape swapping is suspended, and the sign of the calling sequence indicates whether the tape is rewound (-) or not (+). The file description table is stabilized and may subsequently be re-opened.

h. Close Output Reel

```
SLJ    VCOR
 +     File
```

Execution of this subroutine causes all output currently stacked in the pool to be dispatched and two end-of-reel sentinels to be written. The subroutine executes a rewind and swaps tapes.

i. Close Output File

```
SLJ    VCOF
 ±     File
```

Execution of this subroutine causes the last unused buffer to be released to the pool, all output currently stacked in the pool to be dispatched, and two end-of-file sentinels to be written. Tape swapping is suspended, and the sign of the calling sequence indicates whether the tape is to be rewound (-) or not (+). The file description table is stabilized and may subsequently be re-opened.

j. General Close Reel

```
SLJ    VCLR
 +     File
```

This subroutine performs the input and output reel closing functions described above, determining by analysis of the file description table whether an input function or an output function is required. The routine is designed for use in programs which do not know at the time the calling sequence is programmed which function the file table will be performing at execution time.

k. General Close File

SLJ VCLF
± File

This subroutine performs the input and output file closing functions described above, determining by analysis of the file description table whether an input function or an output function is required.

CARD TO TAPE SYMBIONT (CTS)

A. PURPOSE

The purpose of CTS is to convert alphanumeric punched cards to UNIVAC III magnetic tape in standard file format.

B. METHOD

CTS is called and controlled by operator type ins. Type outs from CTS indicate end-of-reel, reader errors, and reader end-of-file conditions.

Each block on tape contains from one to twenty-five items, depending on the user's specification in the LABEL card. A blocking factor of 25 is assumed if this specification is not present. Smaller blocks than specified may be written prior to restart points or the end-of-file point. Each card image becomes a 20-word item.

CTS is currently assembled to write on the tape unit specified in file 14 (tape assignment table entry 0216). This may be altered by reassembling, changing the EQU card which defines CTSTAPE (CTSTAPE EQU 0216). CTSTAPE may be left undefined at assembly time by removing the EQU card. In this case, it must be defined at object time or SUCO time by an operator type-in. (See DEF card description in DECO, SUPPORT III Section 3-0005.) Channel number (CDCH) and the tape channel priority reservation entry CTSTENT can be changed by the user in the same manner.

C. CONTROL CARDS

CTS recognizes three control cards. They are the LABEL card, the END OF FILE card, and the RESTART card. The format of each follows. Words printed in capital letters must be punched as shown. Lower-case words represent fields to be supplied by the user.

1. LABEL Card

Card Columns	Content
1	Z (12-0-2 punch)
2	Δ
3 - 7	LABEL
8	Δ
9 - 16	file identity (8 characters, which will become words 1 and 10 of the output file label)
17	Δ
18 - 23	date

Card Columns	Content
24	Δ
25 - 26	number of items (card images) per block. If this field is blank, 25 items per block are recorded.
27	Δ
28 - 29	reel number. If blank, reel number in the file label will be 000001 (decimal).

The fields supplied by the user in the LABEL card will be placed in the appropriate words of the label block on the output tape.

2. RESTART Card

Card Columns	Content
1	Ƶ (12-0-2 punch)
2 - 8	RESTART

A RESTART card will cause CTS to write four bypass sentinels on the output tape. If a situation arises which requires recovery during the card-to-tape process, the tape will be repositioned to the last restart point (4 bypass sentinels) when an SkΔRESTART type-in is given.

put cards after RESTART back in reader?

3. END OF FILE Card

Card Columns	Content
1	Ƶ (12-0-2 punch)
2 - 12	ENDΔOFΔFILE

When the END OF FILE card is encountered, end-of-file sentinels are written onto the output tape, and that tape is then rewound. A RDRΔEOF message is typed out, and CTS releases control.

An END OF FILE card should be followed by 4 blank cards.

D. TYPEWRITER COMMUNICATION

Console type-ins and type-outs shown below are used to control the symbiont. (k in all messages refers to the card reader channel number.)

1. Symbiont Initialization

a. RSΔCALLΔCTSΔk

This type-in will cause the symbiont (CTS) to be called into memory. k is the channel designator, and its specification in the CALL type-in is for purposes of typewriter communication only.

After initialization, if card reader channel number, file number, and/or tape channel priority reservation entry have been defined with a DEF card, the following will occur: CDCH, CTSTAPE, and/or CTSTENT will type out. The operator must then type in channel number, file number, and/or reservation entry, respectively.

b. RS△KILL△k1△CALL△CTS△k2

This type-in will cause the symbiont currently using channel k1 to be killed, and CTS is called, assigned to channel k2. If there are no symbionts to be killed, 0 may be entered for k1 (or the RS△CALL type in described above may be used).

Main programs may be called in the same manner, using program name in place of CTS, and 2 for k2.

2. Type-ins

a. Sk△START

This type-in causes card reading to begin. The first five cards are checked for a LABEL card. Data on cards preceding the LABEL card is not recorded on the output file. If a LABEL card is not found, the symbiont types out the message NO△LABEL.

The LABEL card should always be preceded by four blank cards,

b. Sk

This type-in causes the symbiont to cease processing. The type-in Sk△GO will cause processing to resume.

c. Sk△GO

This type-in causes the symbiont to continue processing after a stop.

d. Sk△RESTART

Prior to this type-in, the operator should have repositioned the

input card file to the previous RESTART card. SkΔRESTART causes the symbiont to position tape prior to the last restart point on the output file. Card reading is resumed, and the first five cards are checked for a RESTART card. The recovery point is written on the output file and normal operation is then resumed.

3. Type-Outs

a. NOΔLABEL

When a LABEL card is not found among the first five cards read, this message is typed out. The operator may type in GO to ignore the label, or he may put a LABEL card in the reader and repeat the SkΔSTART type-in. The message may have been caused by an illegally punched LABEL card.

b. NOΔRESTART

In the event that one of the first five cards read on recovery is not a RESTART card, the type-out NOΔRESTART occurs. When the proper restart point in the card file is located, the SkΔRESTART type-in should be repeated.

c. CH5ΔEOT

This type-out occurs when an end-of-reel condition is encountered on the output file. End-of-reel sentinels are recorded, and the file is rewound. If further card to tape conversion is desired, a LABEL card specifying the proper reel number should precede the remainder of the card file, and after a blank tape is mounted, SkΔSTART should be typed in.

d. RDRΔEOF

This type-out occurs when an END OF FILE card is read. End-of-file sentinels are written on the output file and the output file is rewound.

e. WHAT

This type-out is the response to any illegitimate type-in.

f. READER

This message is typed out when a card reader error or fault occurs. The cards, if any, in stacker 0 should be repositioned in front of the remaining card file. An SkΔGO will cause card reading to proceed.

TAPE TO PRINTER (TPRS)

A. PURPOSE

The purpose of TPRS is to provide a standard tape-to-printer procedure with a moderate amount of automatic carriage control.

B. METHOD

TPRS accepts an input file in standard data format. The tape must have been written using standard label, bypass, end-of-file, end-of-reel, and block sentinel conventions. Each print item (record) on the tape will contain a line control word followed by a line image of from 1 to 32 words.

Maximum block size for the input file is 254 data words (256 words, including the two Data Descriptor words).

Each page will have a standard heading and footing of six lines each, assuming the operator has correctly aligned the paper. The 6-line heading and 6-line footing may be modified by the user with a carriage control word. A 66-line form (11 inches) is assumed by the symbiont. If a different form length is used, its length should be indicated in a carriage control word.

TPRS is currently assembled to read from the tape unit specified in file 15 (tape assignment table entry 0217). This could be altered by reassembling, changing the EQU card which defines TPRSTAPE (TPRSTAPE EQU 0217). TPRSTAPE could also be left undefined at assembly time by removing the EQU card. It should then be defined at object time or SUCO time by an operator type-in. (See the DEF card description in SUPPORT III, Section 3-0005.)

The channel number (PRCH) assigned to the printer, and the tape channel priority reservation entry TPRSTENT might also be altered in the same manner.

C. CONTROL WORDS

1. The line control word must be the first word of each print item. It specifies the number of words in each line image, and it controls page and line advance. Printing begins in print position 1.

6 lines per inch

The format of the line control word:

Bits	Content
25	0
24 - 22	not used
21 - 16	length of line ($\leq$ 32)
15 - 9	0000000
8	eject control
7 - 1	line advance

If there is a 1 in bit 8 (eject control) of the control word, page ejection will occur immediately. Ordinarily, printing will occur on a page until the footing space is reached, at which time there is an automatic eject to the first line after the heading space of the next page.

A combination of a 1 in eject control and an entry in line advance (bits 7 - 1) causes an immediate eject to the line number on the next page specified in bits 7 - 1.

An entry in line advance without an entry in eject control specifies the number of lines to be spaced before printing. Thus the user can double-space, triple space, or print with irregular spacing.

Punch control words are indicated by 017777 in bits 1 - 13, and will be automatically bypassed by TPRS.

2. A carriage control word may be used if the user wishes to specify a non-standard page heading and footing (other than 6 lines each) and/or a non-standard form length (other than 66 lines).

The format of the carriage control word:

Bits	Content
25	1
24 - 22	not used
21 - 15	length of page heading, in lines
14 - 8	length of page footing, in lines
7 - 1	length of form, in lines

When this control word is used, all fields must be filled in since this word will cause new specifications to replace the standard heading and footing and form length. Therefore, even if the user is modifying only page heading, for example, he must indicate 6 for page footing and 66 for form length.

When the carriage control word is used, it should be followed by a line control word, which specifies a line length one less than normal. The remainder of the item should be a dummy item which is one word shorter than normal. The print image in an item containing a carriage control word will not be printed.

D. TYPEWRITER COMMUNICATION

The entry "k" in all typewriter messages indicated here refers to the printer channel number. The user must designate the channel being used.

1. Symbiont Initialization

a. RSΔCALLΔTPRSΔk

This type-in will cause TPRS to be called into memory. The designation here of printer channel number ("k") is for purposes of typewriter messages only. This does not obviate the need for defining an undefined channel number as described under "METHOD".

After initialization, if printer channel number, file number, and/or tape channel priority reservation entry have been defined with a DEF card, the following will occur: The messages PRCH, TPRSTAPE, and/or TPRSTENT will type out. The operator must then type in channel number, file number, and/or reservation entry, respectively.

b. RSΔKILLΔk1ΔCALLΔTPRSΔk2

This type-in will cause the symbiont currently using channel k1 to be killed. TPRS is called and assigned to channel k2. If there is no symbiont to be killed, 0 may be entered for k1, or the RSΔCALL type-in may be used.

2. Operator Type-ins

a. SkΔSTART

This causes tape movement to begin. First the tape will be rewound and if there is a label, the eight-character I D will be typed out (along with date and reel number). The first line item will be printed. The symbiont then releases to allow the operator to check the form alignment.

b. SkΔGO

This type-in causes printing to continue after a stop.

c. SkΔTEST

A single line will be printed. This type-in can be used together with manual printing to set up form alignment. The same line image will be printed each time SkΔTEST is repeated. Typing in SkΔGO will cause the symbiont to continue normal printing, beginning with the next line item.

d. Sk

The symbiont will release without further printing, and printing will be continued when SkΔGO is typed in.

e. SkΔBACKΔn

The symbiont will back up n pages (n = 0, 1...99) and then resume printing. Backing up is accomplished by checking for cumulative line advance on a page when the type-in occurs between the head and foot of a page. When it occurs at the foot of a page, TPRS will back up until the distance backed up exceeds the distance from head to foot. If an eject bit is found anywhere, it is assumed that this is the head of a page. If ejects are not used, then vertical line positioning of a page will normally not agree with that prior to backing up. A precise number of pages backed up is not guaranteed, and the symbiont may back up more pages than specified.

The symbiont will print a single line and then release to permit paper alignment, if desired.

3. Type-outs

a. ENDΔPRINTING

This message is typed out when an end-of-reel or end-of-file sentinel is encountered on the input tape. The tape is not rewound. The print routine will release. A new reel may be commenced by manually rewinding the old reel, replacing it with the new reel, and typing in SkΔSTART. If the user prefers, the symbiont could be called again, and the new reel defined on another unit.

b. WHAT

This message will respond to an unrecognizable type-in. The symbiont will release, and the operator should type in a corrected message.

c. BADΔBYPASSΔSENTINEL

This type-out occurs when an illegal bypass sentinel is encountered on tape. It is not possible to recover since it ordinarily indicates that the tape was incorrectly produced.

d. BADΔEOFΔSENTINEL

This message is caused by an illegal end-of-file sentinel. It ordinarily indicates an incorrectly produced tape, and recovery is not possible. (Restart is not generally necessary since printing is finished at this point.)

e. ILLEGALΔEOFΔORΔLABELΔSENTINEL

This type-out occurs when an illegal end-of-file or label sentinel is encountered when tape is being read backward after an SkΔBACK type-in. The tape should be rewound and the symbiont restarted.

f. NOΔSTARTΔTYPE-IN

This type-out requests an RSΔSTART type-in. Probably an RSΔGO has been used where an RSΔSTART was required, or RSΔSTART was not typed in after symbiont initialization.

TAPE-TO-PUNCH SYMBIONT - (TPCS)

A. PURPOSE

TPCS provides a standard tape-to-punch procedure which will punch translated card images from a conventional punch tape or from a combined printing and punching tape.

B. METHOD

TPCS accepts an input file in standard data format. The tape must have been written using standard bypass end-of-file, end-of-reel, and block sentinel conventions. Each punch item (record) on the tape will contain a punch control word followed by a card image of from 1 to 20 words.

Maximum block size for the input file is 254 data words (256 words, including the two Data Descriptor words).

TPCS is currently assembled to read from the tape specified in file 13 (tape assignment table entry 0215). File number could be changed by the user by reassembling, changing the EQU card which defines TPCSTAPE (TPCSTAPE EQU 0215). Another method would be to leave TPCSTAPE undefined at assembly time by removing the EQU card. A DEF card should be used to indicate the symbol as operator-defined. TPCSTAPE would then be defined by an operator type-in at object time or SUCO time. (See the DEF card description in SUPPORT III, Section 3-0005.)

The channel number (PUCH) assigned to the punch, and the tape channel priority reservation entry (TPCSTENT) might also be altered in the same manner.

C. PUNCH CONTROL WORD

The punch control word must be the first word of each punch item. It specifies the number of words in each card image, and specifies the punching mode (binary or Hollerith). Punching begins in column 1.

The format of the punch control word:

Bits	Content
21 - 16	number of words in punch item
15	punch mode
14	punch stop bit
13 - 1	17777

A "1" in punch mode field (position 15) indicates that punching is to be in Hollerith code. A zero indicates binary code (24-bit words).

When there is a "1" in position 14 (punch stop bit), the card acts as a restart card. Punching will stop, and the symbiont will release after the "restart" card is punched. This permits the operator to mark the position of the last card to enter the stacker for possible later restart. It should be noted that the last card in the stacker is the card immediately preceding the "restart" card since the last card processed is still in the punch. If a restart becomes necessary later, the operator should remove all cards in the stacker after the marked card, including the "restart" card, since it will be punched again.

The 17777 in positions 13 - 1 distinguishes the punch control word. A print control has a different entry in this field, and print items will simply be bypassed by TPCS.

D. TYPEWRITER COMMUNICATION

The entry "k" in all of the following typewriter messages refers to the punch channel number. The user must designate the channel being used.

1. Symbiont Initialization

a. RS△CALL△TPCS△k

This type-in will cause TPCS to be called into memory. The designation of punch channel number ("k") is for purposes of typewriter communication only, and its use here does not cancel the need for elsewhere defining an undefined channel number as described on page 1.

After initialization, if punch channel number, file number, and/or tape channel priority reservation entry have been defined with a DEF, the following will occur: The messages "PUCH", "TPCSTAPE", and/or "TPCSTENT" will be typed out. The operator must then type in channel number, file number, and/or reservation entry, respectively.

b. RS△KILL△k1△CALL△TPCS△k2

This type-in will cause the symbiont currently using channel k1 to be killed. TPCS is called and assigned to channel k2. If there is no symbiont to be killed, 0 may be entered for k1, or the RS△CALL△TPCS type-in may be used.

A main program could also be called with this type-in. Program name would replace "TPCS".

2. Operator Type-ins

a. Sk△START

This type-in causes the input file to rewind, the first block to type out as the label block, and punching to begin.

b. Sk

This causes the symbiont to stop and release control. Punching will continue when Sk△GO is typed in.

c. Sk△GO

Causes the symbiont to continue after a stop.

d. Sk△RESTART

This type-in causes the symbiont to search the tape backwards until a punch control word with a 1-bit in position 14 (punch stop bit) is encountered. TPCS will then resume by punching the record thus flagged as the restart point. After typing in Sk△RESTART, the operator must remember that the first card out of the punch is not the first card from the restart point, but it is the "restart" card, and was actually punched previously. It is assumed, therefore, that before typing in Sk△RESTART, the operator has removed all the cards in the stacker back to the card flagged at the previous restart stop.

3. Type-outs

a. END△PUNCHING

When an end-of-file or end-of-reel sentinel is encountered on the input tape, this message is typed out. TPCS then releases, without rewinding the tape. A new reel may be commenced by manually rewinding the old reel, replacing it with the new reel, and typing in SK△START. The new reel could be defined as a different file. The symbiont would then have to be re-initialized.

b. PUNCH△RESTART

This message is typed out when a punch control word with a 1-bit in position 14 is encountered. The symbiont releases, and the operator should remove and flag the cards in the punch stacker, and then type in SK△GO to continue.

c. WHAT

This type-out occurs in response to an unrecognizable type-in. The symbiont releases, and the operator should type in the corrected message.

PUNCHED PAPER TAPE READER SYMBIONT

A. PURPOSE

To convert punched paper tape data to magnetic tape using 500 or 250 characters per sec paper tape reading speeds.

B. METHOD

The absence of standard conventions on the part of the user as to use and character set for punched paper tape requires that the conversion of punched paper tape to magnetic tape provide an untranslated and unpacked image of the input data in the output file. The user is then free to manipulate the data image according to his own conventions during a subsequent run, availing himself of the higher input speed of magnetic tape.

The output of this run is in standard block format, with standard label and sentinels. Each data block contains 256 data words (equivalent to 256 frames of punched paper tape).

UNIVAC will make adjustment, upon request, of the number of data words (frames) per block of the output file of the symbiont program.

The console type-ins and type-outs shown below are used to control the symbiont.

Sk Δ aaaaaaaa

- k defines the paper tape reader channel number. a is the eight character file ID desired in the output label block. This type-in causes the symbiont to write the output label block and commence conversion of paper tape data to the output file.

Sk Δ GO

- this type-in causes the symbiont to continue processing after any stop. This type-in should not be used after mounting a new reel of paper tape as no recovery point will be established.

Sk Δ END

- this type-in causes end-of-file sentinels to be written on the output file, and rewind of the file.

Sk or Sk ΔΔΔΔΔ STØP

- this type-in causes the symbiont to stop processing.

SkΔ FLT

- this type-out occurs when the fault indicator of the paper tape reader is set. The Fault Indicator may be set by an end of paper tape condition in which case the operator may terminate the run or mount the next reel and call for continued operation. In the event of a Reader Fault other than end of paper tape, the operator may terminate the run or clear the fault and recover from the start of the current paper tape reel.

Sk Δ ERR

- this type-out occurs when the Error Indicator of the paper tape reader is set. The operator may terminate the run or recover from the start of the current reel of paper tape.

Sk Δ NXT

- this type-in is used to establish a recovery point before commencing conversion of the next reel of paper tape. After this type-in, the type-out Sk nn occurs, where nn indicates that the reel of paper tape mounted on the reader is the nn th reel of paper tape to be converted to the current output file. The operator should record the value nn on the paper tape reel to aid in recovery, should such action become necessary.

Sk Δ WS

- this type-out occurs when the paper tape reader wired Stop Indicator is set. The group of characters read from paper tape up to the occurrence of the wired stop character will be written on the output file.

Sk Δ Rnn

- this type-in is used to recover the run by rerunning the conversion from the start of paper tape reel nn. On receiving this type-in, the output file will be rewound and

read forward until the bypass block identifying the start of paper tape reel nn is located. Conversion of reel nn to the output file then occurs.

Sk Δ NØ

- this type-out responds to any illegitimate type-in.

C. OUTPUT FILE FORMAT

1. Label block

Word	Content	
0	-0	label flag
1	aaaa	first 4 characters of symbiont initiating type-in
2	ddmmyy	date assigned by DECO
3	000001	reel number
4 to 9	≠0	unassigned
10	aaaa	last 4 characters of symbiont initiating type-in
10	-0	label flag

2. Data Blocks

Word	Content
0	Data Descriptor
1	Image of first frame
•	
•	
•	
256	Image of last frame
257	Data Descriptor

3. Bypass and Recovery Point Blocks

These blocks, occurring in the sequence illustrated below, are generated by the type-in SkΔNXT, providing a recovery point.

Bypass Block	-020000000
Recovery Point Block	+0000nn
Bypass Block	-020000000
Bypass Block	-020000000

In the recovery point block, nn is the reel number of the paper tape reel whose data blocks follow on the output file.

BOOT

Bootstrap and System Tape Loader

A. Purpose

To provide a system tape bootstrap routine and a loader to load specified routines from the system tape.

B. Method

This routine has the capabilities of reading routines from the system tape or from binary punched cards. Before loading the executive routine, memory may be preset to instructions which transfer control to an error routine.

1. Bootstrapping

BOOT is loaded from the system tape by pressing the load button when the machine is cleared and the system tape is rewound. This routine includes the system tape search routine. If the run button is now pressed, the executive routine is loaded using the system search routine.

2. System Tape Search

The system tape 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 is found, the system tape is rewound and the search continued through the file a second time until either the specified symbol or the end-of-file is encountered. If the symbol is found, the corresponding program is loaded. If the symbol is not found an error is indicated. Control will be transferred to the starting address of the loaded program if so indicated.

C. Operating Procedures

1. To load the executive routine from the system tape, perform these functions:

   a. REWIND
   b. CLEAR
   c. LOAD
   d. PROGRAM STOP
      Optional. If used will preset memory to SLJ (error routine).
   e. RUN

2. To load binary routine through the card reader after having loaded the executive routine,perform these functions:

   a. REWIND
   b. CLEAR
   c. LOAD
   d. PROGRAM STOP
      Optional. If used will preset memory to SLJ (error routine).
   e. REQUEST
   f. RUN

3. To load and execute routine specified by type-in on console typewriter after loading executive routine, perform the following functions:

   a. REWIND
   b. CLEAR
   c. LOAD
   d. PROGRAM STOP
      Optional. If used will preset memory to SLJ (error routine).
   e. RUN
   f. REQUEST
      Type-in: RX routine name
      routine name must appear exactly as it appears on the system tape.
   g. RELEASE

4. Calling sequence to load and execute specified routine from system tape:

```
LA 3, routine name
J    LODX,
```

5. Calling sequence to load specified routine from the system tape and return control to calling program. Load index register 1 with desired return address.

```
LA 3, symbol desired
J    LOAD
(control returns here)
```

Halts and Loops

a.	00342-00347	busy loop
b.	00351 J $	B error or fault
c.	00333 J $	symbol not found

D. Memory Space

0	J 0302
1-0177	Binary Loader
0100-0147	Scat words for search routine
0200-0217	Tape assignment table
0240-0246	TCD, LOAD, LODX communication cells
0300-0377	Search routine

NOTE: If the BOOT routine is referred to by a symbolic program written in the ALMOST assembly system language, standard EQU cards should be placed ahead of the symbolic program to be assembled and the following labels are restricted from other use in the source program. In this manner, the labels for the BOOT routine will be equated with proper absolute addresses.

LOAD
LODX

WST (WRITE SYSTEM TAPE)

A. Purpose

To create a basic absolute system tape from binary cards.

B. Method

This routine reads binary cards and control cards through the card reader and writes corresponding scat records on the system tape. For each standard binary card, a one word segment corresponding to the first word on the card is written followed by a segment containing the words of information from the remaining portion of the card as specified by the first word. The transfer card and the following routine name card are written as a separate block on the tape. The bootstrap routine is loaded into location 010000 and written therefrom onto tape, followed by the routines read from the binary punched cards in the card reader. Only absolute binary cards may be processed by this routine. The setting of cover registers to their proper values is not handled by this routine and must be accomplished by the routines themselves after they have been read in from the system tape. This routine uses index registers 4 and 5 as cover registers and index registers 1, 2 and 3 for working registers.

Program Modification

1. To change the tape unit to be written on, modify bits 21-24 of cell 0214.

2. To prevent writing the bootstrap routine on the system tape, set the contents of 04334 to zero.

C. Operating Procedure

1. CLEAR

2. Feed one card in card reader

3. RUN

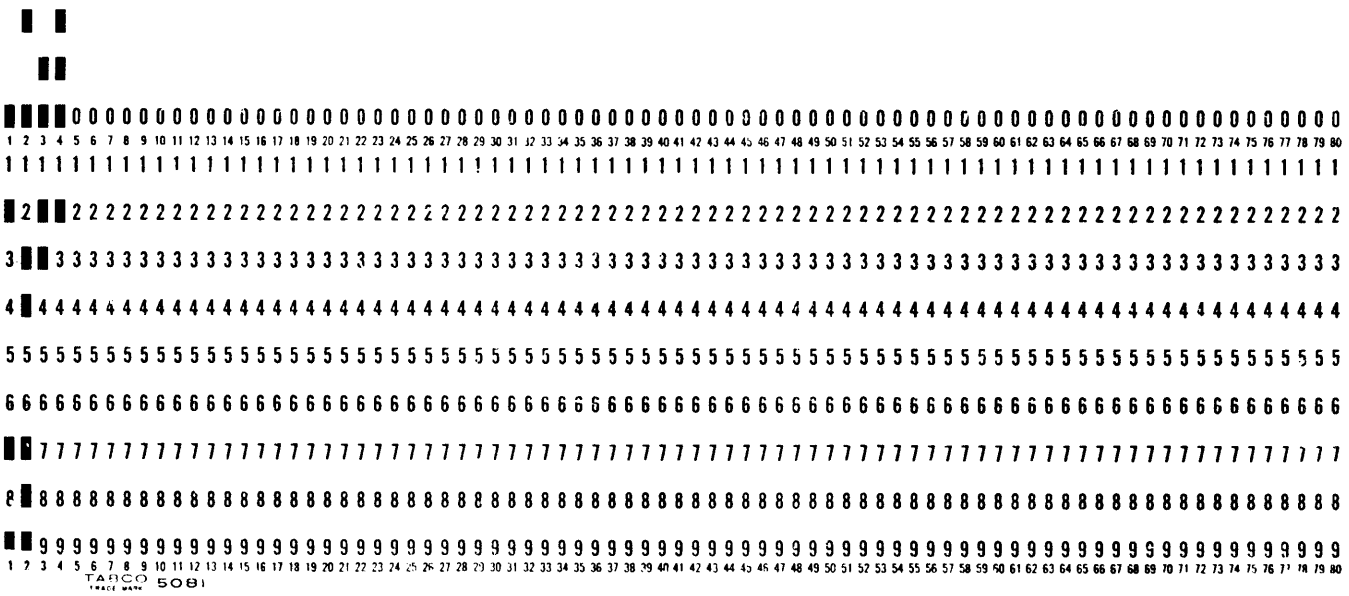
Upon successful completion, the output tape unit will rewind and the WST program will loop. This output tape is then ready for use as the system or program tape. Ordinarily, rings should be removed to inhibit writing on this reel.

D. Memory Space

0 - 0177	binary loader
0200 - 0217	tape assignment table
010000 - 010377	bootstrap program offset from cells 0-0377
04000 - 04363	WST program and constants
04364 - 04435	Scat words to write information blocks
04436 - 06075	card images
06076 - 06121	literals

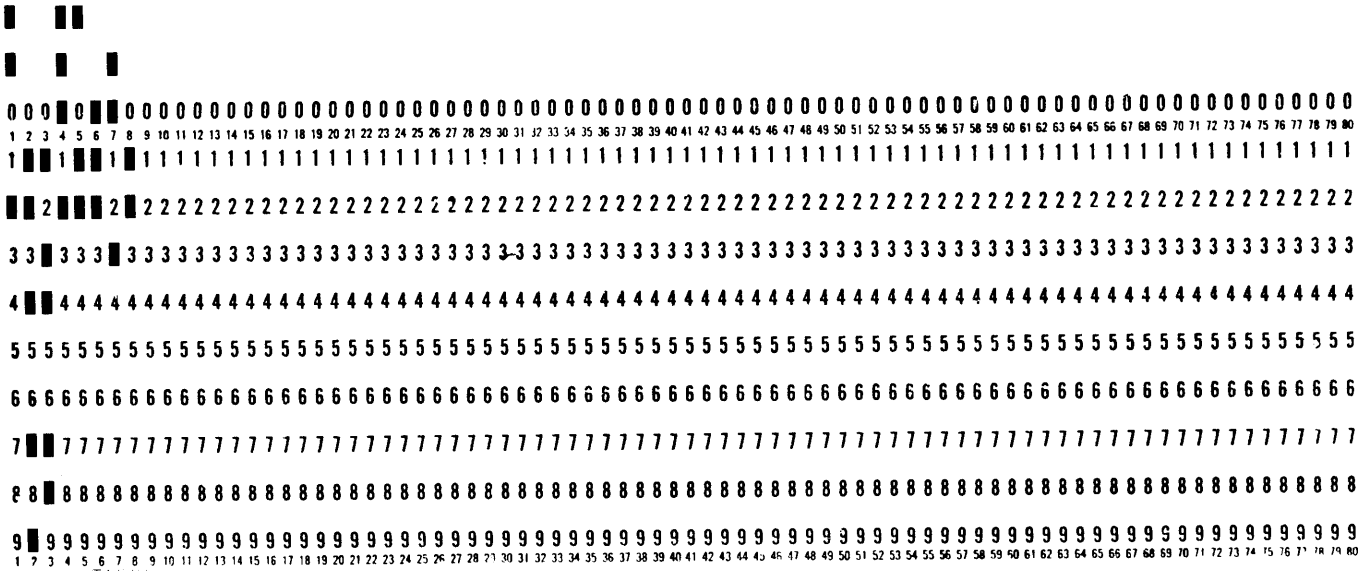
E. Card Formats

1. Routine label card. Binary equivalent of routine label is punched in first two words of card. For example, label for AKTDNW would be punched:



2. Routine program cards. Routine program cards and transfer cards must be in standard binary format. See Section 1-0001 of the SUPPORT III manual for a description of these formats.

3. End-of-file card. Below is a sample of the end-of-file card. It contains the binary equivalent of $\overset{+}{0}$ END-OF-FILE.



UPCO

A. PURPOSE

UPCO (UPdating COntrol) is one of the three General Program Processors associated with the BOSS III SUPPORT System. Its purpose is to provide updating services for the user's source code and object code libraries.

B. INTRODUCTION

UPCO may be used in three major area – in creating symbolic code or relocatable object code libraries, in updating libraries, and in creating control tapes.

A new library tape is created by UPCO by including the desired library information with the control input. In this case there would be no additional library input tapes to UPCO, and there would be no corrections, insertions, or deletions. A new library tape could also be created by using the PRESTO card-to-tape symbiont.

UPCO can create an updated library tape by selecting information from one or more old library tapes. Control input directs the various processes of insertion, deletion, and correction, and the control input may itself contain information to be included on the updated library tape. The updated tape may then be used as library input or as a control tape for another processor (ACCO, DECO, or UPCO itself).

1. Input

Input to UPCO consists of the user's relocatable library or symbolic library tape, which is to be updated, and control information on cards or on a separate tape which directs the updating process. All tape input to UPCO is in the condensed PRESTO format, which means that all consecutive blanks and zeroes have been removed. None of the original information content is lost

a. Library Tapes

A library tape may contain sets of independently compiled relocatable object code, or source code, arranged in groups and elements. Elements contain either source code images (COBOL, FORTRAN, or UTMOST), or binary card images. In either case, there may be control cards within the element.

b. Control Information

Control information defines and directs the updating of the relocatable or the symbolic library tapes, and is contained either on a tape in PRESTO format, or on cards, but not both.

Control input comes from a previous processor (ACCO or UPCO), the PRESTO card-to-tape symbiont, or directly from cards. In addition to control card images, the control input may also contain symbolic or binary elements.

See the section on Control Cards for an explanation, in detail, of the various types of control cards.

2. Output

a. Library Tape

The library tape (PRESTO format) which is produced by an UPCO run will be an updated symbolic or a relocatable library tape, containing updated and/or new information. This tape may be used as input to another processor (ACCO, DECO, or UPCO itself).

b. List/Punch Tape

A List/Punch tape is optional in an UPCO run, and must be specified by a tape assignment parameter card if the user wants it. If it is not so designated, all listing and/or punching will be done on line.

The information which UPCO places on the tape is in the format required by the combined PRINT/PUNCH tape symbiont (CPPS). This tape could be printed and/or punched later by CPPS concurrently with a main program.

There are a number of options available to the user which will control the mode of listing and punching, and the mode can be changed for each element or group within an UPCO run. For punching, there are only two options - punch or no punch. For listing, options available allow the printing of selected information such as diagnostics, error messages, control cards, or a combination of these.

The Options in a MODE control card (or in a COPY control card) determine the printing/punching mode. Once a mode is initiated, it will remain in effect until changed by another MODE or COPY control card. If a mode of listing and punching is not stated, there will be no punching, and the listing will be as specified in the LNOR option of the MODE card.

List/Punch options will be particularized in the Control Card Section.

3. Groups and Elements

A program (job) contains a combination of groups and elements – or a program may consist of only one group or element. An element is the smallest program unit and may contain binary images or source code images, including control cards. A group is a collection of elements or other groups, and may contain control cards, but may not contain source code or binary images unless they are within an element.

Groups and elements are identified by names, which must be unique within a group. Names may not exceed eight characters.

Groups and elements are identified by the following control cards:

BOG	Groupname	(Beginning of group and its name)
EOG	Groupname	(End of group and its name)
ELT	Elementname	(Beginning of element and its name)

On a library tape an element is terminated by the appearance of another ELT, a BOG, an EOG card, or an end-of-file marker.

The BOG and EOG control cards serve as brackets around a section of coding; they are descriptive labels for a group. All other control cards pertinent to a group must appear within the BOG and EOG control brackets. Groups may be nested within groups, as the following example indicates.

```
BOG   GRP1    Group 1 (GRP1) contains groups 2, 3, and 4.
BOG   GRP2    Group 2 (GRP2) contains groups 3 and 4.
BOG   GRP3
EOG   GRP3
BOG   GRP4
EOG   GRP4
EOG   GRP2
EOG   GRP1
```

An ELT control card precedes the card images belonging to the element named in its operand field (Elementname). To be part of a group, elements must be contained within BOG and EOG control cards, and within a group an ELT card separates succeeding elements. Elements which are not within a group are terminated by the next BOG, ELT card, or end-of-file marker encountered.

The following is an example of elements within nested groups:

```
BOG   GRP1    Group 1 (GRP1) contains element 1 (ELT1)
ELT   ELT1              and groups 2, 3, and 4.
BOG   GRP2    Group 2 (GRP2) contains groups 3 and 4. GRP2
                        does not contain source code state-
                        ments in itself.
BOG   GRP3    Group 3 (GRP3) contains elements 2 and 3.
ELT   ELT2
ELT   ELT3
EOG   GRP3
BOG   GRP4    Group 4 (GRP4) contains element 4.
ELT   ELT4
EOG   GRP4
EOG   GRP2
EOG   GRP1
```

To call ELT3 in the preceding example:

```
SELECT GRP1(GRP2(GRP3(ELT3)))
```

or, to call ELT1:

SELECT GRP1 (ELT1)

or, to call GRP3:

SELECT GRP1(GRP2(GRP3))

C. CONTROL CARDS

1. Control Cards are used by programmers and operators to communicate with the BOSS III SUPPORT System. An UPCO control card has a free-form format similar to a line of UTMOST symbolic coding. The exceptions are that it must contain a 12-0-2 punch (nonstandard punch and for use in illustrations in the text the symbol ≠ will be used) (Z) in the first column, the label field may begin in any column following the Z punch, and the line may not contain comments.

≠ LABELΔOPERATIONΔOPERAND

(where Δ indicates at least one blank).

If the label field of a control card contains the name of a processor, the control card is then made "transparent" to all other processors. For example, ≠UPCOΔMODEΔLNOR,PUNCH. This control card may be passed through ACCO or DECO with no action being taken until it is processed by UPCO. The designation of a processor is the only use for the label field in any control card.

The operation field contains a system directive which specifies a particular function to be performed by the processor. The operand field contains one or more parameters which predicate the operation of the system directive. The operation field and any parameters may be up to eight characters in length.

Control cards for UPCO will not be passed on once that processor has acted upon them. The same is true for any processor, excepting control cards which define jobs, elements, groups, and data areas, such as ELT, BOG, EOG, JOB, SEG, and DATA.

The control cards for UPCO may be separated into certain divisions according to their function, as indicated by the following descriptions. Control cards listed in this section will have UPCO in the label field if the cards are also applicable to ACCO or DECO.

a. Servo Assignment Control Card

UPCO SERVO

When running under SUCO, all tape assignments for each UPCO run must be specified by the use of a SERVO control card followed by tape assignment parameter cards. This SERVO card must be the first control card in an UPCO run, and there may be only one SERVO card per job. Any SERVO cards encountered after the first one will cause an error message.

It should be noted that when the UPCO FINIS card specifies a successor run, the input tape to UPCO (file #1) will also be the input tape to the successor processor until an ASSIGN card is found which changes it. In this case, duplicate SERVO cards and associated tape assignment parameter cards will be generated on the UPCO output tape. These duplicate images will be ignored by the successor processor.

The following is an example of the use of SERVO cards and tape assignment parameter cards in a run using all three processors, where execution will be in UPCO→ ACCO→ DECO order.

```
Ƶ DECO          SERVO
          1       INEX          Control input tape
DECOUT    3       OUTPUT        New system tape
          4       SCRACH        Scratch tape
          8       INEX          Reloc. obj code library
ACOUT             ASSIGN  3, 1
Ƶ ACCO          SERVO
          1       INEX          ACCO control input
UPIN      6       INPUT         UPCO control input
ACOUT     3       OUTPUT        Relocatable ACCO output
SLIBRY    9       INPUT         Source code library
          4       SCRACH        Scratch tape
UPOUT             ASSIGN  3, 1
ACOUT             ASSIGN  6, 3
ACOUT     3       SAVE
Ƶ UPCO          SERVO
UPIN      1       INPUT         Control input tape
UPOUT     3       OUTPUT        UPCO output (PRESTO)
SLIBRY    9       INEX          Source code library
UPOUT     3       SAVE
ACOUT     6       SAVE
SLIBRY    9       SAVE
UPIN      1       SAVE
```

The control information for all three processors is contained on one tape (file #1). Control input for ACCO and DECO is copied from this tape onto the UPCO output tape (UPOUT). ACCO obtains its tape assignment information from the original control input which is still file #1. These tape assignments cause UPOUT to become the control input (on file #1) for ACCO, and the blank tape on file #6 in UPCO now becomes ACCO output (ACOUT) on file #3. This process is repeated between ACCO and DECO, in that control information for DECO is copied from UPOUT onto ACOUT, and ACOUT becomes the control input tape for DECO. UPOUT now becomes the DECO output tape, which will be the new system tape.

Dismounting instructions for the original control input tape (UPIN) will be given at the end of the ACCO run, and there will be dismounting instructions for the new system tape at the end of the DECO run.

b. Tape Designation Control Card

UPCO TAPE File Number, Label

The TAPE control card specifies which file number is to be used as the library tape. File number is a decimal number (0 through 15) which indicates an entry in the tape assignment table. For example: TAPEΔ6 specifies the seventh entry in the tape assignment table. The label of the library tape will be matched against Label shown in the control card, and an error message will be typed out if they do not match. If Label is blank, UPCO assumes there is no tape label, and if there is one, it will be ignored.

A TAPE card may be used whenever it is needed for specification of a new library tape.

Each file designated in a TAPE card should have been previously defined by a tape assignment parameter card. An example of this is:

```
ZUPCO          SERVO
LIBRY   07       INPUT   (Tape assignment parameter card)
ZUPCO          TAPE    07
    (balance of cards)
```

If a TAPE card names a file which has not been previously defined, a message will be typed out, and processing will continue as soon as GO is typed in.

c. Processor Termination Card

UPCO FINIS Name

FINIS indicates the end of the UPCO run; Name indicates the processor or run to be entered next. Normally, Name would be DECO or ACCO, providing automatic transfer to the desired processor. Name would be left blank if the user is processing through UPCO only. In this case, at the end of the UPCO run, SUCO will type out a message ("NEXT") upon reaching end of job, and will spin in a stop loop until the operator calls a new program.

d. Mode of Listing/Punching

UPCO MODE List option, Punch option

The MODE control card indicates to the processor the type of information that is to be listed or punched, whether on line or off-line.

For the List option, one of the following should be entered:

LDNT - Diagnostics and error messages

LCTL - Control cards

LNOR - Revised information

LCOR - Revised information with the corrected (superseded) images flagged.

LDTL - Detailed information (includes those groups and elements which had no corrections, as well as those which did.)

For the Punch option,

PUNCH - punch detailed information

NO PUNCH - punch nothing

Once a mode is initiated, it will remain in effect until changed by

another MODE (or COPY) control card. If a mode of listing and punching is not stated, there will be no punching, and the listing will be as specified in the ~~LNOR~~ LIST option of the MODE card.

e. Control Cards for Deletion or Copying without Correction

(1) DELETE

DELETE Name

The DELETE card causes UPCO to copy from the current position of the tape up to, but not including, Name (a group or element name). At the completion of the DELETE operation, the tape will be positioned immediately beyond Name.

(2) COPY

COPY Name, List option, Punch option

This control card causes UPCO to copy from the current location of the library tape through the end of the specified group or element.

List and Punch options are the same as those indicated for the MODE card. The mode of listing and punching in effect at the time the COPY card is encountered will be used for listing and punching all information copied, up to Name. Name will then be listed and punched in the mode indicated in the COPY control card. This mode will remain in effect until a new mode is established by another control card (MODE or another COPY).

(3) SELECT

UPCO SELECT Name

SELECT is similar to COPY. It, however, does not copy the information from the old library tape which appears prior to Name, but instead skips the library tape until it finds Name, and then copies only this element or group.

f. Control Cards which Position a Library Tape for Correction

There are two control cards in this classification: one does only positioning, and the other causes a copy down to a specific position. These two control cards are normally followed by cards which indicate corrections or insertions. In each case, if New name is specified, it causes the title of the element or group to be changed from Old name to New name on the output library tape.

If New name has the title MATE, it causes the contents of Old name to become part of the preceding group or element. The original group or element name will not exist on the new system tape.

(1) FIND

UPCO FIND Old name, New name

FIND causes the library to be positioned (without copying) at the beginning of Old name. If Old name is an element, FIND may now be followed by COR or INS control cards.

(2) CHANGE

CHANGE Old name, New name

CHANGE causes the library tape to be copied from its current position to a position at the beginning of Old name. If Old name is an element, CHANGE may be followed by COR or INS cards.

g. Line Correction or Insertion Cards

These cards serve to permit corrections or insertions to individual lines in an element. COR and INS cards will normally follow a control card which has positioned the tape at the beginning of an element.

(1) COR

UPCO COR n_1, n_2

where n_1 and n_2 are line numbers on a listing of the element to be corrected.

COR will cause the element to be copied until line n_1 is found. Lines n_1 through n_2 will be deleted, and any non-control cards following the COR card will be inserted at this point. If n_2 is not specified, then only line n_1 will be deleted. The effect will be just as if the control card COR n_1, n_1 had been specified.

INSΔΔΔΔ may be used to terminate corrections if the remainder of the element being corrected is to be copied.

(2) INS

UPCO INS n

INS causes the element to be copied down through and including line n, and any non-control cards will be inserted at this point.

INSΔΔΔΔ will cause the information following it to be inserted after the last line of the current element. This form is convenient for inserting relocatable binary corrections, as it eliminates any need to know the count of the binary images.

INSΔΔΔΔ may also be used to terminate corrections or insertions to an element if the remainder of the element is to be copied.

2. Tape Assignment

Tape assignment parameter cards are placed with the beginning parameter information for a run. They are condensed by DECO and are written on the system tape as part of the JOB preamble. During the initialization of a run by SUCO, they are examined and appropriate action is taken. Tape assignment parameter cards, as they apply to a particular job, are transparent to UPCO, and will merely be copied onto the output system tape for another processor.

When running under SUCO, all tape assignments for each UPCO run must be specified by the use of a SERVO control card followed by tape assignment parameter cards.

The format of the tape assignment parameter card does not comply with rules for other control cards. The format is not variable:

Columns	Entry
1 - 6	alias
7	blank or comma
8 - 9	File number, right justified
10	blank or comma
11 - 16	Operation
17	blank
18 . . .	One to three-digit assignment numbers right-justified in columns 20, 24, 28..., and separated by commas or blanks.

The file alias has no logical attachment to any symbols generated by a program, and is carried as a mnemonic device only. Its sole use is on tape assignment parameter cards and on correspondingly generated tape mounting. posting, and dismounting instructions via the console typewriter.

The File number (columns 8 and 9) is a decimal number (0 through 15) which specifies an entry in the tape assignment table. For example, file number 10 specifies the eleventh entry in the tape assignment table. It is not necessary to precede with zeroes.

The function of the assignment numbers (columns 18...) depends upon the particular operation involved. All assignment numbers are decimal.

Tape assignment cards should appear at the beginning of the job to which they apply.

For a description of operator messages which might appear on the console typewriter as a result of the following parameter cards, refer to the BOSS III Programmers Guide.

a. ASSIGN

alias ASSIGN k1, k2

The file entry k1 from the previous run will be assigned to file entry k2 of the current run. This is accomplished by interchanging logical unit numbers between the two file entries in the tape assignment table. A check is made to see if the previous alias for k1 agrees with the alias on the ASSIGN card, and if not, an error message is produced. To ignore the ASSIGN card, type in GO.

The rewind-with-interlock provisions of INEX, OUTPUT and SCRACH will not apply to a tape which has been saved with a SAVE card, and subsequently assigned with an ASSIGN. Neither will there be mounting and dismounting instructions.

b. INPUT

alias k INPUT n

INPUT describes file k as being a protected input file, and causes a beginning-of-job mounting message and an end-of-job dismounting message. File k may not have been rewound with interlock at the end of the previous job. If not, INPUT will cause rewind with interlock at the beginning of the job in which it occurs. (If a tape has been "saved" and subsequently "assigned", file k will only be rewound, and there will be no mounting or dismounting instructions.) n specifies the expected number of reels for file k, thereby permitting an early release of the alternate, if any. An incorrect n will not cause an error.

c. INEX

alias k INEX n

INEX describes file k as being an unprotected input file, and causes a beginning-of-job mounting message only. File k may not have been rewound with interlock at the end of the previous job. If not, INEX will cause a rewind with interlock at the beginning of the job in which it occurs. (If a tape has been "saved" and subsequently "assigned", file k will only be rewound, and there will be no mounting or dismounting instructions.) n specifies the expected number of reels for file k, thereby permitting early release of the alternate, if any. An incorrect n will not cause an error.

d. OUTPUT

alias k OUTPUT

OUTPUT describes file k as being a protected output file. File k may not have been rewound with interlock at the end of the previous job. If not, OUTPUT will cause rewind without interlock at the beginning of the job in which it occurs. In either case, a MOUNT BLANK message is produced. An end-of-job dismounting message will be typed out. If a tape has been "saved", a dismounting message will not occur in the job in which the tape was saved, and only rewind will occur in the job which assigned the tape.

e. SCRACH

~~alias~~ k SCRACH

This describes file k as being a scratch tape. File k may not have been rewound with interlock at the end of the previous job. If not, SCRACH will cause rewind without interlock at the beginning of the job in which it occurs. If the previous job did rewind with interlock, a MOUNT BLANK message is produced at beginning-of-job. If a tape has been "saved" and subsequently "assigned", the reel will be rewound only and there will be no mounting messages.

f. ALT

alias k ALT k1, k2,...

This describes file k as being an alternate to files k1, k2,... If k1 is an input file, then there should only be the entry k1 in the list. If k1 is an input reel, then a MOUNT message will be produced and the unit rewound with interlock if it is not dismounted. If k1 is an output reel, then a MOUNT BLANK message will be produced if the unit is dismounted.

g. SAVE

alias k SAVE

SAVE specifies that file k be carried over to the next run. If file k 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 a MOUNT message and rewind with interlock, if appropriate.

A tape which has been "saved" must be assigned (with an ASSIGN card) in the succeeding job.

h. DUMP

k DUMP

DUMP specifies that file entry (k) is the system dump tape. A dump tape must be specified for each main program which will be run in conjunction with symbionts, or which will use rerun. It may be any output tape which employs standard tape conventions, i.e., has been written under control of the Tape File Label Handling routine.

D. OPERATIONAL CONTROL

1. Nominal Tape Assignment

The following tape assignments are used for UPCO. They may not be changed if UPCO is called by an RXΔUPCO type-in. However, when UPCO is under control of SUCO, the assignments may be changed once in an UPCO run by the use of a SERVO card followed by appropriate tape assignment cards.

File No.	Usage
0	System Tape
1	Basic PRESTO input
2	Print/punch tape
3	PRESTO output
4	Library tape
5	Library tape
6	Library tape
.	.
.	.
.	.
n	Library tape

2. Console Functions

a. SUCO Control (for automatic tape operation)

1. CLEAR
2. REWIND
3. LOAD
4. PROGRAM STOP - sets all memory to SLJ ERR.
5. PROGRAM RUN
6. KEYBOARD REQUEST
7. Type in RSΔCALLΔUPCOΔ2
8. KEYBOARD RELEASE (Activated UPCO)

b. EXEC Control (for card operation)

1. CLEAR
2. REWIND
3. LOAD
4. RELEASE (This causes EXEC control.)
5. PROGRAM STOP - sets all memory to SLJ ERR.
6. PROGRAM RUN
7. KEYBOARD REQUEST
8. Type in RXΔUPCO
9. KEYBOARD RELEASE (Activates UPCO)

ACCO

A. PURPOSE

The purpose of ACCO (Assembler-Compiler COntrol), one of the three General Program Processors associated with the BOSS III SUPPORT System, is to direct the processing of the user's source code through UTMOST, COBOL, and FORTRAN, and to provide a standard input-output processing for assembling and compiling.

B. INTRODUCTION

ACCO permits high efficiency for assembly and/or compilation since a number of source code routines, written in UTMOST, FORTRAN, and/or COBOL (and stacked on one input tape), can be assembled or compiled in one run.

There are essentially two ACCO "passes" for each assembly or compilation requested. (And within ACCO's second pass are the multiple passes or phases of the particular assembler or compiler.) In its initial pass, ACCO will perform library search and copy with corrections for the COBOL, FORTRAN, or UTMOST routine. Control cards for other processors, information included on the control input, or information extracted from library tapes outside the control of the assembler or compilers will be copied directly onto ACCO's output tape during this initial pass. An intermediate scratch tape is used, on which the source code for the assembly or compilation is accumulated.

When the end of the source code to be compiled or assembled is indicated, ACCO will rewind the intermediate scratch tape and enter its second pass (assembly or compilation). As the relocatable object code is generated by UTMOST, COBOL, or FORTRAN, it will be written onto the output tape. On completion of this process, control will return from the compiler or assembler to ACCO, which will continue to process the next sequential information on its input tape.

Thus, the final output includes control input not recognized by ACCO, information not processed by ACCO, and also the compiled or assembled relocatable object code.

1. Input

ACCO will accept three types of input: control information, source code images, and relocatable object code. Relocatable object code input will be copied directly onto the relocatable output tape, with no action being taken on it during the ACCO run. All tape input to ACCO is in the condensed PRESTO format, which means simply that all consecutive blanks and zeroes have been removed. None of the original information content is lost.

a. Control Input

Control information (on either tape or cards) defines and directs the processing of a source code program. Control card images contain the programmer's instructions to the processors for copying tape, for changing or deleting specified code, and for assembling or compiling specified source code.

Control input comes from a previous processor (UPCO), from the PRESTO card-to-tape symbiont, or directly from cards. The control input may contain some or all of the source code, and could even contain binary elements which will simply be passed on to the relocatable output tape.

See the section on Control Cards for an explanation, in detail, of the various types of control cards and their functions.

b. Library Tapes

Library tapes may contain numerous sets of source code or relocatable object code programs or subprograms arranged in elements and groups. Control information may appear within an element or group, and will be processed in the order in which it is encountered.

The use of the library tape input is optional with the user, and the number of library tapes used in an ACCO run is limited only by the number of available tape units. Only one library tape is available to ACCO at a given time, the selection being under control of the user by means of a TAPE control card.

2. Output

a. Relocatable Object Code Tape

This output tape (in PRESTO format) may contain a combination of the following: control card images for other processors, relocatable object code copied from a library tape or the control input, and relocatable object code output of the various assemblies or compilations which may have taken place during the ACCO run.

b. List/Punch Tape

A List/Punch tape is an optional output from an ACCO run. It is specified by including a tape assignment parameter card for file 2 with the other ACCO tape assignment cards. If the tape is not so designated, all listing and/or punching will be done on line.

If a List/Punch tape is designated, the information which ACCO places on it is in the format required by the PRINT/PUNCH tape symbiont (CPPS). This tape could be printed and/or punched later by CPPS, concurrently with a main program.

There are a number of options available to the user which will control the mode of listing and punching. The mode can be changed for each assembly or compilation. For punching, there are only two options - punch or no punch. For listing, options available allow the printing of selected information, such as diagnostics, error messages, control cards, source code language, or a combination of these.

The options given in a MODE, UTMOST, FORTRAN, or COBOL control card determine what information is to be printed or punched. Once a mode is initiated, it will remain in effect until changed by a mode option in another control card. If a mode of listing and punching is not stated, there will be punching, and the listing will be as specified in the LNOR option. List options will vary according to the particular assembler or compiler being used. The various options are listed in the section on Control Cards, under MODE, COBOL, FORTRAN, and UTMOST.

3. Groups and Elements

A program (job) contains a combination of groups and elements - or a program may consist of only one group or element. An element is the smallest program unit; a group is a collection of elements or other

groups. An element contains source code images or relocatable code, and may also contain control cards. Groups may contain control cards, but may not contain source code images or relocatable object code except as part of an element.

Groups and elements are identified by names, which must be unique within a group. Names may not exceed (8) characters.

Groups and elements are identified by the following control cards:

BOG	Groupname	(Beginning of group and its name)
EOG	Groupname	(End of group and its name)
ELT	Elementname	(Beginning of element and its name)

On a library tape an element is terminated by the appearance of another ELT, a BOG, or an EOG card.

The BOG and EOG control cards serve as brackets around a section of coding. These cards are descriptive labels for a group, and should be the first and last cards of any group. All other control cards pertinent to a group must appear within the BOG and EOG control brackets. Groups may be nested within groups, as the following example shows.

```
BOG   GRP1    Group 1 (GRP1) contains groups 2, 3, and 4.
 BOG  GRP2    Group 2 (GRP2) contains groups 3 and 4.
  BOG GRP3
  EOG GRP3
  BOG GRP4
  EOG GRP4
 EOG  GRP2
EOG   GRP1
```

An ELT control card precedes the card images belonging to the element named in its operand field (Elementname). To be part of a group, elements must be contained within BOG and EOG control cards. Elements are terminated by the next BOG or ELT encountered. The following is an example of elements within nested groups:

```
BOG   GRP1    Group 1 (GRP1) contains element 1 (ELT1)
ELT   ELT1    and groups 2, 3, and 4.
BOG   GRP2    Group 2 (GRP2) contains groups 3 and 4.
                GRP2 does not contain source code state-
                ments in itself.
  BOG   GRP3  Group 3 (GRP3) contains elements 2 and 3.
  ELT   ELT2
  ELT   ELT3
  EOG   GRP3
  BOG   GRP4  Group 4 (GRP4) contains element 4.
  ELT   ELT4
  EOG   GRP4
EOG   GRP2
EOG   GRP1
```

To call ELT3 in the preceding example:

SELECT GRP1(GRP2(GRP3(ELT3)))

or, to call ELT1:

SELECT GRP1 (ELT1)

or, to call GRP3:

SELECT GRP1(GRP2(GRP3))

For use of Alias error - see 5-0005, pg 4

C. CONTROL CARDS

1. Control cards are used by programmers and operators to communicate with the BOSS III SUPPORT System. An ACCO control card has a free-form format similar to a line of UTMOST symbolic coding. The exceptions are that it must contain a 12-0-2 punch (Ƶ) in the first position, the line may not contain comments, and the label field may begin in any column after the first.

Ƶ LABELΔOPERATIONΔOPERAND

where Δ indicates at least one blank.

If the label field contains the name of a processor, the control card is then made "transparent" to all other processors. For example, ƵACCOΔMODEΔLNOR,PUNCH. This card may be passed through UPCO or DECO with no action being taken until it is processed by ACCO. The designation of a processor is the only use for the label field in any control card. (ELT, BOG, or EOG may be made transparent to ACCO by using an UPCO or DECO label. This will indicate to ACCO the end of an assembly or compilation.)

The operation field contains a system directive which specifies a particular function to be performed by the processor.

The operand field contains one or more parameters which define the operation of the system directive. The operation field or any of the parameters may be up to eight characters in length.

Many of the control cards shown in the following discussions are universal, i.e., they are also applicable to DECO or UPCO. These are indicated here by the appearance of ACCO in the label field.

a. UTMOST

UTMOST, List option, Punch option

The UTMOST control card is a request for an UTMOST assembly. It precedes the source code (or the control cards that call the source code) which comprises the input to be assembled. The options indicate to the processor the type of information to be listed and punched. If a List option is not entered, the listing will be the same as for the LNOR option. Punching will occur if a Punch option is not given.

Available List options:

LNOR - Normal assembly listing (symbolic and relocatable object code representation).

LCOR - Deleted or corrected lines listed in front of assembly output in addition to COR lines.

LDTL - Same as LCOR

LDNT - No listing.

LCTL - Control cards only.

Available Punch options:

PUNCH - Relocatable object code

NO PUNCH - No punching

b. COBOL

COBOL List option, Punch option, Class

The COBOL control card is a request for a COBOL compilation. It precedes the source statements (or the control cards that call the source code) which make up the input to be assembled. The options indicate the type of information to be listed and punched. The listing will be as for the LNOR option if a List option is not entered in the COBOL card. If a Punch option is not specified, punching will occur.

Available options for listing:

LNOR - Source language listing and diagnostics.

LCOR - Deleted or corrected lines listed in front of source language.

LDTL - Complete UTMOST listing of generated code plus LCOR output.

LDNT - Diagnostics.

LCTL - Control cards.

Options for punching:

PUNCH - Relocatable object code

NO PUNCH - No punching

There are two available options for Class:

(1) MAIN - Indicates a main program for which there are independently compiled subprograms.

(2) SUB - Indicates a subprogram.

c. FORTRAN

FORTRAN List option, Punch option

The FORTRAN control card is a request for a FORTRAN compilation. It precedes the source code (or the control cards that call it) which comprises the input to be compiled. The options indicate the type of information to be listed and punched. The listing will be as for the LNOR option if a List option is not entered in the FORTRAN card. If a Punch option is not specified, punching will occur.

Available options for listing:

LNOR - Source language listing and diagnostics.

LCOR - No listing.

LDTL - Source language plus an UTMOST-like edited output.

LDNT - No listing.

LCTL - Control cards.

Options for punching:

PUNCH - Relocatable object code

NOPUNCH - No punching

d. SELECT

SELECT Library name

A library tape which has been specified by a preceding TAPE card will be searched for Library name. The information within Library name (a group or element) is then given to the processor for assembling or compiling.

e. FIND

FIND Library name

A library tape which has been specified by a preceding TAPE card will be searched for Library name. FIND may now be

followed by COR or INS control cards. The FIND control card will enable a user to correct an element prior to compiling or assembling.

f. COR

ACCO COR n_1, n_2

where n_1 and n_2 are line numbers on a listing of the element to be corrected.

The COR control card must follow a FIND control card, and will cause the element to be copied until line n_1 is reached. Lines n_1 through n_2 will be deleted, and any non-control cards following the COR card will be inserted at this point. If n_2 is not specified, then only line n_1 will be deleted. The effect will be just as if the control card COR n_1, n_1 had been specified.

g. INS

ACCO INS n

INS n causes the element to be copied down through and including line n, and any non-control cards will be inserted at this point. INS must follow a FIND card. INSΔΔΔΔ will cause the information following the INS card to be inserted after the last line of the current element. This form may be used to insert relocatable binary corrections, and it eliminates any need to know the count of the binary images.

INSΔΔΔΔ may also be used to terminate corrections and insertions to an element, if the remainder of the element is to be copied.

h. MODE

ACCO MODE List option, Punch option

The MODE control card indicates to the processor the type of information that is to be listed or punched, whether on line or off line.

For the List option, one of the following should be entered:

LDNT - Diagnostics and error messages.

LCTL - Control cards.

LNOR - Normal source language listing.

LCOR - Deleted or corrected lines listed in front of normal output.

LDTL - Detailed information.

For the Punch option:

PUNCH - Punch relocatable object code

NO PUNCH - Nothing is punched

Once a mode is initiated, it will remain in effect until changed by another MODE, UTMOST, FORTRAN, or COBOL control card. If a mode of listing and punching is not stated, there will be punching, and the listing will be as specified in the LNOR option.

i. JOB, DATA, SEG, and CHAIN

JOB Job name

DATA Location ± Increment

SEG Segment name, Origin

CHAIN Link name

These are DECO control cards, but when ACCO encounters them, they will cause termination of an assembly or compilation. These cards will be copied onto the ACCO output tape.

j. ELT, BOG, EOG

ELT Element name

BOG Group name

EOG Group name

Any of these control cards encountered within source code being assembled or compiled will be deleted. ~~If encountered outside such a source code set, they will be copied onto the ACCO output. If any of these cards contain a processor name (UPCO or DECO) in the label field, they will cause termination of assembly or compilation, and will be copied onto the output tape.~~

```
Z           ELT         MNCHAIN
Z     ACCO  TAPE        7,SCLIBE
Z           UTMOST
Z     ACCO  SELECT      GRPONE (ELTTWO)
Z     ACCO  SELECT      GRPTHREE
Z     DECO  ELT         SUBONE
Z           UTMOST
Z     ACCO  SELECT      GRPTWO (ELTFOUR)
Z     ACCO  FINIS
```

In this example, the ELT MNCHAIN card image will be written onto the output tape followed by the object code created during the assembly of GRPONE(ELTTWO) and GRPTHREE. Any BOG, EOG or ELT card images encountered during this selection process are deleted. The DECO ELT SUBONE card image indicates to ACCO the termination of the input for assembly. This ELT card is also copied onto the output tape followed by the resultant object code of the second assembly. This second assembly is terminated by the FINIS card image.

k. SERVO

ACCO SERVO

When running under SUCO, all tape assignments for each ACCO run must be specified by tape assignment parameter cards following a SERVO control card. This SERVO card must be the first control card in an ACCO run, and there may be only one SERVO card per job. Any SERVO cards encountered after the first one will cause an error message.

l. TAPE

ACCO TAPE File number, Label

The TAPE control card specifies which file number is to be used as the library tape. File number is a decimal number (0 through 15) which specifies an entry in the tape assignment table. For example, TAPEΔ6 specifies the seventh entry in the tape assignment table. If Label (an eight-character identification) is present, the label of the specified library tape will be matched against this label, and an error message typed out if they do not match. If Label is blank, ACCO assumes there is no label, and if there is one, it will be ignored.

A TAPE card may be used whenever it is desired to specify a new library tape. Whenever it is used, the specified file will be rewound.

Each file designated in a TAPE card must have been previously defined by a tape assignment parameter card. An example of this is:

```
Z  ACCO        SERVO
     LIBRY 07    INPUT
Z  ACCO        TAPE        07
     (balance of cards)
```

m. FINIS

```
ACCO  FINIS  Name
```

The FINIS card must be used to terminate an ACCO run. Name indicates the processor or run to be entered next. Normally, Name would be UPCO or DECO, providing automatic transfer to the desired processor. Name will be left blank if the user is processing through ACCO only. In this case, at the end of the ACCO run, SUCO will type out a message ("NEXT") upon reaching end of job, and will spin in a stop loop until the operator calls a new program.

2. Tape Assignment

Tape assignment parameter cards are placed with the beginning parameter information for a run. They are condensed by DECO and are written on the system tape as part of the JOB preamble. During the initialization of a run by SUCO, they are examined and appropriate action is taken. Tape assignment parameter cards, as they apply to a particular job, are transparent to UPCO, and will merely be copied onto the output system tape for another processor.

When running under SUCO, all tape assignments for each UPCO run must be specified by the use of a SERVO control card followed by tape assignment parameter cards.

The format of the tape assignment parameter card does not comply with rules for other control cards. The format is not variable:

Columns	Entry
1 - 6	alias
7	blank or comma
8 - 9	File number, right justified
10	blank or comma
11 - 16	Operation
17	blank
18 . . .	One to three-digit assignment numbers right-justified in columns 20, 24, 28..., and separated by commas or blanks.

The file alias has no logical attachment to any symbols generated by a program, and is carried as a mnemonic device only. Its sole use is on tape assignment parameter cards and on correspondingly generated tape mounting, posting, and dismounting instructions via the console typewriter.

The File number (columns 8 and 9) is a decimal number (0 through 15) which specifies an entry in the tape assignment table. For example, file number 10 specifies the eleventh entry in the tape assignment table. It is not necessary to precede with zeroes.

The function of the assignment numbers (columns 18...) depends upon the particular operation involved. All assignment numbers are decimal.

Tape assignment cards should appear at the beginning of the job to which they apply.

For a description of operator messages which might appear on the console typewriter as a result of the following parameter cards, refer to the BOSS III Programmer's Guide.

a. ASSIGN

alias ASSIGN k1, k2

The file entry k1 from the previous run will be assigned to file entry k2 of the current run. This is accomplished by interchanging logical unit numbers between the two file entries in the tape

assignment table. A check is made to see if the previous alias for k1 agrees with the alias on the ASSIGN card, and if not, an error message is produced. To ignore the ASSIGN card, type in GO.

The rewind-with-interlock provisions of INEX, OUTPUT and SCRACH will not apply to a tape which has been saved with a SAVE card, and subsequently assigned with an ASSIGN. Neither will there be mounting and dismounting instructions.

b. INPUT

alias k INPUT n

INPUT describes file k as being a protected input file, and causes a beginning-of-job mounting message and an end-of-job dismounting message. File k may not have been rewound with interlock at the end of the previous job. If not, INPUT will cause rewind with interlock at the beginning of the job in which it occurs. (If a tape has been "saved" and subsequently "assigned", file k will only be rewound, and there will be no mounting or dismounting instructions.) n specifies the expected number of reels for file k, thereby permitting an early release of the alternate, if any. An incorrect n will not cause an error.

c. INEX

alias k INEX n

INEX describes file k as being an unprotected input file, and causes a beginning-of-job mounting message only. File k may not have been rewound with interlock at the end of the previous job. If not, INEX will cause a rewind with interlock at the beginning of the job in which it occurs. (If a tape has been "saved" and subsequently "assigned", file k will only be rewound, and there will be no mounting or dismounting instructions.) n specifies the expected number of reels for file k, thereby permitting early release of the alternate, if any. An incorrect n will not cause an error.

d. OUTPUT

alias k OUTPUT

OUTPUT describes file k as being a protected output file. File k may not have been rewound with interlock at the end of the previous job. If not, OUTPUT will cause rewind without interlock

at the beginning of the job in which it occurs. In either case, a MOUNT BLANK message is produced. An end-of-job dismounting message will be typed out. If a tape has been "saved", a dismounting message will not occur in the job in which the tape was saved, and only rewind will occur in the job which assigned the tape.

e. SCRACH

alias k SCRACH

This describes file k as being a scratch tape. File k may not have been rewound with interlock at the end of the previous job. If not, SCRACH will cause rewind without interlock at the beginning of the job in which it occurs. If the previous job did rewind with interlock, a MOUNT BLANK message is produced at beginning-of-job. If a tape has been "saved" and subsequently "assigned", the reel will be rewound only and there will be no mounting messages.

f. ALT

alias k ALT k1, k2,...

This describes file k as being an alternate to files k1, k2,... If k1 is an input file, then there should only be the entry k1 in the list. If k1 is an input reel, then a MOUNT message will be produced and the unit rewound with interlock if it is not dismounted. If k1 is an output reel, then a MOUNT BLANK message will be produced if the unit is dismounted.

g. SAVE

alias k SAVE

SAVE specifies that file k be carried over to the next run. If file k 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 a MOUNT message and rewind with interlock, if appropriate.

A tape which has been "saved" must be assigned (with an ASSIGN card) in the succeeding job.

h. DUMP

k DUMP

DUMP specifies that file entry (k) is the system dump tape. A dump tape must be specified for each main program which will be run in conjunction with symbionts, or which will use rerun. It may be an output tape which employs standard tape conventions, i.e., has been written under control of the Tape File Label Handling routine.

D. OPERATIONAL CONTROL

1. Nominal Tape Assignment

The following tape assignments are used by ACCO. If ACCO is called by an RXΔACCO type-in. these assignments are fixed and may not be changed. Also, the control input will be from cards, not tape.

When ACCO is under SUCO control, the tape assignments may be changed once at the beginning of the run by the use of appropriate tape assignment parameter cards following a SERVO control card. The dump tape must be specified as an output tape which would be read by a system which recognizes bypass sentinels. The List/Punch tape satisfies this requirement, but if it is not being used, a separate output tape must be specified.

a. Tape Assignment for UTMOST

File No.	Usage
0	System tape
1	PRESTO control input
2	List/Punch tape
3	PRESTO relocatable output
4	Scratch
5 - 15	Library tapes as needed

b. Tape Assignment for COBOL

File No.	Usage
0	System tape
1	PRESTO control input
2	Scratch - List/Punch tape
3	Scratch - PRESTO relocatable output
4	Scratch
5	Scratch
6	Scratch
7	COBOL library
8 - 15	Library tapes as needed

c. Tape Assignment for FORTRAN

File No.	Usage
0	System tape
1	PRESTO control input
2	List/Punch tape
3	PRESTO relocatable output
4	Scratch
5	Scratch
6	Scratch
7 - 15	Library tapes as needed

2. Console Functions

a. SUCO Control (for automatic tape operation)

1. CLEAR
2. REWIND
3. LOAD
4. PROGRAM STOP - sets all memory to SLJ ERR
5. PROGRAM RUN
6. KEYBOARD REQUEST
7. Type in RSΔCALLΔACCOΔ2
8. KEYBOARD RELEASE (Activates ACCO)

b. EXEC Control (for card operation)

1. CLEAR
2. REWIND
3. LOAD
4. RELEASE (This causes EXEC control)
5. PROGRAM STOP - sets all memory to SLJ ERR
6. PROGRAM RUN
7. KEYBOARD REQUEST
8. Type in RXΔACCO
9. KEYBOARD RELEASE (Activates ACCO)

DECO

A. PURPOSE

The purpose of DECO (DEsignation COntrol), one of the three General Program Processors associated with the BOSS III SUPPORT system, is to convert the relocatable programs produced by COBOL, FORTRAN, and UTMOST into an operational system tape - that is, a loadable instruction tape, ready for execution. The information on this system tape will consist of either absolute programs or dynamically relocatable programs.

B. INTRODUCTION

1. Input

DECO will accept three types of input: control input on tape or cards, relocatable object code library tape(s), and system tapes. (All tape input to DECO, except system tapes, is in the condensed PRESTO format, which simply means that all consecutive blanks and zeroes have been removed. None of the original information content is lost.)

a. Control Input

Control information defines and directs the preparation of a system tape. The control information is always in a symbolic format, and is contained either on tape or on cards.

Tape control input is used when automatic operation is employed (i.e., job-to-job chaining under control of SUCO). Card input is used when DECO has been called by means of an operator type-in. Thus control input is optionally tape or card, but not both.

Control input may contain several types of control cards, and may contain binary elements. Binary elements are the relocatable object code output of COBOL, FORTRAN, or UTMOST. Control input comes from a previous processor (ACCO or UPCO), from the PRESTO CTT Symbiont, or directly from

cards. Some of the control information for a job may also come from a relocatable object code library tape. See the section on Control Cards for an explanation, in detail, of the various types of control cards.

b. Relocatable Library Tapes

Library tapes may contain numerous sets of independently compiled relocatable object code arranged in elements and groups, where a group is a collection of elements or other groups. Control information may appear within elements or groups, and will be processed in the order in which it is encountered. The user may thus keep control information which is permanently associated with a job on a library tape with that job, thereby reducing the amount of primary control input needed to process the job through DECO.

The use of library tape input is optional with the user; control information from tape or cards is the only required input. The number of library tapes used as DECO input is limited only by the number of available tape units. Only one library tape is available to DECO at a given time, the selection being under control of the user by means of a TAPE control card.

c. System Tape

The system tape on logical tape unit 0 may be used as input to a DECO run. Jobs on this system tape may be copied onto the system tape being created by DECO under control of KEEP control cards.

2. Output

DECO produces as output a system tape (loadable instruction tape) and an optional List tape.

a. System Tape

The system tape will normally contain a bootstrap loader (BOOT), an in-core executive routine (EXEC), a supervisory control routine (SUCO), diagnostic aids, main programs, and symbionts. (For detail, refer to the System Tape section.) From this system tape, main programs and symbionts will be loaded into memory for execution.

(1) Main Programs

Each of the main programs (jobs) processed by DECO has been converted from relocatable object code to absolute object code. Each job appears on a system tape as a series of blocks which contain absolute object code, preceded by a block with the job's related control information.

Main programs, at execution time, will be located in core immediately above the executive system, or as specified in a SEG card. Main programs are not dynamically relocatable.

(2) Symbionts

The system tape may contain symbionts - programs which normally control the operation of peripheral equipment. A symbiont may operate concurrently with a main program, other symbionts, or it may operate alone. Symbionts are under complete control of operator type-ins.

Symbionts may be absolute or relocatable, depending on the definition of the user at DECO time. When a dynamically relocatable symbiont is called into memory, it is loaded into the highest available memory location. Subsequently, if higher memory locations become available (by the completion of some other symbiont), all dynamically relocatable symbionts in core will be moved

up to overlay the vacated space. For more detailed information on symbionts, refer to the BOSS III ~~Reference Manual.~~

Absolute symbionts, when called, will be loaded into whatever portion of upper memory is specified by the programmer with a RES card at assembly time, or a SEG card at DECO time. Absolute symbionts are not relocated during execution, and no relocatable programs will be loaded above them. Therefore, it is advisable that absolute symbionts be written to occupy the highest addresses practical.

b. Mapping (List Tape)

DECO provides the user with a listing which may show the control cards used by DECO for each job, the name and location of each element, the name and location of externally defined labels, and diagnostic error messages. The user may elect to show error messages only, by using a particular MODE card option.

The list tape is prepared if file #2 has been specified on a tape assignment card following a SERVO card at the beginning of the DECO run. The information which DECO places on a list tape is in the format required by the PRINT/PUNCH tape symbiont. The tape would be printed later, concurrently with some other job, with the PRINT/PUNCH symbiont. The listing will be done on line if file #2 (the list tape) has not been specified. (There is no punch output from DECO.)

C. GENERAL

1. Groups and Elements

A program (job) contains a combination of groups and elements - or a program may consist of only one group or element. An element is the smallest program unit; a group is a collection of elements or other groups. Elements contain either control cards or binary card images, or both. Groups may contain control

cards, but may not contain binary card images unless they are within an element.

Groups and elements are identified by names, which may not exceed eight (8) characters. A group name must be unique within a group, and an element name must also be unique within a group.

Control cards for DECO follow the same free-form format as a line of UTMOST symbolic coding, except that they must contain a 12-0-2 punch in column 1.

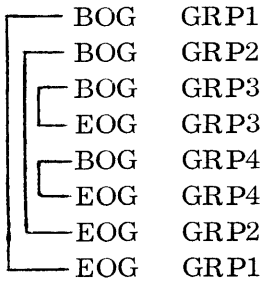
Groups and elements are identified by the following control cards:

BOG	GroupName	(Beginning of group and its name)
EOG	GroupName	(End of group and its name)
ELT	ElementName	(Beginning of element and its name)

On a library tape an element is terminated by the appearance of another ELT, a BOG, or an EOG card.

The BOG and EOG control cards serve as brackets around a section of coding. These cards are descriptive labels for a group, and should be the first and last cards of any group. All other control cards pertinent to a group must appear within the BOG and EOG control brackets. Groups may be nested within groups.

An example of nested groups might be:



Group 1 (GRP1) contains groups 2, 3, and 4.
Group 2 (GRP2) contains groups 3 and 4.

An ELT control card precedes the card images belonging to the element named in its operand field (ElementName). To be part of a group, elements must be contained within BOG and EOG control cards. Within a group, an ELT card separates succeeding elements. Elements which are not part of a group are terminated by the next BOG or ELT encountered.

The following is an example of elements within nested groups:

```
BOG   GRP1    Group 1 (GRP1) contains element 1 (ELT1)
ELT   ELT1              and groups 2, 3, and 4.
BOG   GRP2    Group 2 (GRP2) contains groups 3 and 4.
                        GRP2 does not contain binary
                        elements in itself.
  BOG GRP3    Group 3 (GRP3) contains elements 2 and 3.
  ELT ELT2
  ELT ELT3
  EOG GRP3
  BOG GRP4    Group 4 (GRP4) contains element 4.
  ELT ELT4
  EOG GRP4
EOG   GRP2
EOG   GRP1
```

To call an element within a group, subscripting is used. For example, to call ELT3 in the preceding example:

```
SELECT GRP1(GRP2(GRP3(ELT3)))
```

or, to call ELT1:

```
SELECT   GRP1(ELT1)
```

or, to call all of GRP3:

```
SELECT   GRP1(GRP2(GRP3))
```

2. Binary Card Images

Binary card images appear within elements (but not within groups). DECO recognizes five types of binary card images which are created by UTMOST, COBOL, and FORTRAN:

Relocation card
External Symbol Reference
External Symbol Definition
End Card
Instruction Card

The order of the binary cards within an element is immaterial to DECO.

a. Relocation Card

A relocation card consists of one or more relocation references. Each reference specifies a location which contains a 15-bit address. DECO adds the current base address to this 15-bit address (ignoring sign and masking out any overflow), thereby effecting relocation from the originally defined base to the current base. The original base was assigned at assembly or compilation time. In COBOL and FORTRAN, base zero is always used. In UTMOST, base zero is used if a base is not defined; a base other than zero can be defined with a RES card at assembly time.

For most programs, the relocation base will be initialized for each job at DECO time to a location immediately above the executive routine. This address may be changed, during DECO, by the use of a SEG card. The relocation base is incremented within a job by the lengths of succeeding elements. This length is obtained from the END card for each element.

For symbionts, the type of relocation employed is defined by use of a SYM card.

b. External Symbol Reference

An entry is created in an external reference card image when a symbol (label) is left undefined at assembly or compilation time. This entry consists of from two to five words. (The entry is variable in order to permit different lengths in labels.) The first words in the entry contain the undefined symbol. The last word in the entry specifies the location of the word which contains the undefined field, and the low-order bit position of this 15-bit field. The word which contains the undefined field may be an indirect address word which has been created because of the undefined symbol, or may be an instruction which contained the undefined symbol itself. The undefined field may contain a constant increment or decrement as will be shown in examples following the discussion on external symbol definition. The entire external reference entry is stored in a table called EXTREF during the first pass of DECO.

Definition and relocation of the undefined field await a corresponding external definition card.

c. External Symbol Definition

An entry is created in an external definition card image when a symbol (label) has been marked as an externally defined symbol. (In the UTMOST language, external definition is indicated by a label followed by an asterisk.)

Each entry consists of from two to five words. The first words in the entry contain the defined symbol. The last word in the entry specifies the 15-bit value corresponding to the defined symbol. This may be an absolute value or an address relative to the originally defined base. The last word may specify relocation.

The entire external symbol definition entry is stored in the table EXTDEF during the first pass of DECO. If relocation is specified in the last word of the entry, it is effected by

adding the value for the defined symbol to the current job base address before storing in EXTDEF. If duplicate symbol definitions occur, the first definition encountered holds and an error message is written.

As each external symbol reference or definition is found by DECO, that entry is placed in the appropriate table (EXTREF or EXTDEF). After DECO has passed the entire job (all groups and elements of a program), the EXTREF table is matched against the EXTDEF table. As a match is made, that external reference becomes a defined reference. A library search will then be made for those external references which are undefined, if library search has been specified by a LIBE card. This search is performed by using the undefined external symbol reference label as a key. The elements within the specified group are scanned for a matching externally defined symbol. When a match is made, the entire element which contains the matched label is brought in as part of the library routines for the job. Error messages will be written for those references still undefined after the library search, if there was one. A second pass on the job is then made, and the values for the defined references are inserted into the program.

The following example will illustrate how DECO handles an external symbol reference and definition. Assume that the following job is being processed through DECO. (For ease of representation, symbolic coding is used in the example.)

```
(1)  Z      JOB    EXAMPLE     (where Z is a 12-0-2 punch)
(2)  Z      ELT    PART1
(3)         SA     3,SAVE+1
                   .
                   .
(4)         END
(5)  Z      ELT    PART2
                   .
                   .
                   .
```

```
(6)   SAVE*   RES   2
                .
                .
(7)           END
(8)   Z       FINIS
```

The coding in element PART1 will create an entry (SAVE) at assembly time in an external reference card, and will create an indirect address word containing the increment 1 (+00000001). The coding in element PART2 will create an entry (SAVE) in an external definition card.

Assume the following: that the address within PART2 for SAVE is 0100, that the base address assigned to the job is 07000, and that the length of PART1 is 0300.

During DECO's first pass the external reference entry for SAVE is placed in the EXTREF table, and the external definition entry for SAVE is placed in the EXTDEF table. During the second pass, the value 07000+0300+0100 (or 07400) will be added to a 15-bit field in the indirect address word as indicated in the external reference entry. In this case, the final value of 07400 + 01 is given. At execution time, line 3 of the example will make an indirect reference to the generated word which will now contain 07401.

d. End Card

Effectively there are two kinds of end cards. One contains only the length of the element in which it appears. The other contains the transfer address (where control is to go after a job has been loaded), the values to be loaded into the cover index registers, as well as the length of the element or subprogram in which the end card appears.

e. Instruction Card

An instruction card contains one or more contiguous binary data or instruction words, together with the location of the first word relative to the original base.

3. Program Structure

Programs and subprograms may be combined in several different ways to permit maximum efficiency. The structure of a program will fall into one of the following classes:

a. Single-Compilation Programs

For single-compilation programs (UTMOST, COBOL, FORTRAN) which fit into core, DECO simply serves as a relocator. DECO will write the program on the absolute system tape for execution loading by BOOT.

b. Separately Compiled Main Programs and Independent Subroutines

This program will consist of a main program and one or more independently compiled subprograms (or subroutines). The entire program (job) will fit into one core load. Main programs and subprograms may be written in UTMOST, COBOL, or FORTRAN.

There are many advantages to this programming technique: assembly or compilation time is minimized, checked out library subroutines may be inserted automatically, and because of the modular structure, debugging is considerably simplified.

The following discussion will indicate how communication between separately compiled or assembled main programs and subroutines is controlled by the programmer.

(1) External Labels

In UTMOST, if a label in the label field is immediately followed by an asterisk, and the line is not within a procedure (PROC), it is an external label, and can be referenced by other separately assembled subprograms.

(2) External Symbol Reference

In UTMOST, if a label in the operand field is left undefined, it is an external symbol reference, and may be defined by a corresponding external symbol definition at DECO time. Such definitions must be in the form of 15-bit addresses.

(3) COMMON

Within the FORTRAN language, two kinds of common data areas may be defined. They are "labeled COMMON" and "blank COMMON". A blank COMMON area is used and defined within an entire job, and its contents may be referenced by any link of a chain, any element, or any segment. A labeled COMMON area may be referenced and defined within any separately compiled subroutine. The first subroutine which references the labeled COMMON area will define for DECO the position of that area. Space will be allocated in the subroutine according to this first definition for the labeled COMMON area. Subsequent references to this particular labeled COMMON area within a link of a chain job, or in any segment of a segmented job, will be treated as references to the first definition of the labeled COMMON area. It is therefore the user's responsibility to see that when a reference is made to a labeled COMMON area, the subroutine which contains the first reference to it will also be in memory.

FORTRAN will create special external symbol definition images which define a symbol as being the name of a COMMON area. The length of the COMMON area is carried in value position of its external symbol definition image. This length is used to define the size of a pseudo-element which becomes the common area.

This created element will be placed following the first element containing an external symbol definition for the labeled COMMON. Multiple definitions of a particular

labeled COMMON may be given; however, the first one encountered will be used to define the length of the pseudo-element (created COMMON area). The other definitions for this labeled COMMON area will not cause error or error messages, but will be ignored. It is the user's responsibility to see that the first definition found will be a proper one.

For all blank COMMON, a special symbol, QCOMMON, is generated by FORTRAN. This special name is recognized by DECO and the procedure outlined above is modified in these respects: the pseudo-element created is placed above the longest link in a chain job or above the library in a segmented job. The length of the pseudo-element is taken as the maximum length specified in the many external symbol definitions given.

It should be noted that a bit is contained in position 17 of the value word of these special external symbol references generated by FORTRAN. Standard external symbol definition images as generated by UTMOST do not contain this bit.

c. Chain Jobs

A chain job is composed of one or more programs or links. Each link may be composed of one or more independently compiled subroutines. Only one link will be in core at a given time. Normally, chain jobs will be used for FORTRAN compiled programs.

Chaining permits these independent links to communicate with each other through "blank COMMON", a common data area. The subprograms within a link may communicate with each other through "labeled COMMON". (Refer to COMMON, Section C. 3)

A link in a chain is defined by the insertion of a CHAIN control card image in front of the binary elements composing the link.

The control card is of the format,

```
CHAIN    LinkName
```

where LinkName can be alphabetic or numeric to conform with current FORTRAN usage.

Within a link, any other link within a chain job may be called by,

```
LA       3, LinkName
 J       LODX
```

where LinkName is the address of the first word of a two-word constant containing the alphabetic name of the link being called.

During the first DECO pass, each link is processed as for a job consisting of a separately compiled main program and subroutines. After each link is processed, the library search is made, excluding the special symbol QCOMMON (blank COMMON area), and the length of the link is remembered. The EXTDEF table is then reset, and the next link is processed. After all links have been processed through the first pass of DECO, the externally defined variable QCOMMON will be assigned the core location above the highest location used by any link. An area equal to the maximum length in the many QCOMMON definitions will be reserved, beginning at this location.

The following example will illustrate the make-up of a chained job:

```
JOB      JOBX
CHAIN    LINK1
ELT      ELTA
.
.
.
```

```
            LA      3, LINKB
            J       LODX
LINKB       'LINK2'
            .
            .
            .
            CHAIN   LINK2
            ELT     ELTB
            .
            .
            .
            LA      3, LINKA
            J       LODX
LINKA       'LINK1'
            .
            .
            .
            FINIS
```

When JOBX is called, LINK1 will be loaded. At some time during its execution, LINK2 may be called and loaded, overlaying LINK1. Similarly, during execution of LINK2, LINK1 may be called, and would then overlay LINK2.

d. Segmented Jobs

Many computer applications require that various sections of a program be brought into core only when they are needed. The segmentation provisions of DECO provide this facility. This is the most powerful way to operate the computer, and also places the greatest burden on the user in planning his overlays and cross referencing.

A segment is defined as a collection of binary elements which comprise a single core load. In this case, "core load" means a program or subprogram which may occupy part or all of the memory space allocated to a job. There may be one or more segments in core at any given time.

A segment's starting point is defined with a SEG control card of the following format:

```
SEG        SegName, Origin
```

Origin may be symbolic (any previously defined externally defined label), blank, or an actual location. When Origin is blank, the segment will be loaded into the normal location immediately above the executive routine.

When a job containing segments is called, the first segment will be automatically loaded into core. The loading process is halted when the symbol block of the second segment is encountered. From this first segment, any other segment(s) may be called into core. Thereafter, any segment in core may call any other segment, in one of two ways:

```
LA         3, SegName
J          LODX
```

where SegName is the address of a two-word constant containing the alphabetic name of the segment being called.

This will load the segment (SegName), and transfer control to a location specified in a "main" program in the segment.

```
LX         1, (ROSIE)
LA         3, SegName
J          LOAD
```

The segment will be loaded, and control will go to ROSIE. ROSIE can be any location within any segment now in core, including the called segment (SegName).

In either of the above cases, the calling segment can be completely overlayed by the called segment.

As each segment is processed by DECO, the external symbol definitions and external symbol references are placed in the appropriate tables. After the entire job has been processed through DECO's first pass, if a library search has been specified by a LIBE card, an automatic library search is made for any undefined external symbol references. This search is made within the group indicated on the LIBE card, on the file specified by the last TAPE card image, using the label in the undefined external symbol reference as the key. Elements called in as a result of this search are relocated above the highest memory used by any segment. At the end of the first DECO pass, this library information is written onto an intermediate tape. During DECO's second pass, the library information on the intermediate tape is written onto the system tape as part of the first segment. It should be noted that when the first segment is loaded, the library portion will be located in higher memory, apart from the segment itself, which is ordinarily loaded into an area just above the executive routine.

If the user requires a library routine only within a particular segment, he may call it with a SELECT card, and this library routine will be loaded with this particular segment. It will not be included with the high-core common library area. If more than one segment requires a library routine, the library routine should NOT be called by a SELECT, but should be included in the high-core library routines.

The FORTRAN externally defined "blank COMMON" symbol QCOMMON will be located above the library routines.

The processing of inter-subroutine or inter-segment communications operates on the entire job. It is the user's responsibility to see that when an external symbol reference to another segment is made, that segment is in core.

An example of a segmented job follows. For ease of representation, symbolic coding is used.

```
(1)             JOB       EXAMPLE
(2)             TAPE      6
(3)             LIBE      PAYROLL
(4)             SEG       SEG1
(5)  START      LA        3,SEG2
(6)             J         LODX
(7)  A*         RES       500
(8)  B*         RES       500
(9)  C*         RES       1498
(10) D*         RES       2000
(11)            END       START
(12)            SEG       SEG2,B
                .
                .
                .
(13)            LA        1,A+12
                .
                .
                .
(14)            LX        1,($+3)
(15)            LA        3,SEG3
(16)            J         LOAD
                .
                .
                .
(17)            LA        1,E
                .
                .
                .
(18)            LX        1,($+3)
(19)            LA        3,SEG4
(20)            J         LOAD
                .
                .
                .
(21)            SLJ       TAX
(22)            LA        1,F
                .
                .
                .
(23)            SEG       SEG3,C+2
                  binary elements
(24)            SEG       SEG4,D
                  binary elements
(25)            FINIS
```

Line 2 specifies that the logical tape unit number in the 7th entry of the tape assignment table is to be the library tape.

Line 3 specifies that the automatic library search is to be limited to the group called PAYROLL.

Line 4: Note this line is redundant, and name of this first segment will be the job name (EXAMPLE).

Lines 5 and 6 cause SEG2 to be loaded and control to be transferred to SEG2.

Lines 7 - 10: The values 500, 500, 1498, and 2000 will define the relative locations in memory of A through D.

Line 11, at DECO time, contains the length of SEG1 as it was assembled. The segment extends from START to the library routines.

Line 12 defines the segment SEG2, and will cause it to be loaded at location B.

Line 13 references a location in SEG1.

Lines 14, 15, and 16 cause SEG3 to be loaded into its defined starting location, C+2. Control is retained in SEG2, at $+3.

Line 17 references a location in SEG3 which is now in memory.

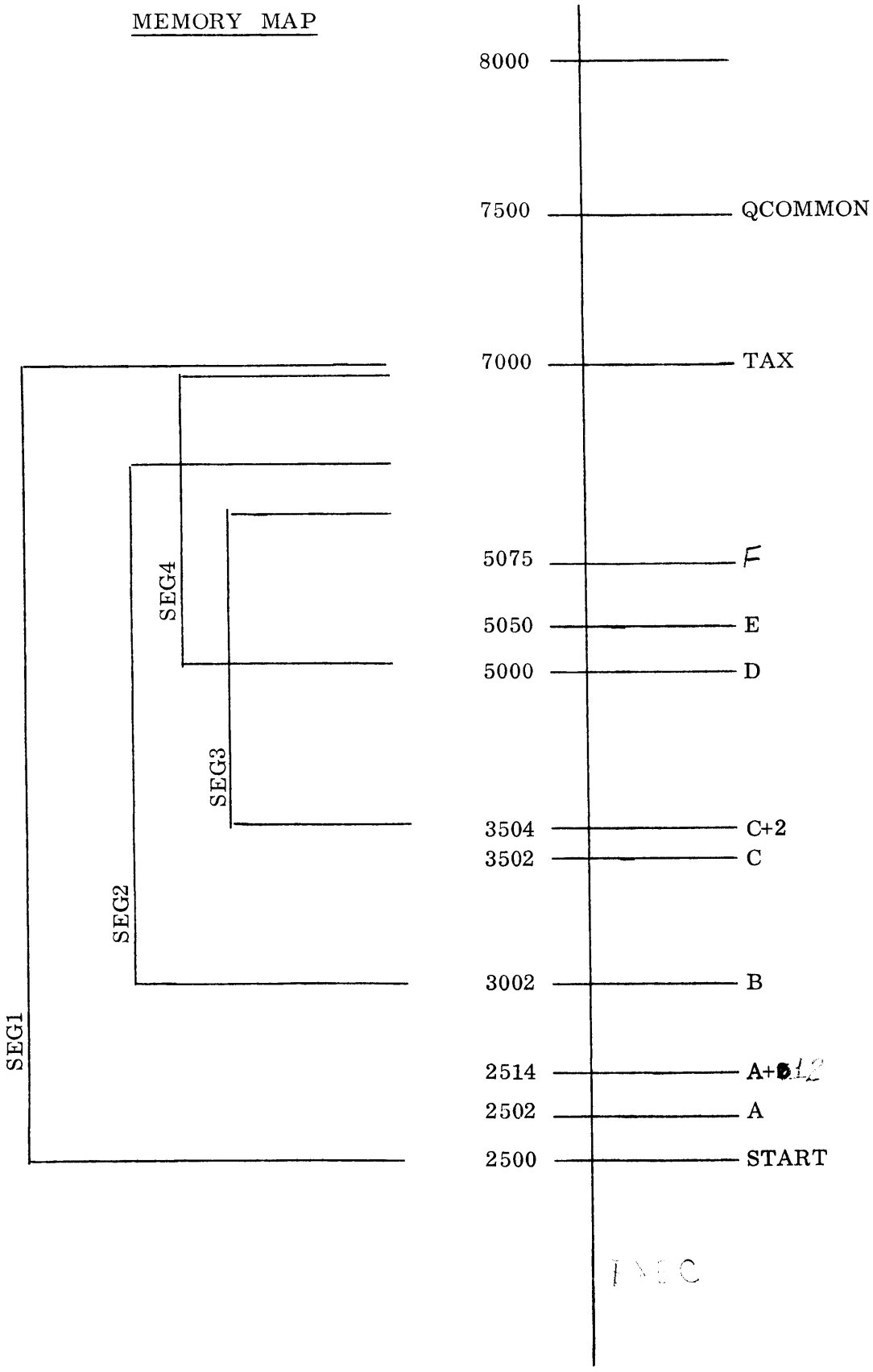
Lines 18, 19, and 20 cause SEG4 to be loaded at location D. Control is retained in SEG2 at $+3.

Line 21 SLJ TAX causes the library to be searched for the routine TAX. This routine will be loaded into the library area, above the highest location used in the job - in this case, above SEG1 or SEG4.

Line 23 defines the segment SEG3, and will cause it to be loaded at location C+2.

Line 24 defines the segment SEG4, and will cause it to be loaded at location D.

MEMORY MAP


8000
7500
QCOMMON
7000
TAX
5075
F
5050
E
5000
D
3504
C+2
3502
C
3002
B
2514
A+B
2502
A
2500
START
SEG1
SEG2
SEG3
SEG4

4. Symbionts

The primary burden for concurrency lies upon the symbionts, programs which control the operation of peripheral equipment. The BOSS III system is designed to permit one principal program in the machine at a time, and to allow concurrent operation of any number of symbionts.

Symbionts are required to release control back to the system when they find that the peripheral unit they are using is busy. They must also save and restore any registers that they use other than the basic registers. (Refer to the BOSS III ~~Reference Manual~~ for detail.) They must reset these registers upon return from any release.

a. Storage Allocation

There are two types of symbionts - absolute and dynamically relocatable. If absolute symbionts are used, automatic memory allocation features are foregone; scheduling and memory area designation become the responsibility of the operating personnel.

If the symbiont is dynamically relocatable, the system will provide automatic memory allocation at all times, and will group the symbionts in contiguous areas in highest available core. (Of these, the Card Reader and Card Punch symbionts are relocatable only in increments of 0100_8, due to hardware requirements.)

All symbionts must be accompanied by a SYM control card at DECO time. The SYM card indicates whether absolute or dynamic relocation is to be used. For the SYM card format, see the section on Control Cards.

b. Allocation of General Purpose Channels and Tape Units

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

Symbionts may be indicated as operator-defined by including a DEF control card with the symbiont at DECO time:

Ƶ DEF S_1,S_2,S_3,S_4

in which S_1,S_2, etc., are undefined symbols representing, in this case, channels or tape units. The DEF card will cause SUCO to generate a typeout of each undefined symbol, following which the operator will type in the symbol definition (tape unit or channel designation). A maximum of four symbols may be entered on a DEF card. Any number of DEF cards may be used within a job, but the total number of symbols given may not exceed fifty.

D. CONTROL CARDS

DECO recognizes two types of control cards. They are DECO control cards, and SUCO control cards introduced at DECO time.

These control cards follow the same free-form format as a line of UTMOST symbolic coding, except that they must contain a 12-0-2 punch in the first position, and must not have comments on them.

ƵLABELΔOPERATIONΔOPERAND

where Ƶ is a 12-0-2 punch, and where Δ indicates at least one blank.

If the first field of a control card contains the name of a processor, the control card is then made "transparent" to all other processors, i.e., ⧧DECOΔSELECTΔName. This control card may be passed through UPCO or ACCO with no action being taken until the card is processed by DECO.

In the following discussion of control cards, formats and examples will not show labels unless they might also apply to another processor.

1. DECO Control Cards

a. JOB

JOB JobName, Installation Information, Carriage Return

The JOB card image defines the name of the program (JobName), and will carry such information as the programmer or installation requires (accounting information, programmer name, estimated time, etc.). DECO will place this entire image in the symbol block for the job on the absolute system tape. JOB cards are searched for by SUCO for object time execution. SUCO will output the entire JOB card image on the console typewriter. A carriage return symbol should be placed immediately following the installation information, to keep the typeout as short as possible.

A JOB card is placed in front of all control cards and binary elements for a particular job.

b. SEG

SEG SegName, Origin

SEG defines the name by which a segment is called (SegName), and the origin of the segment (i.e., location into which the segment will be loaded). Origin may be a symbolic name - any previously defined externally defined label, Origin may be an actual memory location, or Origin may be blank, in which case the segment is loaded in the area immediately above the executive routines.

A SEG card is placed in front of the segment which it defines.

c. CHAIN

CHAIN LinkName

CHAIN defines the name of a link. The chain card precedes the binary elements composing the link. LinkName may be alphabetic or numeric to conform with current FORTRAN usage.

d. MODE

DECO MODE Option

where Option will be one of the following:

LNOR	mapping, and error messages
LCOR	mapping, and error messages
LDTL	mapping, and error messages
LCTL	mapping, and error messages
LDNT	error messages only

Mapping will include control cards, starting locations for each element, and assigned address for each external symbol definition. If no MODE card is present, the listing will be as for the LNOR option.

The listing is done on-line unless the List tape has been specified on a tape parameter card following a SERVO card at the beginning of the DECO run.

The listing mode is in effect until it is redefined by a MODE card. Therefore, a MODE card may be placed anywhere within a DECO run.

e. ELT

ELT ElementName

An ELT card image defines each element to be processed by DECO. It must precede all information which is part of element ElementName. Each ELT card image is output on the memory map. An element is terminated by another ELT, JOB, SEG, CHAIN, DATA, or FINIS card.

f. DATA

DATA Origin $\pm$ Increment

DATA directs DECO to locate the data images which follow into a core position relative to Origin, which is a previously defined externally defined label. (Origin may also be an absolute location.) It is frequently useful to load data with a program at DECO time. The data images thus become a part of the program at object time, but need not be assembled as constants within an element.

Data card images following the DATA control card image will be terminated by another DATA, ELT, SEG, CHAIN, or FINIS card image.

Each data card image follows UTMOST formatting for constants, and can be decimal, alphanumeric, octal, or binary:

+ :12	(decimal)
+ 0120	(octal)
+ 'ABCD'	(alphanumeric)
- 12	(binary)

Binary instruction card images will also be accepted when they follow a DATA control card. This is particularly useful for making patches to object code programs. They are loaded into the specified location (Origin ± Increment). These patches may not, however, contain values which need relocation.

Binary data card images are loaded into an area relative to Origin rather than being controlled by the relocation counter. This will be used for the FORTRAN DATA statement, and for subroutines requiring a fixed origin, such as in communication with EXEC.

g. KEEP

KEEP cards must appear outside of a job.

KEEP JobName

KEEP JobName will cause DECO to copy the specified job (JobName) from the controlling system tape (on logical unit 0) onto the DECO output tape.

KEEP BOOT

DECO will rewind its output tape, and then copy BOOT (bootstrap) onto it.

KEEP ALL

The entire system tape will be copied onto the DECO output tape, after the output tape is rewound.

KEEP JobA, JobB

DECO will search for JobA and copy the input tape from the beginning of JobA through JobB.

KEEP INPUT, n

The KEEP INPUT card will cause KEEP cards subsequent to it to copy from file #n. (If no KEEP INPUT card is given, input will be on file #0.) The KEEP INPUT card itself does not cause copying. As many KEEP INPUT cards may be used as necessary, thus facilitating the creation of a new system tape from several other system tapes.

KEEP OUTPUT, n

The KEEP OUTPUT card causes KEEP cards subsequent to it to copy tape onto file #n. (If no KEEP OUTPUT is given, output will be on file #3.) As many KEEP OUTPUT cards may be used as necessary.

h. MOD

MOD Value

This causes the relocation counter to be raised to the next multiple of Value. For program checkout, it is often convenient to cause a program to begin at the next multiple of some power of 2. This is accomplished by the MOD card, which causes an adjustment to the relocation counter. For example:

MOD 01000

causes the relocation counter to be set to the next multiple of 01000 (512_{10}).

Due to hardware requirements, it is necessary that the data areas for card read and card punch subroutines begin at an address which is a multiple of 0100 (64_{10}). To accomplish this, a MOD 0100 card should be included with the card reader dispatcher and with any subroutines which use the punch dispatcher.

A MOD card should be placed just before the element with which it is associated.

i. SERVO

DECO SERVO

When running under SUCO control, all tape assignments for each DECO run must be specified by use of a SERVO control card followed by tape assignment parameter cards. The SERVO control card must be the first control card in a DECO run, and there may be only one SERVO card per job. Any SERVO cards encountered after the first will ~~be ignored~~.

j. SELECT

DECO SELECT LibraryName

A library tape which has been specified by a preceding TAPE card will be searched for LibraryName. The information within LibraryName (a group or element) will replace the SELECT card in the program sequence.

Previously compiled or assembled subroutines may be kept on relocatable object code library tapes. A particular subroutine may be called into a job or segment by use of a SELECT card. Subscripting may be used.

A SELECT may be used at any place within a job where the user wants to insert an element from a library tape.

k. TAPE

DECO TAPE FileNumber, Label

The TAPE control card specifies which file number is to be used as the library tape. FileNumber is a number (0 through 15) which specifies an entry in the tape assignment table. For example, TAPEΔ6 specifies the seventh entry in the tape assignment table. If Label is present, the label of the specified library tape will be matched against this label, and an error message typed out if they do not match. If Label is not entered, the label block on the specified library tape will be ignored.

Each file designated in a TAPE card must have been previously defined by a tape assignment parameter card. A TAPE card may be used whenever it is desired to specify a new library tape. Always the card is used, [illegible] with a record.

l. FINIS

DECO FINIS Name

A DECO run must be terminated by a FINIS card. The Name entry specifies the processor or run to be entered next. When Name is blank, SUCO will type out a message ("Next") upon reaching EOJ, and will spin in a STOP loop until the operator calls a new program.

m. LIBE

LIBE GroupName

After DECO has called in all the elements of a job, the external symbol definitions and external symbol references in those elements are compared. Any external symbol references which remain undefined are searched for on the currently specified library tape (see TAPE control card) only if a LIBE

control card is present. The use of the LIBE card will limit the search to a particular group (GroupName) on that library tape.

If a LIBE card image is not present an automatic library search will not be performed.

It should be noted that the LIBE card is effective only for the undefined symbol search and has no effect on other tape positioning orders such as SELECT and FIND.

Only one LIBE card image may be used within each job. ~~It should appear at the beginning of the job.~~

2. SUCO Control Cards

In addition to control cards for DECO itself, DECO accepts certain control cards for use by SUCO. These cards are passed on in the job preamble (control block) of the system tape.

These control cards may appear anywhere within a job.

a. NEXT

NEXT Name

Name specifies the program to be called at end of job of the current program. The symbol Name will be placed in Words 4 and 5 of the preamble on the system tape. If NAME is blank, SUCO will go to the ~~typewriter~~ for the ~~next program~~. If no NEXT card is present, SUCO will type out a message upon reaching end of job in the current program, and will spin in a STOP loop until the operator calls a new program.

b. HOLD

HOLD First, Last

This card specifies the first and last locations of the core area which is not to be cleared before loading the job in which the card appears. If no HOLD card is present, the entire core area within the job boundaries will be cleared when a JOB card is encountered. Only one HOLD card may be included in any one job. Ordinarily, First and Last would be absolute memory locations. They could be previously defined symbols within the job.

c. SYM

SYM X

Symbionts may be 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 is used to indicate which type is desired. In the above format,

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.

d. DEF

DEF S1, S2, S3, S4

in which S1, S2, etc., are otherwise undefined symbols.

A maximum of four symbols may be entered on a DEF card. Any number of DEF cards may be used within a job, but the total number of symbols given may not exceed fifty.

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. The DEF control card indicates these symbols as being operator-defined. DEF 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 an installation in order to minimize operator confusion.

The DEF option is available only to symbionts. Main program symbols may not be operator-defined.

3. Tape Assignment

Tape assignment parameter cards are placed with the beginning parameter information for a run. They are condensed by DECO and are written on the system tape as part of the JOB preamble. During the initialization of a run by SUCO, they are examined and appropriate action is taken.

When running under SUCO control, all tape assignments for each DECO run must be specified by use of a SERVO control card followed by tape assignment parameter cards. The SERVO control card must be the first control card in a DECO run, and there may be only one SERVO card per job.

The format of the tape assignment parameter card does not comply with rules for other control cards. The format is not variable:

Columns	Entry
1 - 6	alias
7	blank or comma
8 - 9	File number, right justified
10	blank or comma
11 - 16	Operation
17	blank
18...	One to three-digit assignment numbers right-justified in columns 20, 24, 28..., and separated by commas or blanks.

The file alias has no logical attachment to any symbols generated by a program, and is carried as a mnemonic device only. Its sole use is on tape assignment parameter cards and on correspondingly generated tape mounting, posting, and dismounting instructions via the console typewriter.

The File number (columns 8 and 9) is a number (0 through 15) which specifies an entry in the tape assignment table. For example, file number 10 specifies the eleventh entry in the tape assignment table.

The function of the assignment numbers (columns 18...) depends upon the particular operation involved. All assignment numbers are decimal.

Tape assignment cards should appear at the beginning of the job to which they apply.

For a description of operator messages which might appear on the console typewriter as a result of the following parameter cards, refer to the BOSS III Reference Manual.

a. ASSIGN

alias ASSIGN k1, k2

The file entry k1 from the previous run will be assigned to file entry k2 of the current run. This is accomplished by interchanging the logical unit numbers between the two file entries in the tape assignment table. 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.

The rewind-with-interlock provisions of INEX, INPUT, OUTPUT, and SCRACH will not apply to a tape which has been saved with a SAVE card, and subsequently assigned with an ASSIGN. Neither will there be mounting or dismounting instructions.

b. INPUT

alias k INPUT n

INPUT describes file k as being a protected input file, and causes beginning-of-job mounting instructions and end-of-job dismounting instructions. If reels are not in a dismounted status at the beginning of the run it produces rewind-with-interlock instructions. (If a tape has been "saved" and subsequently "assigned", the reel will only be rewound, and there will be no mounting or dismounting instructions.) It sets the "input" bit in the tape assignment table entry k. n specifies the expected number of reels, thereby permitting an early release of the alternate, if any. An incorrect n will not cause an error.

c. INEX

alias k INEX n

INEX describes file k as being an unprotected input file, and causes beginning-of-job mounting instructions only. If reels are not in a dismounted status at the beginning of the run, it produces appropriate rewind-with-interlock instructions. (If a tape has been "saved" and subsequently "assigned",

the reel will only be rewound, and there will be no mounting or dismounting instructions.) It sets the "input" bit in the tape assignment table entry k. n specifies the expected number of reels, thereby permitting early release of the alternate, if any. An incorrect n will not cause an error.

d. OUTPUT

alias k OUTPUT

OUTPUT describes file k as being a protected output file. It causes assurance of scratch tapes at beginning of job, by rewinding with interlock if the reel is not already in a dismounted status. In either case, a MOUNT BLANK message is produced. Appropriate end-of-job dismounting instructions will be typed out. If a tape has been "saved", dismounting instructions will not occur in the job in which the tape was saved, and rewind will occur in the job which does the assigning.

e. SCRACH

~~alias~~ k SCRACH

SCRACH describes file k as being a scratch reel. It causes assurance of a scratch at beginning of job by rewinding with interlock if the reel is not already in a dismounted status. MOUNT SCRACH message is produced at beginning of job if the file is dismounted. If a tape has been "saved" and subsequently "assigned", the reel will only be rewound, and there will be no mounting instructions.

f. ALT

alias k ALT k1, k2, ...

ALT describes file k as being an alternate to files k1, k2..., and sets the alternate reference bit in entry k1, k2,... of the tape assignment table. If k1 is an input file, then there should only be the entry k1 in the list. If k1 is an input reel, then a MOUNT message will be produced and the unit rewound with interlock if it is not dismounted. If k1 is an output reel, then a MOUNT ~~SCRACH~~ BLANK message will be produced if the unit is dismounted.

g. SAVE

alias k SAVE

SAVE sets the "save" bit in file entry k of the tape assignment table, and thereby causes the file to be carried over to the next run. If file k 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.

A tape which has been "saved" must be assigned (with an ASSIGN card) in the succeeding job.

h. DUMP

~~alias~~ k DUMP

DUMP specifies the file entry (k) for the system dump tape. A dump tape must be specified for each job. It may be any output tape.

E. OPERATIONAL CONTROL

1. Nominal Tape Assignment

The following tape assignments are used for DECO. These assignments may not be changed if DECO is called by an RXΔDECO type-in. When DECO is under control of SUCO, these assignments may be changed once in a DECO run by the use of a SERVO card followed by appropriate tape assignment cards.

File No.	Usage
0	System Tape
1	Basic PRESTO input
2	~~Print/punch~~ tape
3	New System tape output
4	Scratch tape
5	Scratch tape
6	Library tape
.	.
.	.
.	.
n	Library tape

2. Console Functions

1. Clear
2. Rewind
3. Load
4. Release (~~Optional. Press only when~~ SUCO control is not ~~desired.~~)
5. Program Stop - sets all memory to SLJ ERR. Must use.
6. Program Run
7. Keyboard Request
8. Type in RSΔCALLΔDECOΔ2
9. Keyboard Release. (Activates DECO)

F. SYSTEM TAPE

A system tape is created by a DECO run. The first block on the system tape will be BOOT (bootstrap). BOOT will automatically call in EXEC, which will be next 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, an error message is typed out.

A section of the system tape containing a job called JOBA would appear as:

Symbol Block

1. SCAT word - 3, TCD
2. Segment of three words
   a. Transfer address of previous program
   b. JOBA, name of this job
3. SCAT word - 16, 0120
4. Cover register information of previous program

Preamble Block

1. SCAT word - signed negatively to make this block transparent to BOOT
2. Control information for SUCO, such as tape assignment parameters

Information Blocks

As many as 19 segment pairs consisting of:

1. SCAT word - number of contiguous instructions, absolute location of first instruction
2. Segment of contiguous instructions

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

Symbol Block

1. SCAT WORD - 3, TCD
2. Segment
   a. Transfer address of JOBA (or the last segment or link)
   b. Name of next job physically on tape
3. SCAT word - 16, 0120
4. Cover register information for JOBA (or the last segment or link)

ON-LINE MEMORY DUMP

A. Purpose

To provide an on-line memory dump on the printer useful in debugging programs. It can be used either through a calling sequence in a snap shot manner or can be called upon by typewriter input.

B. Method

The memory dump routine uses the editing routines and an interruptable print routine. The memory dump routine saves and stores all registers and the comparison indicators at the time of entry. Internally, the memory dump routine uses all 4 arithmetic registers and index registers 1 through 6. It uses index register 5 as a cover register. At the end of the dump routine, all registers will be restored and control will be returned to the location following the calling sequence that requested the dump or to the point at which the interrupt occurred if the dump was requested by the operator. This exit will occur even if there has been a typed-in restart during the course of the memory dump. (See Section C.) The memory dump routine is capable of producing 5 different fixed formats. These are: instruction, alphabetic, decimal, octal and instruction and octal together.

C. Operating Procedure

1. Program calling sequence

The memory dump routine accepts a calling sequence of the form:

```
SLJ     *MPO
+       'F'
+       FROM, N
+       THRU, N
```

where 'F' represents a single letter format code in the low order position of the word as designated below, FROM is the starting address of the area to be dumped, and THRU is the ending address of the area to be dumped. N can be either 0 or index register 7 through 15. If N is not zero then the current value of the index register specified will be used to establish the area being printed. If the third word in the calling sequence, the FROM location, carries a + sign then the contents of the registers at the time of entry to the routine will be printed out in addition to the specified area. If this sign is negative then the registers will not be printed out.

2. Typewriter Input

The dump routine accepts a typewriter request of the form:

MX 11111 22222

where M is the standard typewriter control symbol for a memory print out, X designates what format is to be used as described below, 11111 designates the first location in octal and 22222 designates the last location in octal of the memory area to be printed. The contents of the registers at the time of the request will also be printed.

Format Codes

The following alphabetic codes will be accepted by the dump:

I	Instruction format	8 words per line
A	Alphabetic format	16 words per line
D	Decimal	8 words per line
O	Octal	8 words per line
B	Both instruction and octal together	4 pairs per line

A memory dump may be stopped and restarted by keying in new limits on the typewriter. This will cause the dump to be interrupted immediately. The printout of the registers occurs only on the original request. Ordinarily, only a memory dump requested by typewriter input would be modified in this manner.

If called for, the contents of the registers at the time of entry to the memory dump routine are printed in the following format:

Low Equal High Indicators AR8 AR4 AR2 AR1
Control Counter Index Registers 1-7
Index Registers 8-15

D. Memory Space

The dump routine occupies cells 02000 through 02777 and is a permanent part of the executive routine.

E. Examples

1.
```
SLJ     *MPO
+       'A'
+       AREA1, 0
+       AREA1+59, 0
```

This calling sequence will cause the 60 memory locations beginning at AREA1 to be printed out in alphanumeric format, 16 locations per line. The contents of the registers at the time of the call will be printed out preceding the memory print out. No Index Register modification will occur.

2.
```
SLJ     *MPO
+       'I'
+       *START, 0
+       END, 0
```

This calling sequence will cause the memory locations beginning at START and ending with END to be printed out in instruction format, 8 locations per line. The contents of the registers will not be printed. No index register modification will occur.

3.
```
SLJ     *MPO
+       'D'
+       DATA, 9
+       DATA+7, 9
```

This calling sequence will cause the memory locations beginning at DATA and ending with DATA+7 to be printed out in decimal format, 8 locations per line. The contents of the registers at the time of the call will be printed out preceding the memory print out. The contents of Index Register 9 at the time of the call will be used to establish the starting and ending addresses.

NOTE: In utilizing the memory dump routine with the ALMOST assembly system, a standard EQU card should be placed ahead of the ALMOST symbolic deck which is to be assembled. In this manner, the label for the dump routine will be equated with the proper absolute address. The following label is then restricted from other use in the source program:

MPO

to provide library to program - do LIBE BOSSIII

ON-LINE EDITED MEMORY DUMP

A. PURPOSE

To provide an on-line memory dump on the printer in edited data or program format for use in debugging.

B. METHOD

The edited memory dump uses an interruptable printer control routine. The arithmetic registers, index registers, and the Low, Equal and Greater indicators are stored and printed on entry to the dump routine and are restored prior to exit. The routine uses the arithmetic registers and index registers 8 and 9. The routine is called as a procedure in the user's program, and is therefore, covered by the use registers assigned by the user. Care must be taken to ensure that the covering index register of the routine is neither 8 nor 9. The edited memory dump can produce either of two formats:

1. Data - which prints each word in octal, alphanumeric, and decimal representations.

2. Program - which prints each word in octal, instruction word, indirect address control word and field select control word representations. The latter three representations are not printed if the word is not legitimate for the individual representation. For example, a word would not be printed in instruction word representation if bits 15 through 20 of that word contain a bit configuration equivalent to an invalid operation code. Similarly, a word is not printed in indirect address control word representation if a one bit exists in positions 16 through 20. A word is not printed in field select control word representation if that word has a one bit in position 25, or the contents of bits 16 through 20 and 11 through 15 are legitimate representations of a left boundary and right boundary.

C. CALLING SEQUENCE

1. The memory dump procedure is included in the user's program by making a call to the procedure DUMP. This call must include

one list. The value of this list expression is the label of the entry line which must precede the call to the procedure. Following the procedure call should be an indirectly addressed jump to the entry line. An example of the calling sequence is:

```
PRNTMEMY   +     $
           DUMP  PRNTMEMY
           J     *PRNTMEMY
```

The label PRNTMEMY in the above examples, is assigned by the user and thus may have any value consistent with UTMOST label conventions.

2. In order to cause a memory dump the user codes the following sequence:

```
SLJ  PRNTMEMY
±    FROM
+    TO
...  return from memory dump
```

In the above example, the operand of the SLJ instruction is a label identical to label of the entry line preceding the procedure call. The label FROM is the label of the starting address of the area to be dumped. If the sign of this entry is + , the area dumped is printed in "program" format. If the sign of this entry is − , the area dumped is in "data" format. The label TO is the ending address of the area to be dumped.

D. MEMORY SPACE

The dump routine requires 581 words in memory, exclusive of the printer control routine.

E. EXAMPLE OF OUTPUT

An example of the output from a dump in data format followed by output in instruction format is shown on page 4. The first line of the dump printout contains the Control Counter contents in octal at the point at which the dump was entered, the contents of the arithmetic registers in octal and an indication of which (if any) of

the low (L), equal (E), or greater (G), indicators was set. The second line of the printout contains the contents of the index registers in octal. The following two lines of the dump printout are in the data format. On the left is the octal address of the first word on the line (four words are printed on each line). Each word is printed in octal, alphanumeric and decimal representation. The next two lines contain the control counter, arithmetic register, index register, and indicator contents on entry to the program format dump. The following lines are in program format. On the left of each line is the address in octal of the first word printed on the line (two words are printed on each line). Each word is represented in octal. To the right of the octal representation is printed the instruction word representation (if valid). The instruction word representation contains the mnemonic operation code, AR expression, octal operand address (preceded by * if position 25 of the word is a one bit), and the index register expression. To the right of the instruction word representation is the indirect address control word representation (if valid) containing the operand address (preceded by * if position 25 of the word is a one bit) and the index register. Following the indirect address control word representation is the field select control word representation (if valid), containing the left boundary, right boundary, operand address in octal and index register. All expressions in each representation (with the exception of operand addresses), appear in the same format as normally coded UTMOST expressions. The operation code J* indicates a conditional jump operation, such as JP, JL, or JS. Examination of the AR expression indicates which command is represented.

```
CC 23441  AR A  76543210 4  44556677 2  22334455 1  11223344   F
IR   1 00010 2 00020 3 00030 4 00040 5 00050 6 00060 7 00070 A 00017 9 00027 0 00100 1 00110 2 00120 3 27420  425420 5 23420
24401   21263611    :C 6  123456       -52741401   -PZ91 -7890 )       24252627    ABCD  212264       -30313233   -EFGH  316368
24405   34444546    IJKL  4-1633       11223344    6.HJ  -2-3=1       22334455    .HJ   1686-'        44556677    J T   6-''8(

CC 23444  AR A  76543210 4  44556677 2  22334455 1  11223344   G
IR   1 00010 2 00020 3 00030 4 00040 5 00050 6 00060 7 00070 A 00017 9 00027 0 00100 1 00110 2 00120 3 27420  425420 5 23420
23420   74502766  LA  1 0766 15                                          74504767  LA  2 0767 15
23422   74510770  LA  4 0770 15                 2  1 0770 15            74520771  LA  A 0771 15                 2  5 0771 15
23424   76442772  LX  1 0772 15                17 14 0772 15            76444773  LX  2 0773 15                17 15 0773 15
23426   76446774  LX  3 0774 15                17 16 0774 15            76450775  LX  4 0775 15                17 17 0775 15
23430   76452776  LX  5 0776 15                17 18 0776 15            76454777  LX  6 0777 15                17 19 0777 15
23432   76457000  LX  7 1000 15                17 20 1000 15            76461001  LX  A 1001 15                17 21 1001 15
23434   76463002  LX  9 1002 15                17 22 1002 15            76465003  LX 10 1003 15                17 23 1003 15
23436   76467004  LX 11 1004 15                17 24 1004 15            76471005  LX 12 1005 15
23440   74342032 SLJ  1 0032 15                                         -00024401 NOP 10*0401  0   *24401  0
23442   00024405 NOP 10 0405  0   24405  0                              74342032 SLJ  1 0032 15
23444   00023420 NOP  9 1420  0   23420  0                              00024530 NOP 10 0530  0    24530  0
23446   74342032 SLJ  1 0032 15                                         00000000 NOP  0 0000  0    00000  0
23450   00013560 NOP  5 1560  0   13560  0                              00342250 SLJ  1 0250  0
23452   00023444 NOP  9 1444  0   23444  0                              76422254  SX  9 0254 15                17  6 0254 15
23454   76420255  SX  8 0255 15                17  5 0255 15            76416256  SX  7 0256 15                17  4 0256 15
23456   76414257  SX  6 0257 15                17  3 0257 15            74436263  SA 15 0263 15                 1 12 0263 15
23460   77014043  J*  6 0043 15                21  3 0043 15            74503006  LA  1 1006 15
23462   74300044   J  0 0044 15                                         74502774  LA  1 0774 15
23464   77016050  J*  7 0050 15                21  4 0050 15            77012052  J*  5 0052 15                21  2 0052 15
23466   74402264  SA  1 0264 15                                         74300054   J  0 0054 15
23470   74643007  OR  1 1007 15                 3 14 1007 15            74300046   J  0 0046 15
23472   74643010  OR  1 1010 15                 3 14 1010 15            74300046   J  0 0046 15
23474  -76462032  LX  9*0032 15                                         76461011  LX  A 1011 15                17 21 1011 15
```

TEST DATA ASSEMBLY PROCEDURES

A. PURPOSE

To provide a method of creating data files of any format using UTMOST assembly.

B. METHOD

Data Files are written on tape using the Intermediate Tape Handling Routine. Three procedures - FILE, BLØCK and FINIS generate calling sequences to the output routines. Data words coded using UTMOST Data Word Generation (see UTMOST, Section II, page 18) following each call to the BLØCK procedure are written as a block on the output file.

C. USAGE

1. To specify a file of data the procedure call

   FILE s, r

   is used. s indicates the Uniservo on which the file is to be written. r indicates rewind if 1, or no rewind if 0.

2. To specify the contents of a block of the output file the procedure call BLØCK is used. Following this call are coded the words desired in the block. The user must code the entire content of all blocks, including data descriptors, label flags, and sentinels if these are desired.

3. The end of a block is indicated in the coding by the next procedure call BLØCK or FILE or FINIS.

4. To end the assembly of data files the procedure call FINIS is used. More than one file of data may be assembled in a single assembly.

5. The amount of data assembled in any single program is restricted to that which can be represented within a program consisting of 12000 words of object code.

6. An END directive should follow the FINIS procedure call. The label in the operand field of the END directive should be identical to the label in the label field of the first call to the FILE procedure.

7. The procedures FILE, BLØCK and FINIS are included in the source code for assembly by selecting FILEGEN on the ACCO run. The intermediate tape handling routines are included on the DECO run.

D. EXAMPLE

The sample coding on page 3 will generate a single file on Uniservo 4. Rewind is specified. The blocks are in order - a label block, a data block consisting of two 5-word items, and two end of file sentinel blocks.

```
1 LABEL  Δ OPERATION  Δ OPERAND
START FILE 4,1
      BLØCK
      - 0
      + 'FILE'
      + :161162
      + :000001
      + 0
      + 0
      + 0
      + 0
      + 0
      + 0
      + 'LABL'
      - 0
      BLØCK
      +020002
      +:123456
      + 'FIRST 5 '
      + 'WO ITEM '
      + :123789
      + 'SECØND 5'
      + 'WO ITEM '
      + 020002
      BLØCK
      - 060003
      BLØCK
      - 060004
      FIWIS
      END START
```

UNIVAC III FLOATING POINT PACKAGE

FPAC

A. PURPOSE

To provide the user with a floating point package of arithmetic, normalizing, and converting routines.

B. METHOD

The floating point package must be used as a unit, and will be included in the system as a packaged subroutine. A reference in UTMOST coding to one of the included subroutines will cause the entire FPAC package to be included with the user's object code at DECO time.

The following routines are included in FPAC and are available to the programmer:

ARITHMETIC ROUTINES

FAD	-	Floating Add (or subtract)
FMP	-	Floating Multiply
FDV	-	Floating Divide
DMP	-	Double Precision Multiply
DDV	-	Double Precision Divide

NORMALIZATION ROUTINES

NRM	-	Normalize

CONVERSION ROUTINES

FTI	-	Floating to Integer
ITF	-	Integer to Floating
FTD	-	Floating to Double Precision
DTF	-	Double Precision to Floating

C. DEFINITIONS

1. Floating Point Number

A UNIVAC III floating point number consists of a two digit exponent (excess fifty) followed by a ten digit mantissa. A twelve digit floating point number occupies two words of UNIVAC III memory. The signs of both words must be the same. The decimal point of the mantissa lies to the left of the high order digit of the mantissa (i.e. the magnitude of the mantissa is less than 1.0). Several examples of floating point numbers are shown below.

+EEMMMM +MMMMMM represents a two digit exponent EE (excess fifty) and a ten digit mantissa MMMMMMMMMM

+1.0 is represented as	+511000	+000000
-0.5 is represented as	-505000	-000000
+0.0 is represented as	+000000	+000000
or	+500000	+000000

2. Normalized

A floating point number is said to be normalized if the high order (leftmost) digit of the mantissa is greater than zero. A double precision number is said to be normalized if its high order digit is greater than zero.

3. Scale Factor

A scale factor is a two digit exponent (not excess fifty) that is associated with a double precision number. A scale factor may take on positive or negative values. A scale factor occupies one word of the UNIVAC III memory. Floating point numbers may be represented as double precision numbers with scale factors as shown below.

-511000 -000000	=	-100000 -000000	+000001
floating point		double precision	scale factor
+485000 +000000	=	+500000 +000000	-000002
floating point		double precision	scale factor

D. FLOATING POINT PACKAGE ROUTINES, CALLING SEQUENCE, ENTRY CONDITIONS AND EXIT CONDITIONS

1. Floating Add (or Subtract)

a. Purpose:

To compute the sum of two floating point numbers.

b. Calling Sequence:

```
SLJ   FAD
```

c. Entry Conditions:

AR8 - most significant part of the first floating point number
AR4 - least significant part of the first floating point number
AR2 - most significant part of the second floating point number
AR1 - least significant part of the second floating point number

d. Exit Conditions:

AR8 - most significant part of the floating point sum
AR4 - least significant part of the floating point sum
AR2 - xxxxxx
AR1 - Positive
HI, LO, and EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, ± 999999 is placed in AR8 and AR4. AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: The addition of two floating point numbers.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+551111	+111111	+552222	+222222
AFTER	+553333	+333333	xxxxxx	positive

g. Timing: 270 microseconds (average)

2. Floating Multiply

a. Purpose:

To compute the product of two floating point numbers.

b. Calling Sequence:

SLJ FMP

c. Entry Conditions:

AR8 - most significant part of the first floating point number
AR4 - least significant part of the first floating point number
AR2 - most significant part of the second floating point number
AR1 - most significant part of the second floating point number

d. Exit Conditions:

AR8 - most significant part of the floating point product
AR4 - least significant part of the floating point product
AR2 - xxxxxx
AR1 - Positive
HI, LO, and EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, $\pm$ 999999 is placed in AR8 and AR4. AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the product of two floating point numbers.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+521500	+000000	-53400	-000000
AFTER	-546000	-000000	xxxxxx	positive

g. Timing: 627 microseconds (average)

3. Floating Divide

a. Purpose:

To compute the quotient of two floating point numbers.

b. Calling Sequence:

SLJ FDV

c. Entry Conditions:

AR8 - most significant part of the dividend
AR4 - least significant part of the dividend
AR2 - most significant part of the divisor
AR1 - least significant part of the divisor

d. Exit Conditions:

AR8 - most significant part of the quotient
AR4 - least significant part of the quotient
AR2 - xxxxxx
AR1 - Positive
HI, LO, and EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If the divisor equals zero or floating point overflow occurs, ± 999999 is placed in AR8 and AR4. AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the quotient of two floating point numbers.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	-521500	-000000	-515000	-000000
AFTER	+513000	+000000	xxxxxx	positive

g. Timing: 840 microseconds (average)

4. Double Precision Multiply

a. Purpose:

To compute the product of two double precision numbers.

b. Calling Sequence:

SLJ DMP

c. Entry Conditions:

AR8 - most significant part of the first double precision number
AR4 - least significant part of the first double precision number
AR2 - most significant part of the second double precision number
AR1 - least significant part of the second double precision number

d. Exit Conditions:

AR8 - most significant part of the double precision product
AR4 - least significant part of the double precision product
AR2 - xxxxxx
AR1 - Positive
HI, LO, and EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

None

f. Example: Computing the product of two double precision numbers.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+333333	+333333	+300000	+000000
AFTER	+999999	+999999	xxxxxx	positive

g. Timing: 392 microseconds (average)

5. Double Precision Divide

a. Purpose:

To compute the quotient of two double precision numbers.

b. Calling Sequence:

SLJ DDV

NOTE: DDV is a special purpose divide routine that is used by the floating point mathematical function routines (Sin, etc). DDV should be used only if the divisor is normalized. The quotient produced will have an error not greater than three in the last digit. In most cases, the quotient will have no error.

c. Entry Conditions:

AR8 - most significant part of the double precision dividend
AR4 - least significant part of the double precision dividend
AR2 - most significant part of the double precision divisor
AR1 - least significant part of the double precision divisor

d. Exit Conditions:

AR8 - most significant part of the double precision quotient
AR4 - least significant part of the double precision quotient
AR2 - xxxxxx
AR1 - Positive
HI, LO, and EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If the magnitude of the dividend is greater than or equal to the magnitude of the divisor, it is considered an overflow, and ± 999999 is stored in AR8 and AR4. AR1 is set negative. If the most significant part of the divisor is zero, it is considered an overflow.

f. Example: Computing the quotient of two double precision numbers.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+100000	+000000	+700000	+000000
AFTER	+142857	+142857	xxxxxx	positive

g. Timing: 710 microseconds (average)

6. Normalize

a. Purpose:

To normalize a floating point number.

b. Calling Sequence:

SLJ NRM

c. Entry Conditions:

AR8 - most significant part of the unnormalized floating point number
AR4 - least significant part of the unnormalized floating point number
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the normalized floating point number
AR4 - least significant part of the normalized floating point number
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If underflow occurs, the number + 000000 is placed in AR8 and AR4.

f. Example: Normalizing a floating point number.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+510034	+987654	xxxxxx	xxxxxx
AFTER	+493498	+765400	xxxxxx	positive

g. Timing: 260 microseconds (average)

7. Floating to Integer

a. Purpose:

To convert a floating point number into a twelve digit integer.

b. Calling Sequence:

SLJ FTI

c. Entry Conditions:

AR8 - most significant part of the floating point number
AR4 - least significant part of the floating point number
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the integer
AR4 - least significant part of the integer
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If the integer is greater than 12 digits, it is considered an overflow, and $\pm$ 999999 is stored in AR8 and AR4. AR1 is set negative. If underflow occurs, the number + 000000 is placed in AR8 and AR4.

f. Example: Converting a floating point number into a 12 digit integer.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	-551234	-567890	xxxxxx	xxxxxx
AFTER	-000000	-012345	xxxxxx	positive

g. Timing: 152 microseconds (average)

8. Integer to Floating

a. Purpose:

To convert a twelve digit integer into a floating point number.

b. Calling Sequence:

SLJ ITF

c. Entry Conditions:

AR8 - most significant part of the integer
AR4 - least significant part of the integer
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the normalized floating point number
AR4 - least significant part of the normalized floating point number
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If underflow occurs, the number + 000000 is placed in AR8.

f. Example: Converting a 12 digit integer into a floating point number.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+123123	+121212	xxxxxx	xxxxxx
AFTER	+621231	+231212	xxxxxx	positive

g. Timing: 242 microseconds (average)

9. Floating to Double Precision

a. Purpose:

To convert a floating point number into a normalized double precision number with a two digit scale factor. The scale factor is not excess fifty and may take on positive or negative values.

b. Calling Sequence:

SLJ FTD

c. Entry Conditions:

AR8 - most significant part of the floating point number
AR4 - least significant part of the floating point number
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the normalized double precision number
AR4 - least significant part of the normalized double precision number
AR2 - scale factor in the two low order digits
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

None

f. Example: Converting a floating point number into normalized double precision number with a 2 digit scale factor.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+489876	+543210	xxxxxx	xxxxxx
AFTER	+987654	+321000	-000002	positive

g. Timing: 108 microseconds (average)

10. Double Precision to Floating

a. Purpose:

To convert an unnormalized double precision number with a two digit scale factor into a floating point number. The scale factor is not excess-fifty and may take on positive or negative values.

b. Calling Sequence:

SLJ DTF

c. Entry Conditions:

AR8 - most significant part of the unnormalized double precision number
AR4 - least significant part of the unnormalized double precision number
AR2 - scale factor in the two low order digits
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the floating point number
AR4 - least significant part of the floating point number
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, ± 999999 is placed in AR8 and AR4. AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Converting double precision to floating point.

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+000012	+345678	+000022	xxxxxx
AFTER	+681234	+567800	xxxxxx	positive

g. Timing: 238 microseconds (average)

E. STORAGE ALLOCATION AND TIMING CHART

FOR

UNIVAC III FLOATING POINT PACKAGE

Routine	Length		Average Time In Microseconds
FAD	59		270
FMP	32		627*
FDV	33		840**
DMP	13		392
DDV	23		710**
NRM	7		260***
FTI	26		152
ITF	15		242***
FTD	10		108
DTF	14		238***
DDV2	38	Internal Subroutines	
NRM2	47		
OFLO	4		
UFLO	3		
CONSTANTS AND ERASABLE STORAGE	74		
FPAC	398		

* Includes time for DMP

** Includes time for DDV2

*** Includes time for NRM2

UNIVAC III MATHEMATICAL PACKAGE

MATHPAC

A. PURPOSE

To provide the user with a set of routines for computing trigonometric, hyperbolic, exponential, and logarithmic functions, and for evaluating roots and powers of numbers.

B. METHOD

The mathematical package for the UNIVAC III consists of a set of routines for evaluating trigonometric, hyperbolic, exponential and logarithmic functions, and for finding roots and powers of numbers. This set of routines is called MATHPAC. In general, the routines in MATHPAC are independent of each other. A reference to one of the MATHPAC routines will cause the object code for that routine to be included with the user's object code at DECO time.

Some of the MATHPAC routines are not independent and require the presence of other MATHPAC routines. All the MATHPAC routines require the UNIVAC III Floating Point Package (FPAC). If one or more of the MATHPAC routines is used, FPAC will automatically be included with the user's object code at DECO time.

ROUTINES IN MATHPAC:

TRIGONOMETRIC FUNCTIONS

SIN -	Sine (x)
COS -	Cosine (x)
TAN -	Tangent (x)
TNGT-	Tangent (x)
ASIN -	Arcsine (x)
ACOS-	Arcosine (x)
ATAN-	Arctangent (x)

HYPERBOLIC FUNCTIONS

SINH - Hyperbolic-Sine (x)
COSH - Hyperbolic-Cosine (x)
TANH - Hyperbolic-Tangent (x)

ROOT FUNCTIONS

SQRT - Square Root (x)
CBRT - Cube Root (x)

EXPONENTIAL FUNCTIONS

EXP - e^x
TENX - 10^x

LOGARTHMIC FUNCTIONS

LOGN - Log (x) (Base e)
LOGT - Log (x) (Base 10)

POWER FUNCTIONS

XTOP - x^p

The following routines are not independent:

SUBROUTINE	OTHER ROUTINES USED
TNGT	SIN, SQRT
ASIS-ACOS	ATAN, SQRT
SINH	EXP
COSH	EXP
TANH	EXP
XTOP	TENX, LOGT

C. DEFINITIONS

1. Floating Point Number

A UNIVAC III floating point number consists of a two digit exponent (excess fifty) followed by a ten digit mantissa. A twelve digit floating point number occupies two words of UNIVAC III memory. The signs of both words must be the same. The decimal point of the mantissa lies to the left of the high order digit of the mantissa (i.e. the magnitude of the mantissa is less than 1.0). Several examples of floating point numbers are shown below:

+EEMMMMM +MMMMMMM represents a two digit exponent EE (excess fifty) and a ten digit mantissa MMMMMMMMMM

+1.0 is represented as	+511000	+000000
-0.5 is represented as	-505000	-000000
+0.0 is represented as	+000000	+000000
or	+500000	+000000

2. Normalized

A floating point number is said to be normalized if the high order (left-most) digit of the mantissa is greater than zero. All input to MATHPAC must be normalized floating point numbers. Unnormalized numbers will be treated as zeros.

D. MATHEMATICAL ROUTINES, CALLING SEQUENCES, ENTRY CONDITIONS AND EXIT CONDITIONS

1. Floating Sine or Cosine Routine

a. Purpose:

To compute the value of Sine (x) or Cosine (x), where x is a floating point number. x must be expressed in radians.

b. Calling Sequence:

```
SLJ    SIN    or
SLJ    COS
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Sine or Cosine
AR4 - least significant part of the Sine or Cosine
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If the magnitude of x is greater than 10^{10}, +000000 is placed in AR8 and AR4, and AR1 is set negative. If floating point underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Sine (.18)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 501800	+ 000000	xxxxxx	xxxxxx
AFTER	+ 501790	+ 295734	xxxxxx	positive

g. Length: 174

h. Time: SINE requires 3540 microseconds (average)
COSINE requires 3580 microseconds (average)

2. Floating Tangent Routine

a. Purpose:

To compute the value of Tangent (x), where x is a floating point number. x must be expressed in radians.

b. Calling Sequence:

```
SLJ   TAN
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Tangent
AR4 - least significant part of the Tangent
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If x is greater than 10^{10}, +000000 is placed in AR8 and AR4, and AR1 is set negative. If floating point overflow occurs, ±999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Tangent (.26)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 502600	+ 000000	xxxxxx	xxxxxx
AFTER	+ 502660	+ 215417	xxxxxx	positive

g. Length: 169

h. Time: 3780 microseconds (average)

3. Alternate Floating Tangent Routine

a. Purpose:

To compute the value of Tangent (x), where x is a floating point number. x must be expressed in radians.

b. Calling Sequence:

SLJ TNGT

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Tangent
AR4 - least significant part of the Tangent
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If x is greater than 10^{10}, +000000 is placed in AR8 and AR4, and AR1 is set negative. If floating point overflow occurs, $\pm$ 999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Tangent (.26)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 502600	+ 000000	xxxxxx	xxxxxx
AFTER	+ 502660	+ 215417	xxxxxx	positive

g. Length: 14 + SIN + SQRT

h. Time: 7500 microseconds (average)

4. Floating Arcsine-Arcosine Routine

a. Purpose:

To compute the value of Arcsine (x) or Arcosine (x), where x is a floating point number. The Arcsine (x) will lie in the interval (-PI/2, + PI/2). The Arcosine (x) will lie in the interval (O, PI).

b. Calling Sequence:

```
SLJ  ASIN    or
SLJ  ACOS
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Arcsine or Arcosine
AR4 - least significant part of the Arcsine or Arcosine
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If the magnitude of x is greater than 1.0, +000000 is placed in AR8 and AR4. AR1 is set negative. If floating point underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Arcosine (.1986693308)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 501986	+ 693308	xxxxxx	xxxxxx
AFTER	+ 502000	+ 000000	xxxxxx	positive

g. Length: 36 + ATAN + SQRT

h. Time: ARCSINE requires 8000 microseconds (average)
ARCCOSINE requires 8160 microseconds (average)

5. Floating Arctangent Routine

a. Purpose:

To compute the value of Arctangent (x), where x is a floating point number. The Arctangent (x) will lie in the interval (-PI/2, + PI/2).

b. Calling Sequence:

```
SLJ   ATAN
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Arctangent
AR4 - least significant part of the Arctangent
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Arctangent (.353736878)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 503537	+ 368780	xxxxxx	xxxxxx
AFTER	+ 503400	+ 000000	xxxxxx	xxxxxx

g. Length: 141

h. Time: 3990 microseconds (average)

6. Floating Hyperbolic Sine Routine

a. Purpose:

To compute the value of Hyperbolic-Sine (x), where x is a floating point number.

b. Calling Sequence:

```
SLJ    SINH
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Hyperbolic-Sine
AR4 - least significant part of the Hyperbolic-Sine
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, ± 999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Hyperbolic-Sine (.4)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 504000	+ 000000	xxxxxx	xxxxxx
AFTER	+ 504107	+ 523255	xxxxxx	positive

g. Length: 41 + EXP

h. Time: 6510 microseconds (average)

7. Floating Hyperbolic Cosine Routine

a. Purpose:

To compute the value of Hyperbolic-Cosine (x), where x is a floating point number.

b. Calling Sequence:

```
SLJ     COSH
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Hyperbolic Cosine
AR4 - least significant part of the Hyperbolic Cosine
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, ± 999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Hyperbolic-Cosine (.4)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 504000	+ 000000	xxxxxx	xxxxxx
AFTER	+ 511081	+ 072371	xxxxxx	positive

g. Length: 15 + EXP

h. Time: 6470 microseconds (average)

8. Floating Hyperbolic Tangent Routine

a. Purpose:

To compute the value of Hyperbolic Tangent (x), where x is a floating point number.

b. Calling Sequence:

```
SLJ    TANH
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Hyperbolic Tangent
AR4 - least significant part of the Hyperbolic Tangent
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Hyperbolic Tangent (.4)

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 504000	+ 000000	xxxxxx	xxxxxx
AFTER	+ 503799	+ 489621	xxxxxx	positive

g. Length: 53 + EXP

h. Time: 6260 microseconds (average)

9. Floating Square Root Routine

a. Purpose:

To compute the value of the Square Root of x, where x is a floating point number.

b. Calling Sequence:

SLJ SQRT

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Square Root
AR4 - least significant part of the Square Root
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If x is negative, the square root of the absolute value of x is computed. AR1 is set to negative.

f. Example: Computing the Square Root of .44

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 504400	+ 000000	xxxxxx	xxxxxx
AFTER	+ 506633	+ 249581	xxxxxx	positive

g. Length: 95

h. Time: 2160 microseconds (average)

10. Floating Cube Root Routine

a. Purpose:

To compute the value of the Cube Root of x, where x is a floating point number.

b. Calling Sequence:

```
SLJ   CBRT
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Cube Root
AR4 - least significant part of the Cube Root
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

None

f. Example: Computing the value of the Cube Root of .44

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+504400	+000000	xxxxxx	xxxxxx
AFTER	+507605	+904922	xxxxxx	positive

g. Length: 96

h. Time: 2960 microseconds (average)

11. Floating e^x or 10^x Routine

a. Purpose:

To compute the value of e^x or 10^x, where x is a floating point number.

b. Calling Sequence:

SLJ EXP (for e^x) or
SLJ TENX (for 10^x)

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of e^x or 10^x
AR4 - least significant part of e^x or 10^x
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If floating point overflow occurs, +999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of $e^{.16}$

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 501600	+ 000000	xxxxxx	xxxxxx
AFTER	+ 511173	+ 510871	xxxxxx	positive

g. Length: 122

h. Time: e^x requires 4420 microseconds (average)
10^x requires 3780 microseconds (average)

12. Floating Log (Base e) or Log (Base 10) Routine

a. Purpose:

To compute the value of Log (x) (base e) or Log (x) (base 10), where x is a floating point number.

b. Calling Sequence:

SLJ LOGN for log (base e) or
SLJ LOGT for Log (base 10)

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - xxxxxx
AR1 - xxxxxx

d. Exit Conditions:

AR8 - most significant part of the Log
AR4 - least significant part of the Log
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If x is negative, the Log of the absolute value of x is computed, and AR1 is set negative. If floating point overflow occurs, -999999 is placed in AR8 and AR4, and AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of Log $e^{1.584073985}$

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+ 511584	+ 073985	xxxxxx	xxxxxx
AFTER	+ 504600	+ 000000	xxxxxx	positive

g. Length: 111

h. Time: Log (base e) requires 4360 microseconds (average)

Log (base 10) requires 3700 microseconds (average)

13. Floating x^p Routine

a. Purpose:

To compute the value of x^p where x and p are both floating point numbers. The sign of x^p will always be positive.

b. Calling Sequence:

```
SLJ   XTOP
```

c. Entry Conditions:

AR8 - most significant part of x
AR4 - least significant part of x
AR2 - most significant part of p
AR1 - least significant part of p

d. Exit Conditions:

AR8 - most significant part of x^p
AR4 - least significant part of x^p
AR2 - xxxxxx
AR1 - Positive
HI, LO, EQ INDICATORS - may be altered
SENSE INDICATORS - unaltered
INDEX REGISTERS - unaltered

e. Special Conditions:

If x is negative, the absolute value of x is raised to the p-th power. If floating point overflow occurs, +999999 is placed in AR8 and AR4. AR1 is set negative. If underflow occurs, +000000 is placed in AR8 and AR4.

f. Example: Computing the value of $(.6839903787)^{3.2}$

CONTENTS	AR8	AR4	AR2	AR1
BEFORE	+506839	+903787	+513000	+000000
AFTER	+503200	+000000	xxxxxx	positive

g. Length: 8 + TENX + LOGT

h. Time: 8150 microseconds (average)

EDITING ROUTINES

A. Purpose

To provide a means of editing input or output information on a character by character basis. These routines have the ability to delete or insert blanks and to accept octal, decimal or alphanumeric information. These routines do not include any binary decimal conversions.

B. Method

The input edit routine accepts 6 bit characters and transforms them to bites of 6 bits or less. The output editing routine accepts bites of from 1 to 6 bits and transforms them to 6 bit characters.

The editing format codes provided by the calling program as 4-bit X-S 3 numerics act as the heart of the processing for the editing routines. The editing routines process information by examining these format codes on a 4 bit bite-by-bite basis and making a corresponding interpretation of the next sequential input or output character. This resulting character is then placed in the appropriate portion of the word or words being created. The editing format codes are written as decimal digits using as many words as are required to assemble or disassemble (as the case may be) the word or words of information. It is possible to create multiple edited words on input and to edit multiple words on output through the use of repeated linkages to the editing routines.

C. Memory Space

The editing routines occupy approximately 75 words.

D. Operating Procedures

1. Editing Routines

a. Input

The input edit (IE) routine accepts alphanumeric (6 bit) characters one character at a time, and contracts them to 1, 2, 3, 4, 5, or 6 bit edited bites. Blanks may be removed anywhere in a word. This process is terminated by the digit '1' in an editing format word, usually after a complete word has been assembled.

For example, if the format codes consisted of all 4's then 6 whole words and 1 partial word of alphanumeric information would be contracted to 1 edited word (see example 1c in Section F). The edit routine will automatically pick up the subsequent input words.

To utilize the input edit routine the arithmetic and index registers noted below should be loaded as indicated and an SLJ IE instruction executed. When the editing is complete the routine will return control with the assembled word in AR8.

b. Output

The output edit (OE) routine accepts 1, 2, 3, 4, 5 or 6 bit bites, one bite at a time and expands them to 6 bit edited characters. Blanks may be inserted anywhere in a word. This process is terminated by the digit '1' in an editing format word, usually after a whole word has been disassembled. For example, if the format codes consisted of all 4's then 1 data word would be expanded to 6 whole words and 1 partial word of 6 bit characters. (See example 2f in Section F). The edit routine will automatically store each output word in consecutive locations as specified by the calling program.

2. Calling Sequences

The following sequences are normally used for communication with the editing routines:

a. SLJ IE — Edit input information. Assembled word will be left in AR8.

Initial Register Settings

AR1 = first word of input information
AR2 = :000000
IR2 = location of first input word
IR4 = starting address minus 1 of format codes

b. SLJ OE — Disassemble contents of AR1 and output as consecutive characters. Partial word may be left in AR8. Use PNC (see below) to drain out these additional characters.

Initial Register Settings

AR1 = word to be disassembled
AR2 = :000000
AR8 = 1 (causes full word to be gathered before return)
IR3 = location of first word of output information
IR4 = location of first format codes word minus 1

c. SLJ GNC — Get next character (GNC) will place the next input character in bit positions 1-6 of AR4. The rest of AR4 will be zero. This routine will ordinarily be used only by the edit routines themselves.

d. SLJ PNC — Put next character (PNC) will putput bits 1-6 of AR4 into the next consecutive output character. This routine may be used to drain out partially filled output words upon exit from the OE routine.

D. 3. Multiple word editing

Editing of more than one word of input or output may be accomplished by repeated linkages to the proper routine. The index registers shown above will be incremented at the time of return to accept the next consecutive words. In the case of input the calling routine may store the assembled word and reenter the IE routine. In the case of output the next information word to be disassembled may be loaded in AR1 and the OE routine reentered. On output, if it is desired to add to a partially accumulated word in AR8, AR8 should be left intact; otherwise it should be drained out on a character by character basis using linkages to PNC (see above).

E. Editing Codes

Editing codes are written as decimal digits.

Code 0 indicates the end of a format code word. The editing routine will automatically continue on to the next format code word. It is not necessary to end a format code word with zero as in this case the routine will automatically recognize the end of word and continue.

Code 1 terminates an editing linkage.

Code 2 is normally used for output editing only. A code 2 on output will cause a bit to be represented as ' - '. A code 2 on input editing should only be used to interpret the first bit of a word. Codes 2, 4, 5 and 6 all treat a ' - ' as a 1 bit during input editing.

Code 3 (space) will cause a space to be skipped on input or generated for output.

Codes 4, 5 or 6 will treat the information as an octal representation of the respective number of bits 1, 2 and 3. On input they will cause the information to be converted from excess-three representation and on output into excess-three.

Codes 7, 8 or 9 will cause the information to be transcribed directly with only a contraction or expansion in the number of bits 4, 5 and 6 as specified.

Editing code table:

0	end of format code word	
1	end of format code set	
2	sign	
3	space	
4	1 bit octal	implies excess-three conversion
5	2 bits octal	
6	3 bits octal	
7	4 bits (decimal)	
8	5 bits	
9	6 bits (alphanumeric)	

F. Examples

1. Input

	DATA FORMAT	FORMAT CODE WORDS	EDITED DATA FORMAT
a.	Alphanumeric	+277777 +710000	-999999
b.	Alphanumeric	+279977 +100000	-9AA99
c.	Alphanumeric	+444444 +444444 +444444 +444444 +410000	111111111111111111111111

2. Output

	DATA FORMAT	FORMAT CODE WORDS	EDITED DATA FORMAT
a.	1 word octal	+266666	+-777
		+666335	+7777
		+100000	+7SSS
b.	1 word decimal	+327777	+S-99
		+771000	+9999
c.	1 word alphanumeric	+339339	+SSAS
		+991000	+SAAA
d.	1 word instruction	+324630	+S-17
		+663463	+S77S
		+466631	+17S1
			+777S
e.	1 word mixed information	+637335	+7S9S
		+373353	+S3S9
		+733537	+SS3S
		+331000	+9SS3
			+S9SS
f.	1 word binary	+444444	+AAAA
		+444444	+AAAA
		+444444	+AAAA
		+444444	+AAAA
		+410000	+AAAA
			+AAAA
			+ASSS

where in the data and edited data words

7 indicates digits 0-7
3 indicates digits 0-3
1 indicates digits 0-1
9 indicates digits 0-9
A indicates any alphanumeric character
S indicates space
- indicates either - or space on output and the sign bit on input
+ indicates sign is ignored

and the format codes in the format code words are as described in Section E.2.

NOTE: The Editing Routines in the typewriter control section of the executive system should not be used unless contingency interrupt is prevented. In utilizing these Editing Routines with the ALMOST assembly system, standard EQU cards should be placed ahead of the ALMOST symbolic deck which is to be assembled. In this manner, the labels for the Editing Routines will be equated with proper absolute addresses.

If the Editing Routines of the typewriter control section are used then the following labels are restricted from other use in the source program:

IE
OE
GNC
PNC

If the Editing Routines are included in the ALMOST symbolic source program as a separate sub-routine, then the above restrictions do not apply. Instead the following labels must be excluded from other use in the source program:

IE
OE
GNC
PNC
EC1
EC2
EC3
GNM
IE4
IE2
OE1
OE2

July 20,1962

UNIVAC III SUPPORT U-3519

UPDATING PACKAGE A

The attached sheets are the first addition to the SUPPORT III Manual. There are six routines comprising a total of 32 pages. These routines should be read carefully.

BOOT	3-0001
WRITE SYSTEM TAPE	3-0002
ON-LINE MEMORY DUMP	4-0002
EDITING ROUTINES	6-0001
MOVE PROCEDURE	6-0002
FLOATING DOLLAR SIGN ROUTINE	6-0003

These routines should be placed in the manual by their numbers and this page filed directly after the INDEX until a new INDEX is received.

October 10, 1962

UNIVAC III SUPPORT U-3519

UPDATING PACKAGE B

The attached sheets contain important additions to the SUPPORT III Manual. These routines should be read carefully.

INTERMEDIATE TAPE HANDLING ROUTINE 1-0004

TAPE INPUT-OUTPUT ITEM HANDLING ROUTINE 1-0005

These routines should be placed in the manual by their numbers and this page filed directly after the INDEX until a new INDEX is received.

The following routines should be removed from the SUPPORT III Manual and destroyed. Corrected documentation for these routines can be found in the UTMOST Manual.

MOVE PROCEDURE 6-0002

FLOATING DOLLAR PROCEDURE 6-0003

November 16, 1962

UNIVAC III SUPPORT U-3519

UPDATING PACKAGE C

The attached sheets contain additions to the SUPPORT III Manual.

PUNCHED PAPER TAPE READER SYMBIONT	2-0005
ON-LINE EDITED MEMORY DUMP	4-0002
TEST DATA ASSEMBLY PROCEDURES	4-0003

These routines should be placed in the manual in sequence by their section numbers, and this page filed directly after the INDEX until a new INDEX is received.

January 15, 1963

UNIVAC III SUPPORT U-3519

UPDATING PACKAGE D

The attached sheets contain important corrections and additions to the SUPPORT III Manual.

INDEX Replace entire section		3 pages
INTERMEDIATE TAPE HANDLING ROUTINE Replace entire section	1-0004	5 pages
TAPE INPUT-OUTPUT ITEM HANDLING ROUTINE Replace entire section	1-0005	23 pages
TAPE INPUT-OUTPUT VARIABLE SIZE ITEM HANDLING Add (new section)	1-0006	17 pages
FLOATING POINT PACKAGE Add (new section)	5-0001	13 pages
MATHEMATICAL PACKAGE Add (new section)	5-0002	16 pages

March 27, 1963

UNIVAC III
SUPPORT, U–3519

UPDATING PACKAGE E

The attached sheets contain major additions to the SUPPORT III Reference Manual, Section 3 (Utility Routines).

INDEX Replace entire section		3 pages
UPCO (UPdating COntrol) Add (new section)	3–0003	16 pages
ACCO (Assembler Compiler COntrol) Add (new section)	3–0004	18 pages
DECO (DEsignation COntrol) Add (new section)	3–0005	38 pages

This sheet should be retained and inserted after the INDEX to serve as a catalogue of change.

Remington Rand

INTERCOMMUNICATION

TO: UNIVAC III Programmers

FROM (NAME): R. Klose

LOCATION & DATE: New York - April 9, 1963

DEPARTMENT: UNIVAC Data Processing Center

CARBONS:

SUBJECT: UNIVAC III Information Exchange #NY 1

UTMOST Routines

The following routines are available and debugged and may help you in programming in UTMOST. They are procedures and may be called in your program thru SELECT cards at ACCO time. The calling sequence in your coding is indicated.

*** Indicates routine is not on Symbolic library from CSC, but is obtainable from New York Data Processing Center.

DATA (i.e., SELECT DATA)***

Generates constants in UTMOST FORMATS.

Call is

```
DATA      LIST1   LIST2   LIST3     etc.
```

Where LISTn is a list of 1 thru 4 expressions which will generate data words, indirect address control words, etc., according to UTMOST data word generation rules.

DUMP ***

Provides BOSS III on-line memory dump calling sequences for a variety of dump formats including BOSS III formats, any special formats you would care to define in accordance with the description in SUPPORT III (#4-0001) and three special formats as below. Also takes care of repositioning paper in Printer.

Call is

```
DUMP       F, S, E, SIR, EIR        1
     or
DATADUMP   S, E, SIR, EIR           1
     or
PROGDUMP   S, E, SIR, EIR           1
     or
BOTHDUMP   S, E, SIR, EIR           1
```

Where F is format code or address of edit list (see #4-0001 of SUPPORT III),
S, SIR are the starting address and IR (if any) of area to be dumped (see #4-0001 of SUPPORT III)
E, EIR are ending address and IR as above
1 is 1 if Registers and Indicators are desired, blank or zero if not.
DATADUMP call edits each word in alpha, octal and decimal format, 4 words to a line.
PROGDUMP call edits each word in octal, instruction, indirect address, and field select control word formats, 2 words to a line.
BOTHDUMP call edits each word in all formats of DATADUMP and PROGDUMP, 1 word to a line.

EDITPROC ***

Provides the editing subroutines described in SUPPORT III, #6-0001. The feasibility of using the identical coding occurring in BOSS III is nil. To do so, you would have to overlay computer absolute location 19 so that all contingency interrupts can be ignored while using these subroutines. Should be called in some convenient area of your program and then the calling sequences described in SUPPORT III can be used.
Call is
EDIT.

JMS

Provides Jump Minus with consistent AR designation. Uses JPS procedure.
Call is
JMS AR,M,IR

JN ***

Provides consistent sense indicator designation and convenient mnemonic code.
Call is
Jn M,IR
or
Sn
or
Rn

Where n stands for Sense Indicator 1 thru 8
Jn is Jump if Sense Indicator n is set.
Sn is Set Sense Indicator n
Rn is Reset Sense Indicator n

JNL

Provides Jump if not less, if not equal, if not greater.
Call is

```
JNL     M,IR
        or
JNE     M,IR
        or
JNG     M,IR
```

JPS

Provides consistent AR designation for Jump Positive.
Call is

```
JPS     AR,M,IR
```

LAED

Provides consistent memory addressing for LAE.
Call is

```
LAED    AR,M,IR
```

QBR ***

Provides convenient mnemonic for generating sets of instructions for BRanching in your program.
Call is

```
BRX   AR   FLD,ADD,FIR,AIR   FLD,ADD,FIR,AIR     etc.
```

Where BRX is
BRL for branch if less
BRE for branch if equal
BRG for branch if greater
BRNL for branch if not less
BRNE for branch if not equal
BRNG for branch if not greater
AR is the Arithmetic Register(s) specified.
FLD,FIR is the address and IR of field to be tested.
ADD,AIR is the address and IR to jump to if branching condition is met.

TYPEOUT

Provides BOSS III calling sequence for typeout control.
Call is

```
TYPEOUT (A)
    or
TYPEOUT (Y)
```

Where A is the address of the start of the message
Y is the location of A.

In addition to the foregoing, the following are in the works or almost ready:

MOVE - a procedure using a set of closed subroutines to provide a wide variety of calls to move and fill areas of memory efficiently where the necessary parameters can be provided at ACCO time or at object time or any combination thereof.

INITIAL - a proc to completely automate mapping within an UTMOST program and provide a running check on index coverage. Also will, when completed, provide a wide variety of convenient mnemonics including SALT mnemonics. Provides Typeout and Type-in Control and a portion of initialization.

LTRL - a proc, similar to DATA, to generate literal constants, i.e., constants to appear in the constant pool rather than at the spot in your program where they are called.

TYPEIN - Due to changes to UTMOST and the elimination of the necessity of using RES 010000 in programs, the standard TYPEIN proc no longer works. A change to it will be available shortly.

SNAPSHOT and SNIPSHOT, procs to generate open and closed subroutine memory dumps, respectively, of a snapshot nature. Similar to dump, but can call a large number of format-area combinations with a single call. First the registers and indicators and the address of the call are printed and then your specific calls.

Procs are planned to generate coding to edit input images into a series of right (or left?) justified fields of desired formats and, conversely, to edit a series of right (or left?) justified fields of various formats into an edited line image with facility of decimal point, blank, and other character deletion and insertion and zero suppression.

The following routines are as yet unavailable (SUPPORT III):
Paper Tape Symbiont #2-0005
On-line edited Mem. Dump #4-0002.

The latter will probably be provided as a relocatable program rather than a proc as described.

Great care should be exercised in using some of the above procedures. In particular, JMS, JNL, and QBR, while looking like typical instruction lines, actually generate two or, in the case of QBR, sometimes more lines of object coding, a fact which can be easily overlooked when using reflexive addressing in their vicinity.

R. KLOSE

wc

SUPPORT

PRINTED IN U.S.A. U-3519