UNIVAC

DATA PROCESSING DIVISION

9200/9300

SYSTEMS

PROGRAMMING UTILITY

REFERENCE MANUAL

UP-4120

This manual is published by the Univac Division of Sperry Rand Corporation in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC® Systems developments. The information presented herein may not reflect the current status of the programming effort. For the current status of the programming, contact your local Univac Representative.

The Univac Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The Univac Division reserves the right to make such additions, corrections, and/or deletions as in the judgment of the Univac Division are required by the development of its respective Systems.

PRINTED IN U.S.A.

CONTENTS

1. CONVENTIONS

The purpose of this manual is to describe the conventions used in the UNIVAC 9200/9300 System software package and to explain the general utility routines and programs provided with the UNIVAC 9200/9300 System.

Certain uniform conventions and standards have been adopted in the UNIVAC 9200/9300 System to allow the system to be operated in a consistently efficient manner.

Information necessary for an understanding of the material contained in this manual is presented in the following documents:

UNIVAC 9200 System, System Description Manual – UP-4086

UNIVAC 9300 System, System Description Manual – UP-4119

UNIVAC 9200/9300 System Gangpunch-Reproduce Program, Reference Manual – UP-4089

UNIVAC 9200/9300 System Card Report Program Generator, Reference Manual – UP-4106

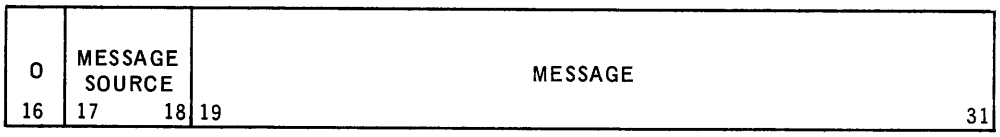
UNIVAC 9200/9300 System Card Assembler, Reference Manual – UP-4092

1.1. STANDARD CARD, EBCDIC, AND PRINTER GRAPHIC CODES

Because of the many card codes and internal computer codes presently used in data processing systems, and because of the interrelationships of these codes to each other and to printer graphics, the UNIVAC 9200/9300 System permits data to be translated whenever it passes from an input unit to the processor memory or from the memory to an output unit. However, the design of the UNIVAC 9200/9300 software requires standard card, internal, and printer graphic codes. These codes and their interrelationships are shown in Table 1–1.

1.2. CONSOLE DISPLAY CONVENTIONS

Indications of certain conditions within the computer are given to the operator by means of a Halt and Proceed (HPR) instruction. This instruction causes the halfword effective address (bits 16–31) of the instruction to be displayed on the console.



(1) Bit 16 is always zero to inhibit indexing.

(2) If bit 17 = 0, the display message is initiated by the problem program.

a. If bit 18 = 0, the message is initiated by the user.

b. If bit 18 = 1, the message comes from a Univac subroutine incorporated in the problem program.

TWO MOST SIGNIFICANT BITS OF ZONE – 00

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000	12-0-9-8-1	12-11-9-8-1	11-0-9-8-1	12-11-0-9-8-1
0001	12-9-1	11-9-1	0-9-1	9-1
0010	12-9-2	11-9-2	0-9-2	9-2
0011	12-9-3	11-9-3	0-9-3	9-3
0100	12-9-4	11-9-4	0-9-4	9-4
0101	12-9-5	11-9-5	0-9-5	9-5
0110	12-9-6	11-9-6	0-9-6	9-6
0111	12-9-7	11-9-7	0-9-7	9-7
1000	12-9-8	11-9-8	0-9-8	9-8
1001	12-9-8-1	11-9-8-1	0-9-8-1	9-8-1
1010	12-9-8-2	11-9-8-2	0-9-8-2	9-8-2
1011	12-9-8-3	11-9-8-3	0-9-8-3	9-8-3
1100	12-9-8-4	11-9-8-4	0-9-8-4	9-8-4
1101	12-9-8-5	11-9-8-5	0-9-8-5	9-8-5
1110	12-9-8-6	11-9-8-6	0-9-8-6	9-8-6
1111	12-9-8-7	11-9-8-7	0-9-8-7	9-8-7

Table 1–1. Internal Code (EBCDIC), Sheet 1 of 4

TWO MOST SIGNIFICANT BITS OF ZONE – 01

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000		12	11	12-11-0
0000	¬	&	–	
0000				
0001	12-0-9-1	12-11-9-1	0-1	12-11-0-9-1
			/	
0010	12-0-9-2	12-11-9-2	11-0-9-2	12-11-0-9-2
0011	12-0-9-3	12-11-9-3	11-0-9-3	12-11-0-9-3
0100	12-0-9-4	12-11-9-4	11-0-9-4	12-11-0-9-4
0101	12-0-9-5	12-11-9-5	11-0-9-5	12-11-0-9-5
0110	12-0-9-6	12-11-9-6	11-0-9-6	12-11-0-9-6
0111	12-0-9-7	12-11-9-7	11-0-9-7	12-11-0-9-7
1000	12-0-9-8	12-11-9-8	11-0-9-8	12-11-0-9-8
1001	12-8-1	11-8-1	0-8-1	8-1
1010	12-8-2	11-8-2	12-11	8-2
	¢	!		:
1011	12-8-3	11-8-3	0-8-3	8-3
		$	,	#
1100	12-8-4	11-8-4	0-8-4	8-4
	<	*	%	@
1101	12-8-5	11-8-5	0-8-5	8-5
	(	)	—	'
1110	12-8-6	11-8-6	0-8-6	8-6
	+	;	>	=
1111	12-8-7	11-8-7	0-8-7	8-7
	|	¬	?	"

Table 1–1. Internal Code (EBCDIC), Sheet 2 of 4

TWO MOST SIGNIFICANT BITS OF ZONE – 10

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000	12-0-8-1	12-11-8-1	11-0-8-1	12-11-0-8-1
0001	12-0-1	12-11-1	11-0-1	12-11-0-1
0010	12-0-2	12-11-2	11-0-2	12-11-0-2
0011	12-0-3	12-11-3	11-0-3	12-11-0-3
0100	12-0-4	12-11-4	11-0-4	12-11-0-4
0101	12-0-5	12-11-5	11-0-5	12-11-0-5
0110	12-0-6	12-11-6	11-0-6	12-11-0-6
0111	12-0-7	12-11-7	11-0-7	12-11-0-7
1000	12-0-8	12-11-8	11-0-8	12-11-0-8
1001	12-0-9	12-11-9	11-0-9	12-11-0-9
1010	12-0-8-2	12-11-8-2	11-0-8-2	12-11-0-8-2
1011	12-0-8-3	12-11-8-3	11-0-8-3	12-11-0-8-3
1100	12-0-8-4	12-11-8-4	11-0-8-4	12-11-0-8-4
1101	12-0-8-5	12-11-8-5	11-0-8-5	12-11-0-8-5
1110	12-0-8-6	12-11-8-6	11-0-8-6	12-11-0-8-6
1111	12-0-8-7	12-11-8-7	11-0-8-7	12-11-0-8-7

Table 1–1. Internal Code (EBCDIC), Sheet 3 of 4

TWO MOST SIGNIFICANT BITS OF ZONE – 11

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000	12-0	11-0	0-8-2	0 0
0001	12-1 A	11-1 J	11-0-9-1	1 1
0010	12-2 B	11-2 K	0-2 S	2 2
0011	12-3 C	11-3 L	0-3 T	3 3
0100	12-4 D	11-4 M	0-4 U	4 4
0101	12-5 E	11-5 N	0-5 V	5 5
0110	12-6 F	11-6 O	0-6 W	6 6
0111	12-7 G	11-7 P	0-7 X	7 7
1000	12-8 H	11-8 Q	0-8 Y	8 8
1001	12-9 I	11-9 R	0-9 Z	9 9
1010	12-0-9-8-2	12-11-9-8-2	11-0-9-8-2	12-11-0-9-8-2
1011	12-0-9-8-3	12-11-9-8-3	11-0-9-8-3	12-11-0-9-8-3
1100	12-0-9-8-4	12-11-9-8-4	11-0-9-8-4	12-11-0-9-8-4
1101	12-0-9-8-5	12-11-9-8-5	11-0-9-8-5	12-11-0-9-8-5
1110	12-0-9-8-6	12-11-9-8-6	11-0-9-8-6	12-11-0-9-8-6
1111	12-0-9-8-7	12-11-9-8-7	11-0-9-8-7	12-11-0-9-8-7

Table 1–1. Internal Code (EBCDIC), Sheet 4 of 4

(3) If bit 17 = 1, the message is initiated by either the Supervisor or an input/output routine.

a. If bit 18 = 0, the message is from the Supervisor.

b. If bit 18 = 1, the message is from an input/output routine.

(4) Bits 19–31 contain the message.

In the case of input/output routines, the two-byte displays adhere to the following conventions:

	HEXADECIMAL DIGIT	ALLOWABLE VALUE	REMARKS
Byte 1	MSD	6	Indicates I/O message
	LSD	0, 1, 2, ...F	Device identification
Byte 2	MSD	Any combination of two hexadecimal digits	Message information
	LSD		

The message information characters need not be used if the console display lights give enough information for the operator to interpret the error. This convention keeps the number of displays to a minimum.

Devices are identified by a hexadecimal digit in the least significant digit of byte 1. Digit assignments for the system input/output devices are as follows:

HEXADECIMAL DIGIT	DEVICE
1	Online card reader
2	Online serial read/punch
3	Online bar printer
5	1001 Card Controller

1.3. LABEL AND FORMAT CONVENTIONS FOR MAGNETIC TAPES

Tapes may have standard or nonstandard labels, or they may be unlabeled. Tape data may have fixed or variable record lengths and may be in blocked or unblocked format. Data may also be in an undefined format of variable-length records. One or more files may appear on a tape.

1.3.1. Standard Labels

A tape with standard labels must have a header label and a trailer label. Tape marks are used to separate data and labels. A tape with standard labels has the following features:

- *Volume Label* – As many as eight volume labels may appear at the beginning of a tape. A volume label is 80 bytes long; the first three bytes are VOL.

- *File Header Label* – At least one file header label must, and as many as eight may, appear at the beginning of a file on a tape. A file header label is 80 bytes long; the first three bytes are HDR. The fourth byte of the first file header label is a 1. A file identifier appears in bytes 5 through 12; a volume number in bytes 30 and 31 (volumes are counted from one); a generation number in bytes 36 through 39; a creation date in bytes 42 through 47; and an expiration date in bytes 48 through 53. (The expiration date is the first date on which a magnetic tape is to be considered a scratch.) Dates appear in the format bYYDDD, where YY are the two least significant digits of the year, and DDD are the day of the year.

- *User Header Label* – As many as eight user header labels may follow the last file header label. A user header label is 80 bytes long, and the first three bytes are UHL.

- *Data* – Data blocks follow the tape mark to indicate the end of the header labels. The data format is explained under heading 1.3.4.

- *End-of-Reel Trailer Label* – If more data for a file is to follow on another tape, at least one end-of-reel label must, and as many as eight may, appear after the tape mark that indicates the end of the data. An end-of-reel label is 80 bytes long, the first three bytes are EOV, and the fourth byte of the first end-of-reel label is a 1. A file identifier appears in bytes 5 through 12; a volume number in bytes 30 and 31; a generation number in bytes 36 through 39; a creation date in 42 through 47; an expiration date in 48 through 53; and a block count in 55 through 60. Only data blocks are counted in the block count. Labels, checkpoint records, and tape marks are not counted.

- *End-of-File Trailer Label* – If data for a file is completed on a tape, at least one end-of-file label must, and as many as eight may, appear after the tape mark that indicates the end of the data. An end-of-file label is 80 bytes long, the first three bytes are EOF, and the fourth byte of the first end-of-reel label is a 1. A file identifier appears in bytes 5 through 12, a volume number in bytes 30 and 31; a generation number in bytes 36 through 39; a creation date in 42 through 47; an expiration date in 48 through 53; and a block count in 55 through 60.

- *User Trailer Label* – As many as eight user trailer labels may follow the last end-of-reel label or the last end-of-file label. A user trailer label is 80 bytes long, and the first three bytes are UTL.

- *Tape Mark* – The tape mark is a hardware-produced configuration. A tape mark follows the last header label, the last data block on the tape or the file, and the last trailer label in an end-of-reel or end-of-file series. The tape mark following the last trailer label on a tape is followed by a second tape mark.

1.3.2. Nonstandard Labels

A tape with nonstandard labels must have a header label and a trailer label with a tape mark between the data and the trailer labels. A tape mark between the header labels and the data is optional. A tape with nonstandard labels has the following features:

- *Nonstandard Header Label* – Any number of nonstandard header labels may appear at the beginning of a file. Their length and format are unrestricted.

- *Data* – If there is a tape mark that indicates the last header label, data blocks follow the tape mark. Otherwise, data blocks follow the last header label.

- *Nonstandard Trailer Labels* – Any number of nonstandard trailer labels may appear after the tape mark which indicates the end of the data. Their length and format are unrestricted.

- *Tape Mark* – The tape mark following the last header label is optional. One or two tape marks may or may not follow the last trailer label. A tape mark must follow the last data block on the tape or in the file.

1.3.3. Unlabeled

An unlabeled tape may or may not have a tape mark at the beginning of a file. If a tape mark is used, data blocks follow the tape mark that indicates the beginning of the file. Otherwise, the file begins with data blocks. One tape mark must, and two tape marks may, follow the last data block in the file.

1.3.4. Data Format

Data may have fixed or variable record lengths and may be in blocked or unblocked format. For fixed-length records, whether blocked or unblocked, only data is recorded on tape.

For variable-length records, whether blocked or unblocked, the first four bytes of a *block* are

nnbb

where nn is a binary number which represents the number of bytes in the block including the first four bytes, and bb is reserved for system use.

For variable-length records, whether blocked or unblocked, the first four bytes of a *record* are

nnbb

where nn is a binary number which represents the number of bytes in the record including the first four bytes, and bb is reserved for system use.

Data may also be in an undefined format. Such data is of variable-length and unblocked but need not conform to the conventions given for variable-length records.

All standard information recorded on tape by the IOCS is represented in standard EBCDIC internal code.

1.3.5. Checkpoint Dump

A checkpoint dump may be made on an output file. The first and last blocks of the dump begin with the following configuration:

///␣CHKPT␣//

The number of blocks in the dump, their length, and their format are unrestricted.

2. UTILITY PROGRAMS AND SUBROUTINES

This section contains a brief description of the various utility programs and subroutines supplied with the UNIVAC 9200/9300 System software.

2.1. MULTIPLY/DIVIDE AND EDIT SUBROUTINES

The functions of multiplication, division, and editing are supplied optionally in the 9200 System. These options are offered as hardware instructions or as fixed, closed subroutines which duplicate the functions of the instructions. The hardware instructions are described in the UNIVAC 9200/9300 Hardware Reference Manual, UP-4139. The subroutines are described under the following headings.

2.1.1. Multiply/Divide Subroutine

The multiply and divide functions are provided by one fixed, closed subroutine which is in relocatable object code and which requires approximately 436 bytes of memory. The multiply/divide subroutine is to be linked to the problem program. To make the linking possible, the following source statement must be inserted in the problem program:

Operation	Operand
EXTRN	MPDP

This source statement provides information which allows the Linker to insert the address required for entering the subroutine. The MPDP symbol must not be defined by the problem program. The Assembler supplies special information in the object code of the program, which permits the subroutine to be entered directly by means of the following coding:

Multiply

Operation	Operand
BAL	15,MPDP
MP	op1, op2

Divide

Operation	Operand
BAL	15,MPDP
DP	op1, op2

The subroutine returns control to the program at the point immediately following the multiply or divide instruction to be executed.

Labels for the BAL, MP, or DP instructions are permitted, but they are not used by the subroutine. The operand addresses of the MP and DP instructions may be indexed or direct. The contents of all registers except register 15 are preserved by the subroutine. The condition code (CC) indicator is preserved and reset by the subroutine.

Since the multiply/divide subroutine essentially duplicates the functions of the hardware instructions, only differences in operation between the hardware instructions and the subroutine are described here.

- In the hardware instructions, the length of operand 1 is determined by detection of a sign which must be one of the following:

Plus	Minus
A(1010)	B(1011)
C(1100)	D(1101)
E(1110)	
F(1111)	

  In the subroutine, operand 1 length must be specified in the instruction. Sign position is thus determined by operand length, and any bit combination in this position is treated as the sign.

- In the hardware instructions, maximum operand 2 length is 16 bytes. In the subroutine, it is eight. If a larger length is specified, the subroutine reduces it by eight.

- In the hardware instructions, a divide check error stops the processor. In the subroutine, it causes a display of '29EE'.

- The operand identities for the multiplier and multiplicand are switched between the hardware instructions and the subroutine, but the product is stored in operand 1 in both cases.

General timing formulas for the multiply/divide subroutine operations are as follows (times are expressed in terms of memory cycles):

- Multiply

$$1950 + 19A + B + C + D + 775E + 388F$$

where:

A = the number of bytes in the multiplicand

B = 10 if the multiplicand is indexed; otherwise, B = 0

C = 78 if the product is negative; otherwise, C = 0

D = 5 if the multiplier is indexed; otherwise, D = 0

E = the number of digits in the multiplier

F = the sum of the multiplier digits

- Divide

$$3695 + 19A + B + C + D + 962E + 640F + 6G$$

where:

A = the number of bytes in the dividend

B = 10 if the dividend is indexed; otherwise, B = 0

C = 78 if the quotient is positive; otherwise, C = 0

D = 5 if the divisor is indexed; otherwise, D = 0

E = one less than the number of digits in the quotient

F = the sum of the quotient digits

G = the number of bytes in the divisor

2.1.2. Edit Subroutine

The edit subroutine is handled in the same way by the problem program as the multiply/divide subroutine. It is also a fixed, closed subroutine in relocatable object code, and it requires approximately 356 bytes of memory. To make the link between the problem program and the edit subroutine, the following source statement must be inserted into the problem program:

Operation	Operand
EXTRN	EDIT

This statement provides the Linker with information needed to insert the actual entrance address of the subroutine. The EDIT symbol must not be defined by the problem program. The Assembler provides special information in the object code of the user program to permit the first of the following statements to be used for entering the subroutine directly:

Operation	Operand
BAL	15,EDIT
ED	op1,op2

The subroutine returns control to the problem program at the point immediately following the edit instruction. All registers except register 15 are also preserved by the subroutine.

The operand addresses of the ED statement may be indexed or direct. Labels for the BAL or ED instructions are permitted but not used by the subroutine.

The edit subroutine achieves the same result as the hardware instruction.

2.1.3. Hardware Compatibility

If a subroutine user upgrades his computer to include the multiply, divide, and edit instructions, he can modify his programs as follows:

1. Remove all BAL 15,MPDP and BAL 15,EDIT instructions from his source code.
2. Reassemble the source code.
3. Do not link the multiply/divide and edit subroutines to the problem program.

2.2. STERLING CONVERSION ROUTINES

The sterling conversion routines convert sterling notation from card input to pence notation for computer operation and convert the results back to sterling notation for card punch or printer output.

The conversion routines assume that amounts in sterling notation are punched in the input cards in Univac standard card code and are translated into Univac standard EBCDIC code. In the input fields, leading zeros may be represented by blank columns.

The sterling conversion routines are entered by way of a BAL instruction. General register 15 is used as the return register. The routines use a common working storage area. The number of fractional pence decimal places is contained in a halfword constant following the BAL instruction. The number may be 0 through 3. The routines handle six different sterling notations (1A, 1B, 2A, 2B, 2C, 2D) and a pence notation. The formats for these notations as they appear in the common working storage area are shown in Table 2–1.

FORMAT	# DECIMALS	BYTE	0		1		2		3		4		5		6		7		8		9		10		11		12		13		14		15		16	
			← LEFT-HANDED EDGE OF COMMON WORKING STORAGE AREA																																	
1A	0	unpkd		0		0		0		£		£		£		£		£		£		£		£		£		s		s		d	x	d
1A	1	unpkd		0		0		£		£		£		£		£		£		£		£		£		s		s		d		d	x	f
1A	2	unpkd		0		£		£		£		£		£		£		£		£		£		s		s		d		d		f	x	f
1A	3	unpkd		£		£		£		£		£		£		£		£		£		s		s		d		d		f		f	x	f
1B	0	pkd	0	0	0	£	£	£	£	£	£	£	£	£	£	s	s	d	d	x														
1B	1	pkd	0	0	£	£	£	£	£	£	£	£	£	£	s	s	d	d	f	x														
1B	2	pkd	0	£	£	£	£	£	£	£	£	£	£	s	s	d	d	f	f	x														
1B	3	pkd	£	£	£	£	£	£	£	£	£	£	s	s	d	d	f	f	f	x														
2A, 2B	0	unpkd		0		0		0		£		£		£		£		£		£		£		£		£	x	£		s		s		d	
2A, 2B	1	unpkd		0		0		£		£		£		£		£		£		£		£		£		s		s		d	x	f		
2A, 2B	2	unpkd		0		£		£		£		£		£		£		£		£		£		s		s		d		f	x	f		
2A, 2B	3	unpkd		£		£		£		£		£		£		£		£		£		s		s		d		f		f	x	f		
2C, 2D	0	unpkd		0		0		0		£		£		£		£		£		£		£		£		£	x	£		s		d			
2C, 2D	1	unpkd		0		0		£		£		£		£		£		£		£		£		£		s		d	x	f				
2C, 2D	2	unpkd		0		£		£		£		£		£		£		£		£		£		s		d		f	x	f				
2C, 2D	3	unpkd		£		£		£		£		£		£		£		£		£		s		d		f		f	x	f				
pence	0	pkd	0	0	0	0	d	d	d	d	d	d	d	d	d	d	d	d	d	x														
pence	1	pkd	0	0	0	d	d	d	d	d	d	d	d	d	d	d	d	d	f	x														
pence	2	pkd	0	0	d	d	d	d	d	d	d	d	d	d	d	d	d	f	f	x														
pence	3	pkd	0	d	d	d	d	d	d	d	d	d	d	d	d	d	f	f	f	x														

£ POUND DIGIT
s SHILLING DIGIT
d PENNY INTEGER DIGIT
f PENNY DECIMAL FRACTIONAL DIGIT
x SIGN

Table 2–1. Sterling Notation Format

In sterling notations 2A and 2D, ten pence is represented by an 11-punch; eleven pence, by a 12-punch. In 2B and 2C, ten pence is represented by a 12-punch; eleven pence, by an 11-punch. In 2C and 2D, ten shillings is represented by a 12-punch, and eleven through nineteen shillings is represented by A-punches through I-punches, respectively.

A minus sign is represented by an 11-overpunch. On input to the routines, a plus sign is represented by no overpunch or a 12-overpunch; on output, a plus sign is represented by a 12-overpunch.

There are eleven conversion routines to convert one format to another.

CONVERSION ROUTINES	FORMAT	
	FROM	TO
R1A	2A	1B
R1B	2B	1B
R1C	2C	1B
R1D	2D	1B
R3	1B	Pence
R4	Pence	1B
R5A	1B	2A
R5B	1B	2B
R5C	1B	2C
R5D	1B	2D
R6	1B	1A

Conversion routine R6 also suppresses leading zeros in the tens digit of the shillings and pence fields.

2.3. MEMORY DUMP ROUTINE

The Memory Dump routine is used to obtain a printed record of the contents of a part or all of memory at some time during or after the execution of a program. The Memory Dump may be either a closed subroutine or a self-loading program.

As a closed subroutine, the Memory Dump is combined with the problem program at Assembler or Linker time and is executed when needed by means of a BAL instruction in the problem program.

As a self-loading program, the Memory Dump is loaded in the form of an object code deck from the card read unit. In this case, the Memory Dump begins to function automatically after the card deck is loaded.

To minimize the time required to print a memory dump, the routine uses a second output area in addition to the reserved printer output buffer area. In all other respects, the space required by the Memory Dump routine is minimized, so that in the case of the self-loading form, the amount of memory which must be altered to produce a memory dump is minimized.

The Memory Dump routine is completely self-contained. It incorporates its own printer subroutine, and to minimize the size of this subroutine, the Memory Dump routine is executed in I/O mode. When the Memory Dump is in the closed subroutine form, the assumption is made that the subroutine was entered in processor mode, and that mode is restored before control is returned to the problem program.

The Memory Dump routine is provided in the form of a macro. Whether the Preassembly Macro Pass is to produce source code in the closed subroutine form or in the self-loading form is determined by parameter specifications. Parameter specifications also determine the following:

(1) Whether the dump is to be made on a 96-, 120-, or 132-character printer.

(2) Whether the printer has a 63- or 48-character printer bar.

(3) At what memory location the dump begins.

(4) At what memory location the dump ends.

(5) For the self-loading routine –

   a. whether the card read unit is the card reader or the 1001 Card Controller; and

   b. where the Memory Dump routine is to be loaded.

After the source code deck is produced, it is still possible to change the parameters which specify the beginning and ending locations of the dump by replacing two cards in the source code deck. The beginning and ending dump locations may be further changed by appropriate Replace (REP) cards even after the source code deck is assembled. In the subroutine form, the REP cards may be used at Linker time. In either the subroutine or self-loading form, the REP cards may also be used as input to SQUEEZE, and that output is then placed at the end of the object code deck. In the case of the subroutine form, the problem program may externally reference the Memory Dump routine to change the beginning and ending dump locations.

The printout of a memory dump is as follows:

(1) Lines are doublespaced.

(2) The dump is printed in hexadecimal notation.

(3) The first line of print is the unmodified contents of the reserved printer output buffer area.

(4) Each subsequent line of the dump has the following format:

a. The address of the first byte printed on the line is at the extreme left followed by two spaces.

b. The contents of each byte are represented in two consecutive print positions: the zone of the byte is in the left position, and the digit of the byte is in the right position.

c. There are eight bytes to a group. Groups are separated by two spaces. Bytes are not separated within groups.

(5) Following the printer output buffer area printout, the next four lines are a dump of memory locations 0 through 127. Line 2 gives the contents of locations 0–31; line 3, locations 32–63; line 4, locations 64–95; and line 5, locations 96–127. These four lines have four groups to a line.

(6) The memory dump as specified begins at line 6.

(7) The number of groups per line of the memory dump depends on the number of print positions used in the printer. For the 96-character printer, five groups are printed per line; for the 120-character printer, six groups; for the 132-character printer, seven groups.

(8) The four least significant bits of the beginning address of the dump are erased, and the dump begins at the resulting address.

(9) During the memory dump, each halfword processed is compared to the immediately preceding halfword, with the exception of the first halfword. If each halfword in a line is equal to its predecessor, then instead of printing the line, the routine prints a line of asterisks. After a line of asterisks, nothing is printed until a line is detected in which some halfword is not equal to the immediately preceding halfword. The routine then returns to printing in normal output format. The last line of the memory dump is always printed.

2.4. SQUEEZE

SQUEEZE provides a simple method for modifying absolute program decks. Input consists of REP cards (as described for the Linker) and an END card (described below). The user specifies at Linker time whether the card input is from the card reader or the 1001 Card Controller. The translation table used for translating the control cards is similarly determined by the user when SQUEEZE is linked.

For each REP card read, SQUEEZE produces a Text card containing the location and text specified in the REP card. If possible, two or more REP cards are combined to produce one Text card. The process continues until the END card is read. A count is maintained of the number of Text cards produced. On the basis of the specifications given in the END card, SQUEEZE then produces a Transfer card and halts.

An END card contains the control card identifier END in columns 8–10. The specifications begin in column 14, are terminated by a blank, and have the following form:

a,i

where "a" is the address in hexadecimal, to which control is transferred after the updated element is loaded.

"i" is the increment, in hexadecimal, to be added to the cumulative count of the Text cards produced by SQUEEZE. The final count is punched in the Transfer card.

If the Text cards produced by SQUEEZE are placed just before the Transfer card in an absolute program deck, the text specified in the REP cards replaces the text in the program deck at the location specified in the REP cards. The replacement takes place when the absolute program deck is loaded.

The Transfer card produced by SQUEEZE may be used to replace the Transfer card in an absolute program deck. A NO in the "a" specification prevents SQUEEZE from producing a Transfer card.

SQUEEZE produces a listing of all REP and END cards processed. The user determines at the time SQUEEZE is linked whether the listing is to be produced on the 63- or 48-character print bar printer.

If the UNIVAC 9200/9300 System is restarted after a SQUEEZE operation is completed, the operation is reinitiated, and the system proceeds to process a new set of input cards.

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

UP-4120