SPERRY RAND

UNIVAC

9200/9200 II
9300/9300 II
SYSTEMS

CARD UTILITY PROGRAMS

PROGRAMMERS REFERENCE

UP-4120 Rev. 2

This manual is published by the Univac Division of Sperry Rand Corporation in loose leaf format. This format provides 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 product. For the current status of the product, 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 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 Systems.

UNIVAC is a registered trademark of Sperry Rand Corporation.

PRINTED IN U.S.A.

CONTENTS

1. INTRODUCTION

1.1. GENERAL

The purpose of this manual is to describe the card utility programs supplied as part of the software package provided with the UNIVAC 9200/9200 II/9300/9300 II Card Systems. Descriptions of the macro instructions used to initiate and generate these utility routines and programs as well as descriptions of the displays resulting from or associated with the routines and programs are also included.

A knowledge of the *UNIVAC 9200/9200 II/9300/9300 II Systems Central Processor and Peripherals Programmers Reference, UP-7546* (current version) and the *UNIVAC 9200/9200 II/9300/9300 II Systems Card Assembler Programmers Reference, UP-4092* (current version) is helpful in using this manual.

Section 1 of this manual defines the declarative and imperative macro instructions and their parameters as used in this manual. A description of the statement conventions used to illustrate the statements in this manual is also provided in this section.

The remainder of the manual is arranged as follows:

- Section 2 contains information pertaining to multiply/divide and edit subroutines.
- Section 3 contains information pertaining to sterling conversion routines.
- Section 4 contains information pertaining to the memory dump routine.
- Section 5 contains information pertaining to the snapshot dump routine.
- Section 6 contains information pertaining to the squeeze program.
- Section 7 contains information pertaining to the symbolic list program.

1.2. MACRO INSTRUCTIONS

A macro instruction is similar in format to a source code instruction. It may or may not contain an entry in the label field, but it must contain an operation code in the operation field and one or more parameters in the operand field. The macro instructions described in this document are classified as either declarative macro instructions or imperative macro instructions. The declarative and imperative macro instructions differ in three aspects: the purpose for which they are used, the format of the parameters specified in their operand fields, and the type of coding they generate.

1.2.1. Declarative Macro Instructions

Declarative macro instructions used keyword parameters to describe all the aspects of the file to be processed by the system. These aspects include parameters, constants, storage areas, special conditions, status, and options. Essentially, the declarative macro instruction defines each file required by the problem program. The code generated by the declarative macro instruction is nonexecutable and therefore should be separated from the inline file processing code.

The term "keyword parameter" refers to parameters which can be written in any order within the operand field. Keyword parameters must be separated by commas, but it is not required that the omission of a keyword be indicated. Keyword parameters are recognizable by their format, which consists of a word or code immediately followed by an equals sign, which is in turn followed by one specification.

1.2.2. Imperative Macro Instructions

Imperative macro instructions are used to point to the files described by the declarative macro instructions. In addition, imperative macro instructions are also used in providing additional details specifying the processing action to be taken. When executed, the imperative macro instruction generates many lines of inline, executable code.

The parameters contained in the operand field of the imperative macro instruction are positional parameters rather than the keyword parameters used in the declarative macro instruction. Positional parameters, as signified by their name, must be written in the order specified and separated by commas. When a positional parameter is omitted, the comma must be retained to indicate the omission except in the case of omitting trailing parameters.

1.3. STATEMENT CONVENTIONS

The conventions used to illustrate statements in this manual are as follows:

- Capital letters and punctuation marks (except braces, brackets, and ellipses) are information that must be coded exactly as shown.
- Lowercase letters and terms represent information that must be supplied by the programmer.
- Information contained within braces represents the necessary entries, one of which must be chosen.
- Information contained within brackets represents the optional entries that (depending on program requirements) are included or omitted. Braces within brackets signify that one of the entries must be chosen if that operand is included.
- An ellipsis indicates the presence of a variable number of entries.

Commas are required after each parameter, except after the last parameter specified. When a positional parameter is omitted from within a series of parameters, the comma must be retained to indicate the omission.

2. MULTIPLY/DIVIDE AND EDIT SUBROUTINES

2.1. GENERAL

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. These subroutines are supplied as relocatable card decks and must be linked to the problem program in which they are used. The subroutines are described in the paragraphs that follow.

2.2. 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. To utilize the multiply/divide subroutine, it is necessary to link this subroutine to the problem program in which it is used. This is accomplished by use of the EXTRN source statement which must be inserted into the problem program. The multiply/divide subroutine symbol (MPDP) specified in the operand of this source statement provides the information which allows the linker to insert the address required for entering the subroutine. The MPDP symbol, however, must not be defined by the problem program. The format of the EXTRN source statement used for linking the multiply/divide subroutine is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	EXTRN	MPDP

The multiply/divide subroutine is entered from the problem program by means of two instructions: the branch-and-link (BAL) instruction and the multiply-pack (MP) instruction for a multiplication function, or the BAL instruction and the divide-packed (DP) instruction for a division function. The BAL instruction performs two functions: it stores in processor register 15 the address used to return control to the problem program and it provides the mnemonic symbol MPDP for entering the multiply/divide subroutine. The specific operation to be performed (multiply or divide) and values to be used by the subroutine are specified by the appropriate MP and DP instructions. Examples of the coding format required to enter the multiply/divide subroutine into the program are as follows:

Multiply

LABEL	OPERATION	OPERAND
unused	BAL	15,MPDP
unused	MP	op1,op2

Divide

LABEL	OPERATION	OPERAND
unused	BAL	15,MPDP
unused	DP	op1,op2

Upon completing the execution of the multiply or divide subroutine, control is returned to that point of the problem program immediately following the multiply or divide instruction.

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.

2.2.1. Instruction Format

The multiply/divide subroutine essentially duplicates the functions of the system hardware instructions. (Both the hardware instructions and the subroutine instructions are of the storage-to-storage (SS2) format type.) Differences, however, exist in the manner in which operational conditions are specified in these two instruction versions. Therefore, only the differences between the hardware instructions and the subroutine instructions are described herein.

- In the hardware instructions, the length of operand 1 is determined by detection of a sign byte. The sign byte for the decimal numbers contained in operand 1 is represented by hexadecimal digits A through F, where A, C, E, or F represents a plus sign and B or D represents a minus sign.

  In the subroutine instructions, the length of operand 1 must be specified within the instruction. The sign of the operand is therefore defined by its position in an operand of a given length. Any bit configuration in this position is interpreted as the sign of the operand.

- In the hardware instructions, the maximum length of operand 2 is 16 bytes. In the subroutine instructions, the length is limited to eight bytes. If a length larger than eight bytes is specified, the subroutine reduces the specified length to eight bytes.

- In the hardware instructions, a divide check stops processor operations. In the subroutine instructions, it causes a display of '29EE' to be presented on the control console.

- The operand identities for the multiplier and the multiplicand of the subroutine multiply instruction are the reverse of the operand identities specified for the hardware instruction. That is, in the subroutine instruction, the multiplier is specified in operand 2 and the multiplicand is specified in operand 1. The resulting product, however, is stored in the location specified by operand 1 for both versions of the instruction.

2.2.2. Timing Formulas for Multiply/Divide Subroutine

Timing for multiply/divide subroutine operations can be calculated by the use of two general timing formulas. These two formulas (one for multiply operations and one for divide operations) provide times that are expressed in terms of memory cycles. The timing formulas are as follows:

- Multiply Operation

$$1950 + 19a + b + c + d + 775e + 388f$$

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 Operation

$$3695 + 19a + b + c + d + 962e + 640f + 6g$$

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

2.3. 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 EXTRN source statement must be inserted into the problem program:

LABEL	OPERATION	OPERAND
unused	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.

To enter the edit subroutine, it is necessary to execute a BAL instruction. Execution of this instruction performs two functions: It stores into processor register 15 the address used to return control to the problem program, and it identifies the edit subroutine by specifying the EDIT symbol. The format of the BAL instruction for entering the subroutine is as follows:

LABEL	OPERATION	OPERAND
unused	BAL	15,EDIT

The parameters to which the edit subroutine must function are provided by the operands of the edit (ED) instruction which follows the BAL instruction in the problem program. The format of the ED instruction is as follows:

LABEL	OPERATION	OPERAND
unused	ED	op1,op2

Execution of the edit instruction transfers data from the bytes specified by operand 2 (op2) to the bytes specified by operand 1 (op1). The data format of op2 is changed from packed decimal to unpacked decimal, zone bits and editing symbols are inserted, and leading zeros are suppressed as part of the editing process which takes place during the data transfer. This editing process is controlled by the editing mask in op1 which is overlaid by execution of the instruction. Operand 2 must be a packed decimal field, and op1 must contain the editing mask to avoid unpredictable results from the execution of this instruction.

Upon completion of the subroutine, control is returned to the problem program at the point immediately following the edit (ED) instruction. All registers except register 15 are 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.

2.3.1. Timing Formula for Edit Subroutine

The formula for calculating an approximate timing for the edit routine is as follows: (Times are expressed in memory cycles.)

$$766 + a + b + c + 174d + 14e + 86f + 172g + 254h + 326i + 390j + 472k + 3701 + 316m + 426n$$

where:

a = 202 if operand 2 is indexed; otherwise, a = 0

b = 202 if operand 1 is indexed; otherwise, b = 0

c = 52 if the data value is nonzero; otherwise, c = 0

d = the number of data digits

e = the integral part of d/2

f = the number of ANSCII minus signs in the data

g = the number of EBCDIC minus signs in the data

h = the number of plus signs in the data

i = the number of digit select bytes for which suppression is performed

j = the number of digit select bytes for which digits are selected

k = the number of significant start bytes

l = the number of edit pattern bytes that are suppressed

m = the number of edit pattern bytes that are not suppressed

n = the number of field separator bytes

2.4. 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.

3. STERLING CONVERSION ROUTINES

3.1. GENERAL

Sterling notation is the expression of monetary fields in terms of pounds, shillings, and pence. One pound is equal to 20 shillings; one shilling equals 12 pence.

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.

Sterling notation is punched in the input cards in UNIVAC 9200/9200 II/9300/9300 II Systems standard card code and is translated into EBCDIC internal code (Appendix A). The output of the routines is also in EBCDIC code. In the input fields, leading zeros may be represented by blank columns.

The sterling conversion routines use a common working storage area. 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 3–1.

LEFT-HAND EDGE OF COMMON WORKING STORAGE AREA

FORMAT	# DECIMALS	BYTE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
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 3–1. Sterling Notation Format

3.1.1. Format 1A

This format is used if the output is to be printed. Format 1A represents sterling amounts in the standard notation of pounds, shillings, and pence, including decimal fractions of a penny.

Format 1A permits a maximum of 10 positions for pounds, 2 positions for shillings, 2 for pence, and 3 for decimals. Table 3–2 illustrates format 1A maximum sizes.

FORMAT	POUNDS (£)	s	d	DECIMALS (f)	USE
1A	££££££££££	ss	dd	fff	PRINTING (OUTPUT)
2A, 2B	££££££££££	ss	d	fff	READING/PUNCHING (INPUT/OUTPUT)
2C, 2D	££££££££££	s	d	fff	READING/PUNCHING (INPUT/OUTPUT)
Pence	d d d d d d d d d d d d			fff	COMPUTATION (INTERNAL)

£ pound digit
s shilling digit
d penny digit
f penny fractional digit

Table 3–2. Maximum Sizes of Sterling Notation Formats

Table 3–3 shows examples of sterling amounts written in Format 1A.

	STERLING AMOUNT											
	UNSIGNED				NEGATIVE				POSITIVE			
FORMAT	POUNDS	17 SHILLINGS	11 PENCE	.155 DECIMAL	123 POUNDS	8 SHILLINGS	10 PENCE	.123 DECIMAL	456 POUNDS	10 SHILLINGS	7 PENCE	DECIMAL
1A	0	19	11	155	123	08	10	12L	456	10	0G	
2A	0	19	+	155	123	08	–	12L	45F	10	7	
2B	0	19	–	155	123	08	+	12L	45F	10	7	
2C	0	I	–	155	123	8	+	12L	45F	+	7	
2D	0	I	+	155	123	8	–	12L	45F	+	7	
Pence	239			155	29626			12L	10956G			

\+ 12 punch
– 11 punch

Table 3–3. Sterling Amounts Expressed in Available Formats of Notation

3.1.2. Format 1B

Format 1B is the same as format 1A except that the information is packed rather than unpacked.

3.1.3. Format 2

Format 2 is used for both input and output. Input cards are read and output cards punched in this format.

Format 2 permits a maximum of 10 positions for pounds and 3 positions for decimals. Shillings and pence may be represented in either the British Standards Institution (BSI) or the Hollerith code.

3.1.3.1. Shillings

- BSI Code

  A single column is used to represent the shillings field. Amounts of 0 through 9 are indicated by the punches 0 through 9. Ten shillings are represented by a 12 punch in the column. Eleven through 19 shillings are represented by the A through I punches, respectively.

- Hollerith Code

  Two columns are used to represent the shillings field. Decimal notation is used, the first column representing the tens position (that is, it contains either 0 or 1) and the second representing the units (digits 0 through 9).

3.1.3.2. Pence

- BSI Code

  The pence field is represented by a single column. Amounts of 0 through 9 are indicated by the punches 0 through 9. Ten pence are represented by a 12 punch; 11 pence by an 11 punch.

- Hollerith Code

  The Hollerith code is similar to the BSI code in that both use a single column to represent pence. Amounts of 0 through 9 are indicated by the punches 0 through 9. However, 10 pence are represented by an 11 punch; 11 pence by a 12 punch.

3.1.3.3. Format 2 Types

Format 2 may be used in any of the following combinations of BSI and Hollerith code.

TYPE OF FORMAT	SHILLINGS	PENCE
Format 2A	Hollerith code	Hollerith code
Format 2B	Hollerith code	BSI code
Format 2C	BSI code	BSI code
Format 2D	BSI code	Hollerith code

Table 3–2 illustrates the maximum sizes of format 2. Refer to Table 3–3 for examples of sterling amounts written in format 2.

3.1.3.4. Signs

The sign is found in the least significant digit of the decimal portion of a sterling field. However, if no decimal fractions of a penny exist in the field, the zone punch identifying the sign is placed in the units position of the pounds field.

The sign codes are as follows:

– (minus) = 11 punch

+ (plus) = 12 punch or blank

NOTE: An 11 punch always requires a digit underpunched in the same column. A 12 punch in the appropriate position is always used to signify positive output amounts.

3.1.3.5. Leading Zeros

Leading zeros can be represented by blanks in sterling input fields.

3.1.4. Pence Format

The pence format is a "pence only" notation of a sterling amount; all pound and shilling fields are converted to pence.

Pence format allows for a maximum of 16 positions: 13 positions for pence and 3 decimal positions.

Table 3–2 illustrates the maximum size of the pence format. Refer to Table 3–3 for an example of sterling amounts in pence format.

The pence notation is operated on with decimal arithmetic instructions.

A summary of sterling notation formats is given in Table 3–4.

	PRINTING FORMAT	CARD READING/PUNCHING FORMAT	
FIELDS	FORMAT 1A	FORMATS 2A, 2B	FORMATS 2C, 2D
Pounds	1–10 positions	1–10 positions	1–10 positions
Shillings	2 positions (contents not to exceed 19)	2 positions (contents not to exceed 19)	1 position
		Hollerith code	BSI code
Pence	2 positions (contents not to exceed 11)	1 position BSI or Hollerith code	1 position BSI or Hollerith code
Decimals	0–3 positions	0–3 positions	0–3 positions

Table 3–4. Summary of Sterling Notation Formats

3.2. CONVERSION ROUTINES

There are 11 routines to convert one format to another. Table 3–5 lists the conversion routines and shows the relationship of the routines to the individual formats.

CONVERSION ROUTINES	CONVERSION		APPROX. NUMBER OF BYTES	APPROX. MAXIMUM TIME FOR EXECUTION*	FORMAT	
	FROM	TO			INPUT	OUTPUT
R1A	2A	1B	94	1.2	16 bytes unpacked	9 bytes packed
R1B	2B	1B	94	1.2	16 bytes unpacked	9 bytes packed
R1C	2C	1B	116	1.4	15 bytes unpacked	9 bytes packed
R1D	2D	1B	116	1.4	15 bytes unpacked	9 bytes packed
R3	1B	Pence	126	5.0	9 bytes packed	9 bytes packed
R4	Pence	1B	118	5.8	9 bytes packed	9 bytes packed
R5A	1B	2A	78	1.0	9 bytes packed	16 bytes unpacked
R5B	1B	2B	78	1.0	9 bytes packed	16 bytes unpacked
R5C	1B	2C	110	1.3	9 bytes packed	15 bytes unpacked
R5D	1B	2D	110	1.3	9 bytes packed	15 bytes unpacked
R6**	1B	1A	56	1.0	9 bytes packed	17 bytes unpacked

*In milliseconds

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

Table 3–5. Conversion Routines

3.3. USING CONVERSION ROUTINES

Any one of the routines may be used individually or in conjunction with others to meet any user's requirement.

3.3.1. Common Work Area

The problem program delivers data to be processed by a sterling conversion routine by storing the data in a common work area. The sterling conversion routine also uses the common work area to deliver the converted data to the problem program. This common work area must be supplied by the problem program, must be 24 bytes long, and must be labeled SWKA.

3.3.2. Converting Data

When the problem program requires conversion of data, it places the data in SWKA and then enters the sterling conversion routine by means of a BAL instruction that uses register 15 and that branches to the name of the routine. The BAL instruction should be immediately followed by a half-word constant containing, right justified in binary notation, the number of fractional pence decimal places in the data to be converted. The number may be 0 through 3. When the sterling conversion routine has finished the conversion, it stores the converted data in SWKA and returns control to the instruction immediately following the constant specifying the number of decimal places.

Register 14 is used by the routines as a working register. The sterling routines do not save the contents of register 14. Therefore, the problem program should not have usable information in register 14 when linking to the sterling routines.

Figure 3–1 illustrates a typical use of the routine R1A, converting format 2A to format 1B. The letter x in the DC directive must be replaced by one of the digits 0, 1, 2, or 3 to indicate the number of decimal positions required.

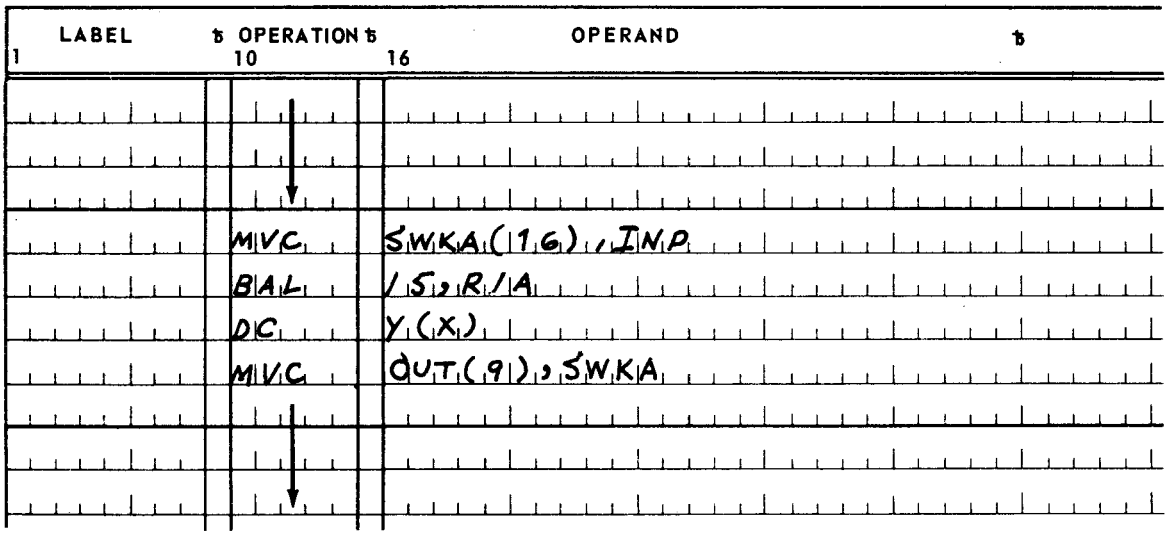


Figure 3–1. Linking Sequence, Format 2A to Format 1B

3.4. INCLUSION OF ROUTINES INTO THE PROBLEM PROGRAM

The user may incorporate sterling conversion routines into his problem program in either of two ways. He may include the sterling routines in his source program for joint assembly, or he may assemble the sterling routines separately and link them to the problem program.

The sterling conversion routine R3 requires multiplication. The sterling conversion routine R4 requires division. These two routines are written on the assumption that they are to be used with a configuration that does not include the multiply and divide instructions. Therefore, the following is required:

a. If the assumption is true, then whenever routine R3 or R4 is included in a program, the multiply/divide subroutine named MPDP must also be included.

b. If the assumption is false, then the BAL instructions with the following format should be removed from routines R3 and R4 before they are assembled. There are two such instructions in each of these routines.

LABEL	♭ OPERATION ♭	OPERAND
unused	BAL	15,MPDP

3.4.1. Joint Assembly

If the user desires to assemble the sterling routine(s) together with his source program, he incorporates the work area definition (SWKA) and the sterling routines in the source program. The sterling routine object program becomes an integral part of the object program.

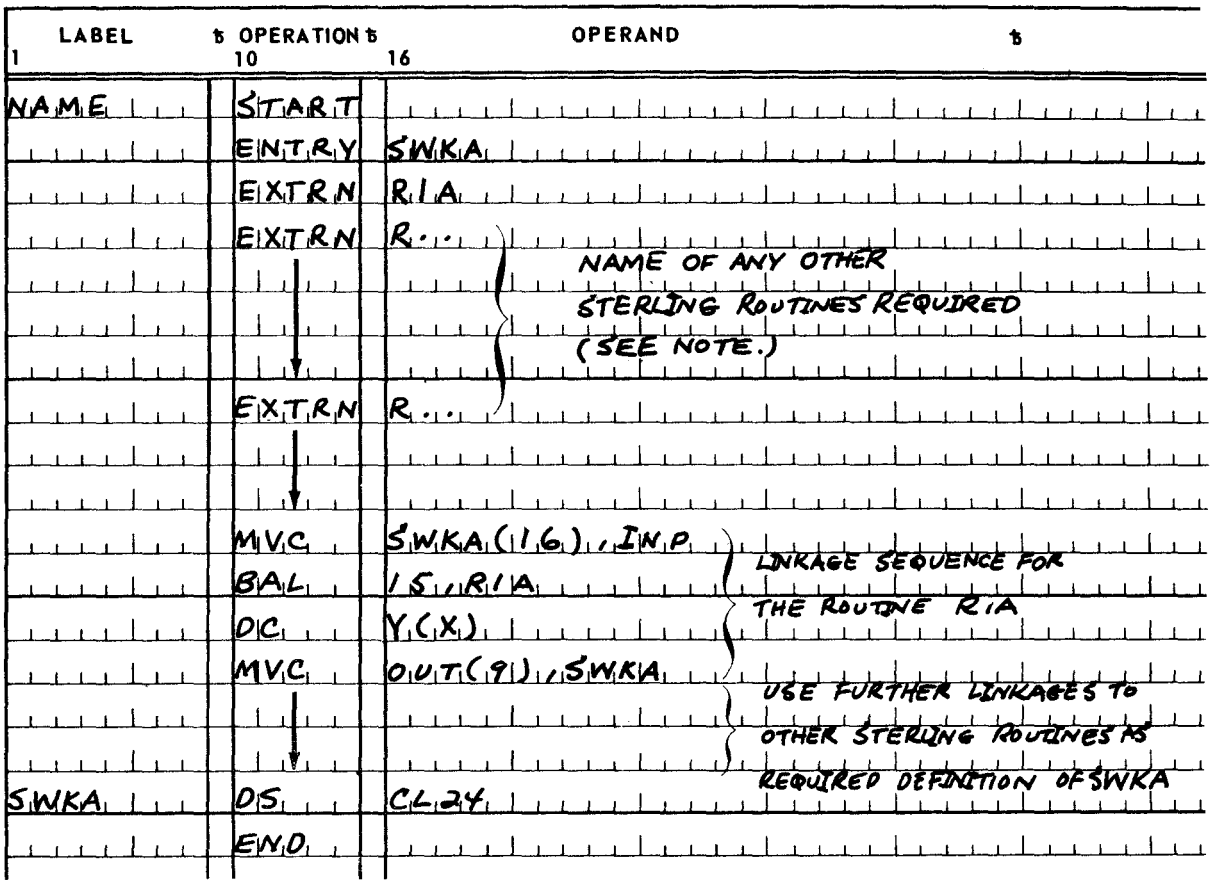
3.4.2. Separate Assembly

For separate assembly of the sterling routine package, which is composed of the desired sterling routines and an END statement, the user must include the following in the problem program:

- The name of the common work area (SWKA) must be identified by means of an ENTRY statement.
- The names of all the sterling routines used must be identified by means of EXTRN statements.
- SWKA must be defined.

Figure 3–2 illustrates the coding for the main program for separate assembly of sterling routines.

The letter x in the operand field of the DC statement in the calling sequence shown in Figure 3–2 must be replaced by one of the digits 0, 1, 2, or 3 to indicate the number of decimal positions required.



NOTE: Those sterling routines denoted by EXTRN statements must also be assembled separately and then linked with the problem program.

Figure 3–2. Coding the Problem Program for Separate Assembly of the Sterling Routines

4. MEMORY DUMP ROUTINE

4.1. GENERAL

The memory dump routine is a means of printing the entire contents of main storage or a specified portion of it. The routine is provided in compressed code format and, by use of appropriately prepared macro instructions in the preassembly macro pass, can be generated in the form of a closed subroutine or in the form of a self-loading card deck.

In the closed subroutine form, the source code generated by the preassembly macro pass may be incorporated into the problem program at assembly time or may be assembled separately from the program and incorporated into the program at linker time. The routine may then be executed by a BAL instruction in the problem program.

In the self-loading form, the source code generated by the preassembly macro pass should be assembled as an independent program. The object code of the self-loading memory dump routine can then be loaded from the online card reader or the 1001 card controller when a program ends or aborts. The routine, if loaded into an area which will not be cleared or overlayed by a subsequent program, can remain resident and can be accessed manually from the control console as often as required.

The beginning and ending locations of the main storage area to be dumped are defined by two type Y define-constant (DC) statements labeled MBGN and MEND. The following is an example of the format of these two statements.

LABEL	♭ OPERATION ♭	OPERAND
MBGN	DC	Y(c)
MEND	DC	Y(c)

where:

c, the constant subfield, represents either an absolute or a relocatable main storage address

The dump locations specified in these DC statements may be changed by replacing the appropriate cards contained in the source code deck generated by the preassembly macro pass. This method applies to the closed subroutine and the self-loading forms of the memory dump routine. If the source deck has been assembled, then the beginning and ending dump locations are changed by submitting appropriately prepared replace (REP) cards to the squeeze routine. (Refer to Section 6 for a detailed discussion of the purpose and function of the squeeze routine.) The REP cards can also be used at linker time if the subroutine form of the memory dump is being incorporated into the problem program by the linker routine.

Finally, in the case of the subroutine form, the problem program can externally reference the memory dump routine in order to change the beginning and ending dump locations at execution time.

In addition to the data contained between the beginning and ending memory dump locations specified by the routine, the memory dump routine always prints the contents of the fixed hardware output area and the control area (address 0 through 127).

4.2. CONTENTS OF CONTROL AREA

The portions of the control area used by the memory dump routine during its execution are the input/output program-state-control (I/O PSC) word, I/O registers 13 through 15, and the line advance control byte (location 80). These areas are altered by the memory dump routine. However, the subroutine form of the memory dump routine preserves the initial contents of these areas before beginning its dump execution proper. These preserved contents can be found at the following locations:

- MD+8 through MD+11 for the I/O PSC
- MD+12 through MD+17 for I/O registers 13 through 15
- MD+18 for the line advance control byte
- MD+24 for the processor PSC

The beginning and ending locations of the dump can be found at the following locations:

- MD+2 through MD+3 for the MBGN constant
- MD+4 through MD+5 for the MEND constant

MD is the first location of the routine loaded into main storage. In the self-loading form of the routine, the contents of the part of control area stored in addresses 0 through 79 are destroyed by the loader when the routine is loaded.

4.3. PRINT FORMAT

The print format of the memory dump is as follows:

- Lines are double-spaced.
- Contents of the fixed-hardware-printer output area are dumped first and printed on print line 1 without format modification.
- Contents of the control area (locations 0 through 127) are dumped next and are printed on print lines 2 through 5 in the format described in this paragraph.
- Contents of the memory dump proper are dumped last. This portion of the dump begins on print line 6 in the format described in this paragraph.
- The print format for the memory dump, with the exception of print line 1, which contains the fixed-hardware-printer output area, is as follows:
  - The address of the first byte printed on each line is presented in hexadecimal notation, positioned at the left margin of the line, and separated from the rest of the line by two spaces.
  - The contents of each byte are represented in two consecutive print positions. The left position represents the zone of the byte and the right position represents the digit of the byte. Both positions are expressed in hexadecimal notation.
  - Bytes are formed into eight-byte groups (16 printed characters). Groups are separated by two spaces. However, bytes within each group are not separated.

- Four eight-byte groups are printed on each of the four print lines (2 through 5) containing the contents of the control area.
- The number of eight-byte groups printed for each line of the memory dump proper depends upon the printer used. That is, for a printer with 96 print positions, five eight-byte groups are printed per line; a printer with 120 print positions prints six eight-byte groups per line; and a printer with 132 print positions prints seven eight-byte groups per line.
- Under certain conditions, an entire print line of asterisks may appear in the printing of the memory dump proper. The printing of these asterisks indicates that each half-word contained in the line is equal to the last half-word of the previous line. After a line of asterisks has been printed, no more output is produced until a line in which a half-word not equal to the last half-word of the previous line is detected. With this line, the routine returns to producing lines of print in the normal output format. The last line of the memory dump proper is always printed.

■ The paper is automatically skipped to the home paper position after the memory dump has been completed.

4.4. MAIN STORAGE REQUIREMENTS

The main storage requirements of the memory dump routine are based upon the form of the routine. If the routine is in the self-loading form, the storage requirement is fixed at 590 bytes. If the routine is in the closed subroutine form, the storage requirements are determined by the number of print positions specified in the print position keyword parameter of the memory-dump-subroutine (MDSBR) and the memory-dump-self-loading-form (MDSLF) macro instructions. The number of bytes and their associated POS parameter values are as follows:

622 bytes when MDSBR POS=96

646 bytes when MDSBR POS=120

658 bytes when MDSBR POS=132

4.5. CHARACTERISTICS OF CLOSED SUBROUTINE FORM OF MEMORY DUMP ROUTINE

A description of the macro instructions used for the closed subroutine form of the memory dump, as well as a sample of the problem program coding used to produce a memory dump, is provided in the paragraphs that follow.

4.5.1. Memory-Dump-Subroutine (MDSBR) Declarative Macro Instruction

The MDSBR declarative macro instruction consists of four keyword parameters which define the number of print positions per line, the character set of the print bar, and the beginning and ending locations of the dump. The format of the MDSBR macro instruction is as follows:

LABEL	OPERATION	OPERAND
unused	MDSBR	POS = {96 / 120 / 132},
		CH = {48 / 63}
		[,BGN=nnnnn]
		[,END=nnnnn]

- Print Position (POS) Keyword Parameter

The POS keyword parameter defines the maximum number of valid print positions to appear in a line of print for a specific routine generated. If the maximum number of print positions of the printer being used exceeds that specified by the POS parameter, the remaining print positions of the line should be disregarded. The formats of the POS parameter are as follows:

- POS=96

  This format of the keyword parameter limits the number of print positions per line to 96.

- POS=120

  This format of the keyword parameter limits the number of print positions per line to 120.

- POS=132

  This format of the keyword parameter limits the number of print positions per line to 132.

- Character Set (CH) Keyword Parameter

The CH keyword parameter is used to specify the character set of the print bar to be used. Since two character sets are available, the CH keyword parameter can be specified in the one to be used. The CH keyword has the following formats:

– CH=48

This format of the CH keyword parameter specifies a print bar with 48 valid characters.

– CH=63

This format of the keyword parameter specifies a print bar with 63 valid characters.

■ Dump Beginning Address (BGN) Keyword Parameter

The BGN keyword parameter specifies the main storage location at which the dump is to begin. The use of the BGN is optional. If the BGN parameter is omitted, a main storage location of 256 is assumed by the routine, and the memory dump will start at that location. The format of the BGN parameter is as follows:

BGN=nnnnn

The value for nnnnn can be any number from 256 through 32767. It should be noted that the memory dump routine erases the four least significant bits of the beginning address when it starts to produce the memory dump. Therefore, the address which is nearest to but less than the specified address and which is an integral multiple of 16 is adopted by the routine as the actual beginning address for the memory dump.

■ Dump Ending Address (END) Keyword Parameter

The location at which the memory dump is to end is specified by the END keyword parameter. The use of the END parameter is also optional. However, the value of 8191 is adopted by the routine as the ending address if the END parameter is omitted. The format of the END keyword parameter is as follows:

END=nnnnn

The value for nnnnn can be any number from 256 through 32767. The actual ending location of the memory dump may exceed that specified by the END parameter. This is due to the fact that the routine always produces the memory dump in groups of eight bytes including the entire eight bytes in which the specified ending location is contained. (Refer to 4.3 for information concerning print format for the memory dump.) If the calculation of the address of the last eight bytes to be dumped produces an address that is beyond the memory limits of the machine, the memory dump terminates in a processor abnormal condition.

4.5.2. END Imperative Macro Instruction

The END imperative macro instruction must follow the MDSBR macro instruction, which is described in 4.5.1. The function of the END macro is to define the completion or end of the closed subroutine having the name specified by the symbol contained in the operand of this instruction. The format for the END macro instruction used with the closed-subroutine form of the memory dump is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	END	MENT

4.5.3. Instructions Required for Execution of Closed Subroutine

The closed subroutine for memory dump is executed when the properly coded BAL instruction is encountered in the problem program. The entry point to the closed subroutine is defined by the MENT symbol contained in the operand field of the BAL instruction. Also contained in the operand field of the BAL instruction is the parameter that specifies processor register 15 as the return register. The branch from the problem program to the closed subroutine can be made by use of the following coding:

LABEL	♭ OPERATION ♭	OPERAND
unused	BAL	15,MENT

The memory dump routine is designed to be entered in processor mode, and it returns to processor mode before returning control to the problem program. The memory dump itself is executed in I/O mode. Three symbols, MBGN, MEND, and MENT, are declared by ENTRY directives in the closed-subroutine form of the memory dump routine. Thus, the memory dump routine and problem program can each be assembled separately and still be linked together by EXTRN directives, which are declared in the problem program.

When the memory dump is in the form of a closed subroutine, it may be necessary to change the beginning and ending locations of the dump in the problem program at running time. Accordingly, the memory dump routine has the following symbols for the constant areas that specify beginning and ending locations:

- MBGN for BGN address constant area. The length is a half-word.
- MEND for END address constant area. The length is a half-word.

A sample of the problem program coding which produces a memory dump from locations 1000 through 2000 is as follows:

```
LABEL      OPERATION      OPERAND
1          10             16
           MVC    MBGN(2),CON1
           MVC    MEND(2),CON2
           BAL    15,MENT
           .
           .
           .
CON1       DC     Y(1000)
CON2       DC     Y(2000)
```

Since MBGN and MEND are placed in adjacent main storage locations, the preceding coding can also be written as follows:

```
           MVC    MBGN(4),CON1
           BAL    15,MENT
           .
           .
           .
CON1       DC     Y(1000)
           DC     Y(2000)
```

4.6. CHARACTERISTICS OF SELF-LOADING FORM OF MEMORY DUMP ROUTINE

The following paragraphs contain descriptions of the macro instructions, programming considerations, and operating instructions for the self-loading form of the memory dump routine.

4.6.1. Memory-Dump-Self-Loading-Form (MDSLF) Declarative Macro Instruction

The MDSLF declarative macro instruction contains eight keyword parameters. These parameters define the line length and character set of the memory dump printout, as well as the location at which the routine is to be loaded, and which card reader unit is to be used. The beginning and ending locations of the memory dump are also provided as keyword parameters of this macro instruction. The format of the MDSLF macro instruction is as follows:

LABEL	♭ OPERATION ♭	OPERAND
[name]	MDSLF	POS = {96 / 120 / 132},
		CH = {48 / 63}
		{ [{,BGN=nnnnn,END=nnnnn / ,BGN=nnnnn / ,END=nnnnn}] ,LOAD=nnnnn
		[,MEN = {8K / 12K / 16K / 24K / 32K}] }
		[,RDR=1001,CHAN= {7 / 8 / 9 / 10 / 11 / 12}]

- Print Position (POS) Keyword Parameter

  The POS keyword parameter defines the maximum number of valid print positions to appear in a line of print for a specific routine generated. If the maximum number of print positions of the printer being used exceeds that specified by the POS parameter, then the remaining print positions of the line are filled with hash and are to be disregarded. The formats of the POS parameter are as follows:

  – POS=96

    This format of the keyword paramter limits the number of print positions per line to 96.

– POS=120

This format of the keyword parameter limits the number of print positions per line to 120.

– POS=132

This format of the keyword parameter limits the number of print positions per line to 132.

■ Character Set (CH) Keyword Parameter

The CH keyword parameter is used to specify the character set of the print bar to be used. Since two character sets are available, the CH keyword parameter is used to specify the one to be used. The formats for the CH keyword parameter are as follows:

– CH=48

This format specifies a print bar with 48 valid characters.

– CH=63

This format specifies a print bar with 63 valid characters.

■ Dump Beginning Address (BGN) Keyword Parameter

The BGN keyword parameter is an optional parameter used to specify the main storage location at which the memory dump is to begin. The value specified in this parameter may be any number from 256 through 32767. It should be noted, however, that this parameter must be omitted if the MEM parameter is used in this instruction. If both the BGN parameter and the MEM parameter are omitted from the instruction, the memory address 256 is adopted as the beginning address of the memory dump. The format for the BGN keyword parameter is as follows:

BGN=nnnnn

The value of nnnnn can be any number from 256 through 32767. The actual beginning location, however, must be an address which is an integral multiple of 16. Therefore, if the address specified by the BGN parameter is not an integral multiple of 16, the dump starts at the next lower address that is a multiple of 16.

■ Dump Ending Address (END) Keyword Parameter

The END parameter is also an optional parameter. It is used to specify the main storage location at which the memory dump is to end. The value specified in this parameter may be any number from 256 through 32767. The END parameter must be omitted if the MEM parameter is used in the instruction. If both the END parameter and the MEM parameter are omitted from the instruction, a memory address of 8191 is adopted as the ending address of the memory dump. The format for the END keyword parameter is as follows:

END=nnnnn

The value of nnnnn can be any number from 256 through 32767. The actual ending location of the dump must be an address which is an integral multiple of 8. Therefore, if the address specified by the END parameter is not an integral multiple of 8, the dump ends at the next higher address that is a multiple of 8.

- LOAD Keyword Parameter

The LOAD keyword parameter specifies the beginning main storage location into which the memory dump routine is to be loaded. The value specified in this parameter may be any number from 260 through 32100. The LOAD parameter must be omitted if the MEM parameter is used in this instruction. The format of the LOAD instruction is as follows:

LOAD=nnnnn

The value of nnnnn can be any number from 260 through 32100.

- Memory (MEM) Keyword Parameter

The MEM keyword parameter is used to specify the main storage size for a routine that is to dump the entire contents of main storage. It also indicates that the memory dump routine is to be loaded into the highest locations of main storage. If the BGN, END, or LOAD parameters are used in the macro instruction, then the MEM parameter must be omitted. The formats of the MEM parameter used for the various sizes of main storage are as follows:

– MEM=8K

This format of the MEM parameter is used when the entire contents of an 8K main storage are to be dumped.

– MEM=12K

This format of the MEM parameter is used when the entire contents of a 12K main storage are to be dumped.

– MEM=16K

This format of the MEM parameter is used when the entire contents of a 16K main storage are to be dumped.

– MEM=24K

This format of the MEM parameter is used when the entire contents of a 24K main storage are to be dumped.

– MEM=32K

This format of the MEM parameter is used when the entire contents of a 32K main storage are to be dumped.

- Reader (RDR) Keyword Parameter

  The RDR keyword parameter is an optional parameter that is used only when the UNIVAC 1001 Card Controller is used as the routine load device as opposed to the online card reader. If the routine is to be loaded from the online card reader, then the RDR parameter and its associated CHAN parameter must be omitted from the instruction. The format for the RDR parameter is as follows:

  RDR=1001

- Channel (CHAN) Keyword Parameter

  The CHAN keyword parameter is an optional parameter which is used to specify the number of the channel to which the card controller is attached. This parameter, therefore, is only used when the RDR parameter is included in the instruction format. If the RDR is omitted, then the CHAN keyword parameter must also be omitted. The formats for the CHAN keyword parameter are as follows:

  - CHAN=7

    This format specifies that the 1001 card controller is connected to channel number 7.

  - CHAN=8

    This format specifies that the 1001 card controller is connected to channel number 8.

  - CHAN=9

    This format specifies that the 1001 card controller is connected to channel number 9.

  - CHAN=10

    This format specifies that the 1001 card controller is connected to channel number 10.

  A program name may be defined for the memory dump by entering a valid name in the label field of the MDSLF macro instruction. The use of the label field is optional. However, if its use is desired, it may contain any valid symbol desired, except that the characters Y or Q may not be used as the second character in the symbol. If the symbol is omitted, a label defined as MD is assigned as the program name by the macro library when the routine is generated.

4.6.2. END Imperative Macro Instruction

The END imperative macro instruction must follow the MDSLF instruction in the card deck. The format of the END macro instruction for the self-loading memory dump routine is the same as that which follows the MDSBR instruction used for the closed-subroutine form. The END macro instruction identifies the end of the routine and specifies the name of the routine. The format of the END macro instruction is as follows:

LABEL	b OPERATION b	OPERAND
unused	END	MENT

4.6.3. Programming Considerations

After the source code has been generated by the preassembly macro pass, it is assembled to produce a self-loading object deck. Before loading the object deck, the A and J cards (first and second cards of the object deck) must be removed so that the first card to be loaded contains a Q in column 2.

If a problem program is expected to abort or to require a memory dump subsequent to running, it is preferable to load the memory dump routine prior to the execution of the problem program. The routine is therefore resident in memory during the execution of the problem program and can be accessed by manual operation at the control console after the problem program aborts. When using this method, make certain that the problem program does not overlay the memory routine or clear the storage area in which the routine resides. As a precautionary measure the following steps should be taken:

a. Assemble a dump routine which is loaded into the high order areas of main storage.

b. Set the limits of the problem program so that the highest location used corresponds to the following:

Location (Hexadecimal)	Main Storage Size
1DB2	8K
2DB2	12K
3DB2	16K
5DB2	24K
7DB2	32K

c. When linking the problem program to a loader, set the L?HI-labeled EQU directive to the appropriate hexadecimal limit specified in step b. This defines the last location of main storage to be cleared.

d. When linking the problem program to a loader, set the L?AR-labeled EQU directive to the appropriate hexadecimal limit specified in step b minus 340 if the online reader is used as the loading device or to minus 444 if the card controller is used as the loading device. The resulting value for L?AR specifies the beginning of the read area for the load routine.

NOTE: If the problem program is written in RPG, the requirements specified in steps b, c, and d are satisfied by placing the character D in column 80 of the RPG header control card.

4.6.4. Operating Instructions

A programmer may determine the status of the processor by performing the operations specified in the following paragraphs. This must be accomplished prior to the performance of the memory dump since the status conditions stored are destroyed once the memory dump is initiated. The operations required to obtain the status of the processor are initiated from the operator's panel of the control console.

NOTE: Make certain that the CLEAR switch is not set prior to performing this procedure.

a. Check for error conditions as follows:

(1) Set PROC I/O switch to position PROC.

(2) Press and release switch A; observe DISPLAY SELECT indicators. If the third DISPLAY SELECT indicator from the left is on, an address error exists. If the fourth indicator from the left is on, a memory parity error exists.

b. Determine processor state as follows:

(1) Set PROC I/O switch to position PROC.

(2) Press and release switch B; observe DISPLAY SELECT indicators. If the third indicator from the left is on, the central processor unit is in the I/O state and the I/O state location counter points to the next instruction to be executed. (The processor state control points to one instruction beyond the point at which the processor left the processor state.) If the third indicator from the left is off, the processor is in the processor state and the processor state location counter points to the next instruction to be executed. The I/O state location counter is normally reinitialized by the supervisor when an exit from the I/O state takes place.

c. Press and release the CLEAR switch.

d. If the memory dump routine is not resident, display the contents of the following locations which are destroyed by the operation of the memory dump loader.

Location (Hexadecimal)	Data Displayed
2-3	Contents of processor state location counter
12-13	Contents of I/O state location counter
20-2F	Contents of processor state registers
30-3F	Contents of I/O state registers
42-43	Device status and device address

e. If the memory dump routine is resident, display the contents of the following locations which are destroyed by the execution of the memory dump.

Location (Hexadecimal)	Data Displayed
12-13	Contents of I/O state location counter
3C-3F	Contents of I/O registers 14 and 15
42-43	Device status and device address

f. Execute the memory dump as follows:

(1) If memory dump routine is not resident, load the routine by use of the normal card loading procedure.

(2) If memory dump routine is resident, alter locations 16 and 17 (hexadecimal) to indicate 47F0, and alter locations 18 and 19 (hexadecimal) as follows:

Hexadecimal Value	Memory Size
1DD2	8K
2DD2	12K
3DD2	16K
5DD2	24K
7DD2	32K

(3) Press and release CLEAR switch. This places the CPU in the I/O state and makes the instruction at locations 16 through 19 the next to be accessed. (This will be an unconditional branch to MENT.)

(4) Press and release the START switch. The resident memory dump routine is accessed.

4.6.5. Display Stops

A list of the various displays which may appear in the panel of the control console during the execution of a memory dump is provided in Table 4–1. A listing of the cause and, if required, the action to be taken is provided for each display.

DISPLAY (hexadecimal)	CAUSE AND ACTION
61C0	Reader off (normal during load operation). Restart from initial procedure.
63xx	Printer off (normal during memory dump). The error status bits are indicated by xx. Correct the indicated error condition and press and release the START switch to continue operation.
1FFF	Successful stop for the self-loading form of memory dump routine.
650C	Card controller off (normal during load operation). Reload routine and run again.

Table 4–1. Display Stops

5. SNAPSHOT ROUTINE

5.1. GENERAL

The snapshot routine provides a means of printing, during program execution, the contents of processor registers 8 through 15 and the contents of a specified area of main storage. Generation of the snapshot routine is accomplished by a call on the SNAP declarative macro instruction. This generation may be assembled as part of the user's program at assembly time or it may be assembled separately and incorporated into the user's program at linker time. If the routine is to be linked, the user program must define by means of EXTRN directives the labels ZXMP and ZXPT. These two labels represent, respectively, the symbolic address of the snapshot dump routine and the address of the PUT instruction used for printing the dump.

The snapshot routine is entered from the user program by means of a calling sequence consisting of a branch-and-link (BAL) instruction and two define-constant (DC) statements. The purpose of this calling sequence, which appears as coding in the user program, is to specify the address to which the program must branch in order to obtain the snapshot routine and to define the parameters that specify the starting and ending addresses of the memory locations to be dumped. It should be noted that the processor registers used by the snapshot routine (registers 8 through 15) are always dumped upon execution of the snapshot routine and that the contents of these registers are destroyed when the dump occurs. Since the routine operates in the processor state, it must not be used for dumping privileged areas of main storage.

The snapshot routine uses a PUT instruction to print the lines of the dump. The actual printing is accomplished by use of the bar printer. The IOCS for the printer, however, must be defined by use of the appropriately coded DTFPR declarative macro instruction in the user program. The user must also open the print routine prior to entering the snapshot routine.

Since the PUT instruction is used to print the lines of the dump, the snapshot routine will use whatever printer forms overflow provisions have been made in the print routine. The user, therefore, must be particularly observant of this fact if he has written his own forms-overflow routine.

5.2. MAIN STORAGE REQUIREMENTS

The snapshot routine requires 200 bytes of main storage. When assembling the snapshot routine, the user should make certain that the base address of the routine starts at a location greater than 240. This is to avoid the possibility of creating covering errors during assembly time.

5.3. SNAP DECLARATIVE MACRO INSTRUCTION

The SNAP declarative macro instruction, when called upon, generates the snapshot dump routine. SNAP contains two keyword parameters, BKSZ and REG.

The format of the SNAP declarative macro instruction is as follows:

LABEL	b OPERATION b	OPERAND
unused	SNAP	BKSZ=n,REG=n

- Blocksize (BKSZ) Keyword Parameter

  This keyword parameter specifies the number of characters that are to appear in each line printed by the printer. The minimum number of characters specified by this parameter must be equal to or greater than 40 and must be an even number. The maximum number of characters specified is limited by the maximum number of print positions per line of the printer. The format of the BKSZ parameter is as follows:

  BKSZ=n

  where: n may be any even decimal number between the values of 40 and 132.

- Register (REG) Keyword Parameter

  This required keyword parameter is used to specify the number of a processor register to be used by the snapshot routine. The processor register selected by this parameter is used in addition to processor registers 14 and 15, which are always used by the snapshot routine. The format of the REG keyword parameter is as follows:

  REG=n

  where: n may be any decimal number between 8 and 13. The number specified corresponds to the number of the processor register selected.

5.4. INSTRUCTIONS REQUIRED FOR ENTERING SNAPSHOT ROUTINE

The snapshot routine is entered from the problem program by means of a calling sequence consisting of a BAL instruction and two DC 'Y' statements. The format of these instructions and a description of their use are provided in the paragraphs which follow.

5.4.1. Branch-and-Link (BAL) Instruction

The BAL instruction is the instruction which specifies the address of the routine to which the program must branch.

The format of the BAL instruction used for calling the snapshot routine is as follows:

LABEL	OPERATION	OPERAND
[name]	BAL	14,ZXMP

5.4.2. Define-Constant (DC) Statement

The calling sequence for the snapshot routine requires the use of two DC 'Y' statements. One of the statements is used to define the main storage address at which the main storage dump proper is to begin. The other statement defines the main storage address at which the main storage dump proper is to end. The format for the DC 'Y' statements is as follows:

LABEL	OPERATION	OPERAND
unused	DC	Y (start)
unused	DC	Y (end)

where: "start" is either an absolute or relocatable expression representing the main storage address at which the dump is to begin and "end" is an absolute or relocatable expression representing the main storage address at which the dump is to end.

If the address specified by the start parameter is greater than the address specified by the end parameter, then the contents of processor registers 8 through 15 only will be dumped. A dump of main storage does not occur.

5.5. REQUIREMENTS FOR PRINTING SNAPSHOT DUMP

The printing of the snapshot dump is accomplished by means of a PUT instruction. This instruction must be stored by the user into locations ZXPT through ZXPT+5 prior to entering the snapshot routine. The parameters of the PUT instruction are user supplied since the IOCS for the printer is defined by the DTFPR macro instruction included as part of the user program. The symbolic name of the print routine specified in the PUT instruction must, therefore, agree with the name of the file defined in the label field of the DTFPR declarative macro instruction. The following is an example of the coding which might be executed when using the PUT instruction to print the snapshot dump.

1 LABEL	10 OPERATION	16 OPERAND
PAUL	PUT	PRNT,LINE
	MVC	ZXPT(6),PAUL

In the example shown, the print routine labeled PRNT transfers the contents of the work area labeled LINE for printing.

5.5.1. Print Format of Snapshot Dump

The print format of the snapshot dump consists of an address followed by a dump of a specified number of bytes. The address printed is the address of the first byte to appear in the associated line of print. The first line printed contains the address 0020 and a dump of eight half-words. Each half-word represents, respectively, the contents of processor registers 8 through 15 at the time that the snapshot routine was entered. The remaining lines of print contain the contents of those locations in main storage according to the address specified in the DC 'Y' statements. All information printed is formatted in hexadecimal notation.

6. SQUEEZE PROGRAM

6.1. GENERAL

The squeeze program is a utility program which enables a user to modify the contents of absolute program decks. The modification is accomplished by means of specially prepared input replace cards which specify in hexadecimal format the new data or text to be used in program n odification and the storage locations in which the new data are to be placed. The input cards to the squeeze program may be read from either the online serial card reader or the 1001 card controller. In either case, the user must specify at linker time the device to be used. The output produced by the squeeze program consists of output cards punched by either the online serial reader/punch or the row reader/punch. (The specific device used must also be specified by the user.) The output punch cards produced by squeeze are formatted so that they are acceptable to the loader but still contain the new data and storage locations required for program modification.

To allow the user to punch input statement cards in any code that he desires, the squeeze program utilizes a user-supplied translation table for code translations. The code produced by the translation is assumed to be EBCDIC. If the input cards are punched in Hollerith code, then the user may use the Hollerith-to-EBCDIC translation table supplied as part of the systems software package.

In addition to producing output punch cards, the squeeze program also prepares a printer listing of all START, REP, and transfer cards processed. Descriptions of the card inputs, the card outputs, and the module linking required to compose the squeeze program are provided in the paragraphs that follow.

6.2. CARD INPUT

The card input to the squeeze program consists of the START card, REP cards, and the END card. These cards are described in the following paragraphs in the sequence in which they must be read.

6.2.1. START Card

The START card is optional and not generally used. If used, the three-character expression contained in the operand field of the directive will be punched in columns 73-75 of each output text card and each transfer card generated by the squeeze program. If the START card is omitted, the PID in the output text card and the transfer card will be blank. The format of the START card is as follows:

LABEL	b OPERATION b	OPERAND
[name]	START	p

where: p equals the program identification (PID) in decimal or hexadecimal representation.

6.2.2. Replace (REP) Card

The REP card is used when it is desired to make changes or corrections to the assembled elements being linked. The contents of the REP card consist of the address of the leftmost byte of the data to be altered and the new data to be incorporated. When read, the contents of the REP card are used by the squeeze program to produce text cards which are placed immediately in front of the transfer card of the element to be altered. Where possible, the squeeze program combines the contents of two or more REP cards into one text card. The process described continues until an END card is read. A count is maintained of the number of text card produced by the squeeze program. The format of the REP card is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	REP	a,t,t,t,...,t♭

where:

a represents a field of one to four hexadecimal characters specifying the storage address of the leftmost byte of data that is to be altered as a result of the contents of this card. If less than four hexadecimal characters are specified, they are to be right justified and zeros are to fill the field.

t represents a field of one to four hexadecimal characters denoting the data that is to be right justified in the half-word storage address specified above. The number of data fields which may appear in the operand of the REP card is variable; however, they do specify the contents of successive half-words of storage. Fields are separated by a comma, and the last field must be followed by a blank column. (Column 71 is the last card column that may contain data; column 72 must be blank.)

6.2.3. END Card

The END card indicates that the last card of the program has been encountered. The format of the END card is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	END	{address / NO} ,n

- Positional Parameter 1

address – a one to four character hexadecimal address specifying the address to which control is to be transferred after elements of the REP card are loaded. An address must be provided if the squeeze program is to produce a transfer card.

NO – used when squeeze program is not to produce a transfer card.

- Positional Parameter 2

  n – a one to four character hexadecimal number specifying an increment to be added to the count of text cards previously accumulated by the squeeze program. The results of the incrementation become the card count punched into the transfer card produced by the squeeze program.

NOTE: The hexadecimal values presented in both positional parameter 1 and positional parameter 2 are right justified and, if necessary, the field is zerofilled.

In practice, the END card specification is usually NO. The squeeze program, however, produces text cards, which must be inserted into the card deck that is to be modified. These text cards are inserted immediately preceding the transfer card ('y' card) for the program. When the modified program is loaded, a card count error will be encountered since the count in the old 'y' card is no longer valid. This error condition may be bypassed and the program executed.

Alternatively, a new 'y' card may be generated as described in the preceding paragraph. Positional parameter 2 of the END card is then equal to the card count from the old 'y' card (the 'y' card replaced by the new card generated by the squeeze program).

6.3. OUTPUT

The output of the squeeze program consists of text cards, transfer cards, and a printed listing of all the START, REP, and transfer cards processed by the program. A description is provided for each output type produced by the squeeze program.

6.3.1. Text Card

The text cards produced by the squeeze program contain the instructions and constants of an element, an address indicating the location in which the instructions and constants are to be loaded into storage for execution, and the relocation information pertaining to the instructions and constants. If the text cards are placed 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 at the time that the absolute program deck is loaded. The format of the text card is as follows:

CARD COL.	FIELD NAME	CONTENTS
1	Load key	12-2-9 punch
2	Type	Q (Hollerith)
3	Text length	The number of columns of text information contained on this card
4-6	Load address	The assigned location where the text is to be loaded
7	Hole count	Sum of the bytes punched
11-72	Text	The value to be loaded at the load address specified
73-75	PID	Program identification (as indicated on START card, if present)

6.3.2. Transfer Card

The transfer card produced by the squeeze program can be used to replace the transfer card in an absolute program deck. The following is the format of the transfer card.

CARD COL.	FIELD NAME	CONTENTS
1	Load key	12-2-9 punch
2	Type	Y (Hollerith)
3	Length	Number of bytes of card information
7	Hole count	Sum of the bytes punched
11-13	Card count	Count of output text cards punched added to the card count field of the END card
14-16	Transfer address	Address where control is to be transferred after loading
73-75	PID	Program identification (as indicated on START card, if present)

6.3.3. Print Output

The printed output produced by the squeeze program consists of a listing of all START, REP, and transfer cards processed. At the time when the squeeze program is linked, the user determines whether the printer producing the list is to have a 63-character or a 48-character print bar.

The format of the listing is as follows:

Columns 1-4	Columns 6-85
Error Codes	Contents of input cards

The error codes and the reasons for their display are as listed.

Error Code	Reason
EDER	Edit error (format error)
CDER	Card error (refers to card other than START, REP, END, or blank)

When the squeeze program encounters one of the errors listed, the punching of text cards is terminated. The squeeze program will, however, continue to process the remaining input cards.

6.4. LINKING SQUEEZE PROGRAM MODULES

The squeeze program comprises several separately assembled modules which must be linked by means of a linker run in order to produce a loadable squeeze program. The order in which control cards are arranged in the input deck, as well as the order in which the modules are arranged in the input deck, is described in the following paragraphs.

6.4.1. Phase Control Card

The PHASE control card defines the name and initial storage address for the output element. The PHASE card used by the squeeze program contains three positional parameters and has the following format:

LABEL	OPERATION	OPERAND
unused	PHASE	SQZ,260,A

- Positional Parameter 1

  SQZ – alphabetic characters denoting the squeeze program as the phase name

- Positional Parameter 2

  260 – decimal number representing the starting address in which the squeeze program will be loaded into main storage

- Positional Parameter 3

  A – specifies that the load address provided in positional parameter 2 is an actual value

6.4.2. External Definition (EQU) Cards

The EQU card supplies the definition of a symbol which is not defined in any of the elements being linked or which is defined in an element whose position in the input deck is later than that of the first element containing a reference to the symbol. The EQU cards used in linking the squeeze program include those for the card load routine. These cards should be placed after the phase card and must have the following format.

LABEL	♭ OPERATION ♭	OPERAND
L?AR	EQU	6000
L?PG	EQU	6080
L?HI	EQU	x
L?LO	EQU	80
L?CH	EQU	0
L?AM	EQU	4

The definition of each of the symbols used in the label field of the EQU card is provided in the following list.

LABEL	MEANING
L?AR	Start of the read area (storage address 6000) for the card load routine
L?PG	Start of the coding of the card load routine (storage address 6080)
L?HI	Last of highest storage address to be cleared (x=8191 for 8K main storage; 12,287 for 12K main storage; 16,833 for 16K main storage; 24,575 for 24K main storage; and 32,767 for 32K main storage)
L?LO	First or lowest storage address (80) to be cleared
L?CH	Character with which to fill areas to be cleared (zero)
L?AM	Specifies the storage area (location 4) where alterations are to be stored

6.4.3. Arrangement of Modules in Input Deck

6.4.3.1. Card Load Routine

The card load routine must be the first module in the input deck. The name of this routine is determined by the device from which the input deck is loaded. That is, if the online is used to load the input deck, the routine is labeled LD. If the 1001 card controller is the device used, the routine is labeled LDCC.

6.4.3.2. Squeeze Module

The squeeze module (labeled SQZ) must be the last module of the input deck. The remaining modules comprising the squeeze program may appear in any order desired between the card load routine and the SQZ module.

6.4.3.3. Exec I

Exec I module, named EXEC, may appear in any position between the card load routine and the SQZ module. The primary function of this routine is to monitor interrupts, handle messages to and from an operator, and provide restart communications.

6.4.3.4. Card Read Routine

This routine applies to either the online serial reader or the 1001 card controller and must be generated by use of the Preassembly Macro Pass and the card assembler. If the online serial reader is used, the routine is generated by means of the DTFCR declarative macro instruction. If the 1001 card controller is used, the routine is generated by means of the DTFCC declarative macro instruction. The format for these two declarative macro instructions is provided as follows:

- DTFCR Declarative Macro Instruction

LABEL	♭ OPERATION ♭	OPERAND
CRDR	DTFCR	SENT=NO,MODE=TRANS,ITBL=TBRD,IOA1=REWA

– End-of-File (SENT) Keyword Parameter

This keyword parameter specifies whether or not the end-of-file sentinel is to be recognized. The format of the SENT parameter is as follows:

SENT=NO

This format of the SENT parameter indicates that an end-of-file sentinel is not to be recognized.

– MODE Keyword Parameter

This parameter specifies that the cards are to be read after the translation specified by the ITBL keyword parameter of this instruction has been performed.

The format of the MODE parameter is as follows:

MODE=TRANS

This format specifies that the translation must be performed.

– Input Translation Table (ITBL) Keyword Parameter

This parameter specifies the name of the translation table to be used. If this table is the EBCDIC input translation table that is included as a relocatable module in the card libraries of the UNIVAC 9200/9300 software package, then the following format is used.

ITBL=TBRD

This format specifies that the Univac-supplied input translation table is to be used. If the user desires to use his own translation table, the ITBL parameter has the following format:

ITBL=name

where: name is the user-supplied name consisting of one to four alphabetical characters.

– Input/Output Area (IOA1) Keyword Parameter

This parameter specifies the address of the input buffer area.

The format of the IOA1 parameter is as follows:

IOA1=REWA

■ DTFCC Declarative Macro Instruction

LABEL	♭ OPERATION ♭	OPERAND
CRDR	DTFCC	MODE=TRANS,ITBL=TBRD,FUNC=FUNC,CHNL=n

– MODE=TRANS Keyword Parameter

This parameter specifies that the cards are to be read after the translation specified by the ITBL keyword parameter of this instruction has been performed.

The format of the MODE parameter is as follows:

MODE=TRANS

This format specifies that the translation must be performed.

– Input Translation Table (ITBL) Keyword Parameter

This parameter specifies the name of the input translation table to be used. If the EBCDIC input translation table included as a relocatable module in the card libraries of the UNIVAC 9200/9200 II/9300/9300 II Systems software package is to be used, the format of the ITBL parameter is as follows:

ITBL=TBRD

If the user desires to utilize his own input translation table, the format of the ITBL is as follows:

ITBL=name

where: name is the user-supplied name consisting of one to four alphabetical characters.

– Function (FUNC) Keyword Parameter

This parameter identifies the function area labeled FUNC as the area where the user-supplied function code is stored.

The format of the FUNC parameter is as follows:

FUNC=FUNC

– Channel (CHAN) Keyword Parameter

This keyword parameter specifies the channel number to which the 1001 card controller is connected. The format of the CHAN parameter is as follows:

CHAN=n

The value of n may be any decimal number from 5 to 12.

6.4.3.5. Card Punch Routine

This routine can be a routine for either the online serial read/punch unit or the row read/punch unit and must be generated by use of the Preassembly Macro Pass and the card assembler. If the online serial read/punch unit is used, the module is generated by means of the DTFRP declarative macro instruction. If the row read/punch unit is used, then the module is generated by means of the DTFRW declarative macro instruction. The format for these two instructions is as follows:

■ DTFRP Declarative Macro Instruction

LABEL	↹ OPERATION ↹	OPERAND
CRPH	DTFRP	MODE=CC,OUAR=PUTX,PUNR=YES,TYPE=OUTPUT

– MODE Keyword Parameter

This parameter specifies that the cards are to be read in compressed card code.

The format for the MODE parameter is as follows:

MODE=CC

where: CC represents compressed code.

– Output Area (OUAR) Keyword Parameter

This parameter identifies the output buffer area. The format for the OUAR parameter is as follows:

OUAR=PUTX

This format specifies that the output buffer area is labeled PUTX.

– Punch Error (PUNR) Keyword Parameter

This parameter specifies that the punch will make five attempts to punch after a punch error is detected. The format of the PUNR parameter is as follows:

PUNR=YES

– Type of File (TYPE) Keyword Parameter

This parameter is needed to define the type of file since the serial read/punch unit can be used as an input or output device. The format of the TYPE parameter is as follows:

TYPE=OUTPUT

This format of the TYPE parameter defines the file as an output file.

■ DTFRW Declarative Macro Instruction

LABEL	OPERATION	OPERAND
CRPH	DTFRW	MODE=CC,OUAR=PUTX,TYPE=OUTPUT,CHNL=n

– MODE Keyword Parameter

This required parameter specifies that the cards are read in compressed card code. The format of the MODE parameter is as follows:

MODE=CC

where: CC represents compressed code.

– Output Area (OUAR) Keyword Parameter

This required parameter identifies the output buffer area labeled PUTX. The format for the OUAR parameter is as follows:

OUAR=PUTX

This format identifies PUTX as the label of the output buffer area.

– Type of File (TYPE) Keyword Parameter

This keyword parameter defines the type of file. It is required because the row read/punch unit can be used as either an input or an output device. The format of the TYPE parameter is as follows:

TYPE=OUTPUT

This format of the TYPE parameter defines the file as an output file.

– Channel Number (CHNL) Keyword Parameter

The CHNL keyword parameter is used to specify the number of the channel to which the row read/punch unit is connected. The format of the CHNL parameter is as follows:

CHNL=n

The value of n may be any decimal number from 5 to 12.

6.4.3.6. Print Routine

This module is generated from a call on the DTFPR declarative macro instruction. The format for this macro instruction is as follows:

■ DTFPR Declarative Macro Instruction

LABEL	OPERATION	OPERAND
PRTR	DTFPR	BKSZ=96, FONT=$\left\{\begin{matrix}48\\63\end{matrix}\right\}$, PROV=YES, PRAD=$\left\{\begin{matrix}1\\2\end{matrix}\right\}$ $\left[,\text{OTBL}=\left\{\begin{matrix}\text{name}\\\text{TBRP}\end{matrix}\right\}\right]$

– Block Size (BKSZ) Keyword Parameter

This keyword parameter specifies the number of bytes that are to be moved from the work area of storage to the printer buffer area. The format of the BKSZ parameter is as follows:

BKSZ=96

– FONT Keyword Parameter

This keyword parameter specifies the print bar requirements of the print routine. If the routine requires a 48-character printer bar, the FONT parameter must indicate a 48 and the output translation table (OTBL) parameter must be specified. The format of the FONT parameter for a 48-character print bar is as follows:

FONT=48

If a 63-character print bar is to be used in the printer, then the FONT parameter must indicate a 63, and the OTBL parameter must be omitted from the macro instruction. The format of the FONT parameter for a 63-character print bar is as follows:

FONT=63

– Output Translation Table (OTBL) Keyword Parameter

This parameter specifies that the output file is to be translated by means of the TBRP translation table supplied as part of the UNIVAC software package. This parameter must be included whenever the FONT=48 parameter is used. If the user does not desire to use the TBRP translation table supplied, he must provide an EBCDIC-to-48-character print code translation table and include this table in the linker input deck.

The format of the OTBL parameter for a user-supplied output translation table is as follows:

OTBL=name

where: name may be one to four alphabetical characters representing the name of the user-supplied output translation table.

The format of the OTBL parameter for the UNIVAC-supplied translation table is as follows:

OTBL=TBRP

– Printer Overflow (PROV) Keyword Parameter

This keyword parameter specifies that the overflow routine provided by the IOCS is used if an overflow condition occurs. When this parameter is specified, make certain that a paper loop containing form overflow and home paper punch holes is in the printer. The format of the PROV parameter is as follows:

PROV=YES

– Printer Advance (PRAD) Keyword Parameter

This parameter specifies the number of lines that the printer advances after completing a line of print. If the printer is to advance one line after each line of print, the PRAD parameter has the following format:

PRAD=1

If the printer is to advance two lines after each line of print, the PRAD parameter has the following format:

PRAD=2

6.4.4. END Control Card

The END control card is used to identify the end of the squeeze program. The format of the END control card is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	END	0,SQZ

6.5. OPERATING INSTRUCTIONS

Operating instructions pertaining to the squeeze problem are stated in general terms and in the sequence in which they are to be performed.

a. Place squeeze program in the reader followed by the input cards.

b. Load cards. Completion of run is signified by a '1FFF' display on the control console.

c. To process another set of input cards, place input in reader and press and release START switch.

7. SYMBOLIC LIST PROGRAM

7.1. GENERAL

The symbolic list (SYM) program is used to generate a printed listing of the information contained on program input source cards. This program also generates a printed listing of all the symbolic tags used within the source deck.

The cards of the input source deck may be read from either of two read devices: the online serial card reader or the 1001 card controller. The user, however, must specify at linker time the specific device that is going to be used.

The format of the input source cards to the program is read in compressed code. This code must be translated before the contents of the card can be processed. The translation may be accomplished by use of a user-supplied input translation table or, in the case of cards punched in Hollerith code, by the Hollerith-to-EBCDIC input translation table supplied as part of the Univac-supplied software package. The results of the translation always are assumed to be in EBCDIC code.

To aid in a clearer understanding of the symbolic list program, descriptions of the requirements of the program, of the modules comprising the program, as well as the linking of these modules, are presented in this section. Information concerning the macro instruction used for generating the various modules of the symbolic list program is also provided.

7.2. PROGRAM REQUIREMENTS

The input to the symbolic list routine consists of program source cards. The source card handling capability of the SYM program consists of approximately 600 source cards for an 8K main storage system, 1400 cards for a 16K system, 2200 cards for a 24K system, and 3000 cards for a 32K system. Prior to being processed, these source cards must be translated to EBCDIC code. Once translated to EBCDIC code and processed, the contents of these cards will be listed by the printer. The print list will contain a line number followed by the entire contents of one input source card. The program prints the contents of one line per source card read. Therefore, a printer having a minimum of 96 print positions is required for this program.

Following the source card listing is a summary listing of all the symbolic tags used in the source card statements. This information is formatted so that each four-character symbol is printed followed by the line number in which the symbol first occurred and the line numbers in which the symbol is referenced. The symbolic summary is printed in an alphanumeric sequence by symbol.

7.3. LINKING SYMBOLIC LIST PROGRAM

The symbolic list program comprises several separately assembled modules which must be linked together by means of a linker in order to produce a loadable symbolic list program. The control cards and modules comprising the symbolic list program, as well as their arrangement in the input deck, are described in the paragraphs which follow.

7.3.1. Phase Control Card

The SYM program has one phase. It utilizes the PHASE control card to define the name and initial storage address for the output element of the program. The format of the PHASE card depends upon the device from which the program is to be loaded. That is, if the input deck is read from the online serial card reader, the SYM program must be linked to the card load (LD) routine and the PHASE card must have the following format:

LABEL	♭ OPERATION ♭	OPERAND
unused	PHASE	SRM,410,A

If the input deck is read from the 1001 card controller, the SYM program must be linked to the card controller load (LDCC) routine, and the PHASE card must have the following format:

LABEL	♭ OPERATION ♭	OPERAND
unused	PHASE	SCM,500,A

7.3.2. External Definition (EQU) Card

The EQU card supplies the definition of a symbol which is not defined in any of the elements being linked or which is not defined in an element whose position in the input deck is later than that of the first element containing a reference to that symbol. The EQU card used in linking the SYM program is required for the input translation table. The purpose of the EQU card is to equate the input translation table to the label TRN. If the input translation table is user-supplied, then the format of the EQU card is as follows:

LABEL	♭ OPERATION ♭	OPERAND
TRN	EQU	0,name

where:

name is one to four alphabetic characters representing the name of the user-supplied input translation table

If the user elects to use the supplied Hollerith-to-EBCDIC input translation table, then the format of the EQU card is fixed as follows:

LABEL	♭ OPERATION ♭	OPERAND
TRN	EQU	0,TBRD

7.3.3. Arrangement of Modules in Input Deck

The arrangement of modules in the input deck is discussed in the following paragraphs.

7.3.3.1. Card Load Routine

The first element in the input deck must be the card load routine. The program name for this routine is LD when the online serial card reader is used to load the SYM program being linked. If the 1001 card controller is being used to load the SYM program, the name of the load routine is labeled LDCC.

7.3.3.2. SYM Module

The SYM module must be the last element in the input deck. The remaining modules comprising the SYM program may appear in any order between the card load routine and the SYM module.

7.3.3.3. Exec I

Exec I module, named EXEC, may appear in any order between the card load routine and the SYM module. The primary function of the Exec I module is to monitor interrupts, handle messages to and from an operator, and provide restart communications.

7.3.3.4. Card Read Routine

This routine may also appear in any order between the card load routine and the SYM module. The card read routine applies to either the online serial card reader or the 1001 card controller. The format of the declarative macro instruction generating the card read routine depends upon the device from which the input cards are read. For example, the card read routine is generated by means of DTFCR declarative macro when the input cards are loaded from the online serial card reader. A DTFCC declarative macro instruction is used when the 1001 card controller is used to load the input cards. The format for both the DTFCR and the DTFCC is as follows:

- DTFCR Declarative Macro Instruction

LABEL	♭ OPERATION ♭	OPERAND
CRDR	DTFCR	MODE=CC,SENT=NO,IOA1=IOA1,FUNC=FUNC

– MODE Keyword Parameter

This parameter specifies the mode in which the cards are to be read. The format of the MODE parameter is as follows:

MODE=CC

The format specifies that the cards are to be read in compressed code.

– End-of-File (SENT) Keyword Parameter

This parameter specifies whether or not an end-of-file sentinel is to be recognized. The format of the SENT parameter is as follows:

SENT=NO

This format indicates that the end-of-file sentinel is not recognized by the card read routine.

– Input/Output Area (IOA1) Keyword Parameter

This keyword parameter specifies the name of the address for the input/output buffer area. The format of the IOA1 parameter is as follows:

IOA1=IOA1

– Function (FUNC) Keyword Parameter

This keyword parameter specifies that the function area labeled FUNC is the area where the user-supplied function code is stored. The format of the FUNC parameter is as follows:

FUNC=FUNC

■ DTFCC Declarative Macro Instruction

LABEL	OPERATION	OPERAND
CRDR	DTFCC	MODE=CC,CHNL=n,FUNC=FUNC

– MODE Keyword Parameter

This keyword parameter specifies that the cards are to be read in compressed code. The format of the MODE parameter is as follows:

MODE=CC

In this format, CC signifies compressed code.

– Channel (CHNL) Keyword Parameter

This keyword parameter specifies the channel number to which the 1001 card controller is connected. The format for the CHNL parameter is as follows:

CHNL=n

In this format, the value of n may be any decimal number from 5 to 12.

– Function (FUNC) Keyword Parameter

This keyword parameter identifies the label assigned to the function area in which the user-supplied function code is stored. The format for the FUNC parameter is as follows:

FUNC=FUNC

In this format, the function area containing the user-supplied function code is labeled FUNC.

7.3.3.5. Print Routine

This routine is placed anywhere between the card load routine and the SYM module. The print routine is generated from a call on the DTFPR declarative macro instruction. The format for this macro instruction is as follows:

LABEL	♭ OPERATION ♭	OPERAND
PRNT	DTFPR	BKSZ=96, PROV=YES, PRAD= {1 | 2} , FONT= {48 | 63} [,OTBL= {name | TBPR}]

– Block Size (BKSZ) Keyword Parameter

This keyword parameter specifies the number of bytes to be moved from the work area of storage to the printer buffer area. The format of the BKSZ parameter is as follows:

BKSZ=96

In this format, 96 bytes are moved to the printer buffer area.

– Printer Overflow (PROV) Keyword Parameter

This keyword parameter specifies that the overflow routine provided by the IOCS is used if an overflow condition occurs. When this parameter is specified, make certain that a paper loop containing form overflow and home punch holes is in the printer. The format for the PROV parameter is as follows:

PROV=YES

– Printer Advance (PRAD) Keyword Parameter

This keyword parameter specifies the number of lines that the printer advances after completing a line of print. PRAD may specify a one-line or two-line advancement. When a one-line advancement is specified, the PRAD parameter has the following format:

PRAD=1

When a two-line advancement is specified, the PRAD parameter has the following format:

PRAD=2

– FONT Keyword Parameter

This keyword parameter specifies the type of print bar to be used in the printer. The print bar specified by this parameter should agree with the print bar used in the printer during linker execution. If a 48-character print bar is to be used, the FONT parameter must equal 48, and the output translation table parameter OTBL must also be specified. The format of the FONT parameter for a 48-character print bar is as follows:

FONT=48

When a 63-character print bar is to be used in the printer, the format of the FONT parameter is as follows:

FONT=63

The OTBL parameter is not required when the 63-character print bar is specified.

– Output Translation Table (OTBL) Keyword Parameter

The OTBL parameter is used whenever the FONT parameter equals 48. The OTBL parameter specifies that the output file must be translated by means of a user-supplied translation table or by the Univac-supplied translation table labeled TBPR. If the translation is performed by the user-supplied translation table, the format of the OTBL parameter is as follows:

OTBL=name

where:

name may be one to four alphabetical characters representing the name of the user's output translation table.

If the translation is accomplished by means of the Univac-supplied TBPR output translation table, then the format of the OTBL parameter is as follows:

OTBL=TBPR

7.3.4. END Control Card

The END control card is used to identify the end of the SYM program. The format of the END control card is as follows:

LABEL	♭ OPERATION ♭	OPERAND
unused	END	0,BEGN

If desired, blank cards may be inserted between the modules in the input deck. This is a convenient method of separating the modules since the linker ignores blank cards.

7.4. OPERATING INSTRUCTIONS FOR USING THE SYM PROGRAM

The operating instructions provided are expressed in general terms and are given in the sequence in which they are to be performed.

(1) Place the SYM program in the read device selected, followed by the source.

(2) Feed and load the SYM program deck. An 0F00 display appears on the control console after the SYM program has been loaded.

(3) Perform the following, according to the main storage size of the particular system being operated.

- If main storage equals 8K, key 1F into the location assigned to TBEA symbol by the linker; then press and release RUN switch on control console.
- If main storage equals 16K, press and release the RUN switch on the control console.
- If main storage equals 24K, key 5F into the location assigned to TBEA symbol by the linker; then press and release RUN switch on control console.
- If main storage equals 32K, key 7K into the location assigned to TBEA symbol by the linker; then press and release the RUN switch on the control console.

(4) Observe HALT DISPLAY on the control console. If all allotted storage has been used by the SYM program, the stop display 1F1F will be displayed on the control console. Press and release the RUN switch; this prints the symbolic summary of all the source cards processed.

(5) The end of job is indicated by the 1FFF display on the control console.

APPENDIX A. STANDARD - CARD EBCDIC, AND PRINTER - GRAPHIC CODES

A.1. GENERAL

UNIVAC 9200/9200 II/9300/9300 II software is designed to use the standard-card, EBCDIC, and printer-graphic codes shown in the tables covering the two most significant bits of zones 00, 01, 10, and 11.

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

TWO MOST SIGNIFICANT BITS OF ZONE – 01

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000	¤	12 &	11 –	12-11-0
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 "

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

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


UNIVAC
SPERRY RAND