SPERRY RAND

UNIVAC® 9200/9300

CARD ASSEMBLER

PROGRAMMERS REFERENCE

UNIVAC
FEDERAL SYSTEMS DIVISION

UNIVAC 9200/9300 Systems
Card Assembler Programmers Reference
UP-4092 Rev. 2

June 27, 1968

UPDATING PACKAGE "B"

UNIVAC 9200/9300 Systems P.I.E. Bulletin 11, UP-7535.11, announces the release and availability of Updating Package "B" for the "UNIVAC 9200/9300 Systems Card Assembler Programmers Reference," UP-4092 Rev. 2, 25 pages plus 1 Updating Summary Sheet. This material should be utilized in the following manner:

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Contents	3 Rev. 1 and 4	3 Rev. 1* & 4 Rev. 1
	5 Rev. 1 and 6 Rev. 1	5 Rev. 2 & 6 Rev. 2
	7 Rev. 1	7 Rev. 2
Section 2	5 & 6	5* & 6 Rev. 1
	7 & 8	7 Rev. 1 & 8*
Section 3	1 & 2	1 Rev. 1 and 2*
Appendix A	13 Rev. 1 and 14	13 Rev. 1* & 14 Rev. 1
Appendix B	7 Rev. 1 & 8 Rev. 1	7 Rev. 2 & 8 Rev. 1*
	37 Rev. 1 & 38 Rev. 1	37 Rev. 2 & 38 Rev. 2
	39	39 Rev. 1 & 40**
Appendix C	1 & 2	1 Rev. 1 and 2 Rev. 1
	3 Rev. 1 & 4 Rev. 1	3 Rev. 1* & 4 Rev. 2
	N. A.	5** & 6**

*These are backups of revised pages, and remain unchanged.
**These are new pages.

UNIVAC 9200/9300 Systems
Card Assembler Programmers Reference
UP-4092 Rev. 2

March 15, 1968

UPDATING PACKAGE "A"

UNIVAC 9200/9300 Systems P.I.E. Bulletin 8, UP-7535.8, announces the release and availability of Updating Package "A" for the "UNIVAC 9200/9300 Systems Card Assembler Programmers Reference," UP-4092 Rev. 2, 66 pages plus 1 Updating Summary Sheet. This material should be utilized in the following manner:

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGES NUMBERED
Contents	1 and 2	1* and 2 Rev. 1
	3 and 4	3 Rev. 1 and 4*
	5 and 6	5 Rev. 1 and 6 Rev. 1
	7	7 Rev. 1
3	3 and 4	3 Rev. 1 and 4 Rev. 1
	5 and 6	5 Rev. 1 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8 Rev. 1
	11 and 12	11 Rev. 1 and 12 Rev. 1
4	1 and 2	1 Rev. 1 and 2 Rev. 1
5	5 and 6	5 Rev. 1 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8*
	15 and 16	15* and 16 Rev. 1
	21	21 Rev. 1
Appendix A	5 and 6	5 Rev. 1 and 6 Rev. 1
	7 and 8	7 Rev. 1 and 8*
	13 and 14	13 Rev. 1 and 14*
	19 and 20	19 Rev. 1 and 20*
	21 and 22	21* and 22 Rev. 1
	23 and 24	23 Rev. 1 and 24*
	25 and 26	25* and 26 Rev. 1
Appendix B	7 and 8	7 Rev. 1 and 8 Rev. 1
	9 and 10	9 Rev. 1 and 10*
	11 and 12	11 Rev. 1 and 12*
	17 and 18	17 Rev. 1 and 18*
	21 and 22	21* and 22 Rev. 1
	23 and 24	23* and 24 Rev. 1
	25 and 26	25 Rev. 1 and 26*
	27 and 28	27 Rev. 1 and 28*
	29 and 30	29 Rev. 1 and 30*
	31 thru 38	31 thru 38 all Rev. 1
Appendix C	3 and 4	3 Rev. 1 and 4 Rev. 1

*These pages are backups of revised pages, and remain unchanged.

CONTENTS

CONTENTS	1 to 7
1. INTRODUCTION	1-1 to 1-4
1.1. GENERAL	1-1
1.2. THE PURPOSE OF AN ASSEMBLER	1-1
1.3. CARD ASSEMBLER FOR THE UNIVAC 9200/9300	1-2
1.4. ASSEMBLY LANGUAGE CHARACTERISTICS	1-4
2. THE ASSEMBLER LANGUAGE	2-1 to 2-15
2.1. CHARACTER SET	2-1
2.2. STATEMENT FORMAT	2-1
2.2.1. Label Field	2-1
2.2.2. Operation Field	2-1
2.2.3. Operand Field	2-1
2.2.4. Comments Field	2-2
2.3. EXPRESSIONS	2-2
2.3.1. Decimal Representation	2-2
2.3.2. Hexadecimal Representation	2-3
2.3.3. Character Representation	2-4
2.3.4. Location Counter	2-4
2.3.5. Relative Addressing	2-4
2.3.6. Symbols	2-5
2.3.7. Relocatable and Absolute Expressions	2-5
2.3.8. Length Attribute	2-6
2.4. MACHINE INSTRUCTIONS	2-6
2.4.1. RX – Register to Storage Instructions	2-8
2.4.2. SI – Instruction to Storage Instructions	2-8
2.4.3. SS1 – Storage to Storage Instructions	2-9
2.4.4. SS2 – Storage to Storage Instructions	2-10
2.4.5. Implied Base Register and Length	2-10
2.5. DATA AND STORAGE FORMATS	2-11
2.5.1. DC – Define Constant	2-12
2.5.1.1. Character Representation	2-12
2.5.1.2. Hexadecimal Representation	2-13
2.5.1.3. Expression Constants	2-13
2.5.2. DS – Define Storage	2-14

3. ASSEMBLER DIRECTIVES AND SYSTEM CODES	3-1 to 3-18
3.1. DIRECTIVES	3-1
3.1.1. Symbol Definition	3-1
3.1.2. Assembly Control	3-2
3.1.2.1. START – Program Start	3-2
3.1.2.2. END – Program End	3-3
3.1.2.3. ORG – Set Location Counter	3-3
3.1.3. Base Register Assignment	3-4
3.1.3.1. USING – Assign Base Register	3-5
3.1.3.2. DROP – Unassign Base Register	3-5
3.1.3.3. Function of USING and DROP Directives	3-5
3.1.3.4. Direct Addressing	3-7
3.1.4. Program Linking	3-7
3.1.4.1. ENTRY – Externally Defined Symbol Declaration	3-8
3.1.4.2. EXTRN – Externally Referenced Symbol Declaration	3-8
3.1.5. Assembler Program Listing	3-8
3.1.6. Assembler Control Card	3-14
3.1.7. Operand Format	3-14
3.2. SYSTEM CODES	3-14
4. OPERATING PROCEDURES	4-1 to 4-9
4.1. GENERAL OPERATING INSTRUCTIONS	4-1
4.1.1. Card Controller Operating Instructions	4-1
4.1.1.1. Start Instructions	4-1
4.1.1.2. Second Pass Rerun Instructions	4-2
4.2. ASSEMBLER CARD OUTPUT	4-2
4.2.1. Element Definition Card	4-3
4.2.2. External Definition Card	4-4
4.2.3. Program Reference Card	4-4
4.2.4. External Reference Card	4-5
4.2.5. Text Card	4-5
4.2.6. Transfer Card	4-6
4.3. CARD ASSEMBLER PRINTED OUTPUT	4-7
4.4. LINKING THE CARD ASSEMBLER	4-8
5. LINKER	5-1 to 5-21
5.1. INTRODUCTION	5-1
5.2. LINKER INPUT	5-2
5.3. LINKER CONTROL CARD FORMATS	5-2
5.3.1. CTL	5-3
5.3.2. PHASE	5-3
5.3.3. EQU	5-4
5.3.4. END	5-4
5.3.5. REP	5-5
5.3.6. MOD	5-5
5.4. EXAMPLE	5-6
5.5. ONE- AND TWO-PASS LINKING	5-13

5.6. LINKING THE LINKER 5-13

5.7. CARD OUTPUT FROM THE LINKER 5-18
5.7.1. Type Q Cards 5-18
5.7.2. Type Y Cards 5-18

5.8. LINKER MAP 5-19
5.8.1. Linker Map Print Lines 5-19
5.8.2. Linker Map Error Messages 5-20

5.9. LINKER CONSOLE DISPLAYS 5-21

APPENDICES

A. PREASSEMBLY MACRO PASS A-1 to A-30

A1. GENERAL DESCRIPTION A-1

A2. MACRO INSTRUCTION FORMAT A-2
A2.1. Parameters A-2

A3. WRITING MACRO DEFINITIONS A-4
A3.1. PROC Directive A-4
A3.2. NAME Directive A-5
A3.3. END Directive A-5
A3.4. Comments A-5

A4. INCORPORATING PARAMETERS INTO MACRO CODING A-6

A5. NAME STATEMENT A-7

A6. CONDITIONAL MACRO PASS INSTRUCTIONS A-9
A6.1. DO and ENDO Directives A-9
A6.2. GOTO and LABEL Directives A-11
A6.3. Set Variables A-12
A6.3.1. GBL Directive A-12
A6.3.2. LCL Directive A-13
A6.3.3. SET Directive A-13
A6.3.4. Relational and Logical Operators A-13
A6.3.5. Character Values A-14
A6.3.6. Use of Character Values A-14

A7. CONTINUATION CARDS A-17

A8. LABELS USED IN UNIVAC PRODUCED MACROS A-17

A9. MACRO INSTRUCTION DECK A-17

A10. MACRO PASS OUTPUT FORMAT A-18
A10.1. Source Code Card Format A-18
A10.2. Macro Instruction Card Format A-18
A10.3. Comments Card Format A-18
A10.4. Error Card Format A-19

A11. MACRO PASS CONSOLE DISPLAYS A-19

A12. LINKING THE MACRO PASS A-21
A12.1. Operating Instructions A-22
A12.2. Control Card A-23

A13. THE COMPRESSOR A-23
A13.1. Compressed Macro Library Deck Format A-23
A13.1.1. Data Cards A-24
A13.1.2. Fixups A-24
A13.1.3. Header Specification A-25
A13.2. Error Indications A-25
A13.3. Compressor Console Displays A-27
A13.4. Linking the Compressor A-28
A13.5. Operating Instructions A-30
A13.6. Control Card A-30

B. INPUT/OUTPUT CONTROL SYSTEM (IOCS) B-1 to B-40

B1. GENERAL DESCRIPTION B-1

B2. GENERAL USAGE B-1

B3. DEFINITION STATEMENTS (DECLARATIVE MACROS) B-2
B3.1. Header Entry Card B-2
B3.2. Detail Entry Cards B-2
B3.2.1. Block Size Entry (BKSZ) B-3
B3.2.2. Channel Entry (CHNL) B-3
B3.2.3. Control Entry (CNTL) B-3
B3.2.4. End-of-File Address Entry (EOFA) B-3
B3.2.5. The Function Entry – UNIVAC 1001 Card Controller (FUNC) B-4
B3.2.6. Allowable Functions for the UNIVAC 1001 Card Controller B-4
B3.2.6.1. Transfer-and-Read Functions B-5
B3.2.6.2. Send-and-Receive Data Functions B-5
B3.2.7. Input Area Entry (IOA1) B-6
B3.2.8. Input Area Entry (INAR) B-6
B3.2.9. Input Translate Table Entry (ITBL) B-6
B3.2.10. Mode Detail Entry (MODE) B-6
B3.2.11. Output Area Entry (OUAR) B-7
B3.2.12. Output Translate Table (OTBL) B-8
B3.2.13. Overlap Entry (ORLP) B-8
B3.2.14. Print Bar Entry (FONT) B-8
B3.2.15. Printer Advance Entry (PRAD) B-8
B3.2.16. Punch Error Entry (PUNR) B-8
B3.2.17. Printer Overflow Entry (PROV) B-9
B3.2.18. Type of File Entry (TYPF) B-10

B4. SUMMARY OF DETAIL ENTRY CARDS B-11

B5. DEFINITION STATEMENT EXAMPLES B-12
B5.1. Online Serial Punch File Example Definition B-12
B5.2. Reader File Example Definition B-12
B5.3. Printer File Example Definition B-12
B5.4. Online Serial Read and Punch File Example B-13
B5.5. Card Controller File Example B-13

B6. IOCS MACRO INSTRUCTIONS (IMPERATIVE MACROS) B-13
B6.1. GET Macro Instruction B-13
B6.2. PUT Macro Instruction B-14
B6.3. Work Area Considerations B-14
B6.4. Programming Considerations-Read/Punch Combined File B-14
B6.5. OPEN Macro Instruction B-15
B6.6. CLOSE Macro Instruction B-15
B6.7. CNTRL Macro Instruction B-15
B6.7.1. Printer Spacing B-15
B6.7.2. Printer Skipping B-16
B6.7.3. Stacker Select B-16
B6.7.4. Numeric Printing B-17
B6.7.5. Specifying Columns to be Punched B-18
B6.8. Summary of UNIVAC 9200/9300 Card System IOCS Imperative Macros B-18

B7. PROGRAMMING CONVENTIONS – PROGRAM REGISTERS B-19

B8. GENERAL PROCEDURE SUMMARY FOR USING IOCS B-19

B9. STORAGE REQUIREMENTS B-19

B10. APPROXIMATE TIMES FOR IOCS ROUTINE EXECUTION B-20

B11. CARD READER DEFINITION STATEMENTS B-21
B11.1. Preparing the Card Reader B-21
B11.2. Error Indications B-22

B12. PRINTER DEFINITION STATEMENTS B-23
B12.1. Preparing the Printer B-23
B12.2. Error Indications B-24
B12.3. Paper Low B-25

B13. SERIAL PUNCH DEFINITION STATEMENTS B-25

B14. SERIAL READ DEFINITION STATEMENTS B-26

B15. SERIAL READ/PUNCH DEFINITION STATEMENTS B-27
B15.1. Buffer and Work Area Size B-27
B15.2. End-of-File B-28
B15.3. Preparing the Serial Read/Punch B-28
B15.4. Error Indications B-29

B16. UNIVAC 1001 CARD CONTROLLER DEFINITION STATEMENTS B-30
B16.1. Work Area Size B-31
B16.2. Preparing the Card Controller B-31
B16.3. Error Indications B-32
B16.3.1. STOP 1 (65xx) B-32
B16.3.2. STOP 2 (65yy) B-34

B17. ROW READ/PUNCH DEFINITION STATEMENTS B-35
B17.1. Punch Only B-35
B17.2. Read Only B-36
B17.3. Read and Punch B-36
B17.4. Buffer and Work Area Size B-37
B17.5. End-of-File B-37
B17.6. Preparing the Row Read/Punch B-37
B17.7. Error Indications B-38

B18. IOCS GENERATION B-40

B19. ADDITIONAL KEYWORD PARAMETER SPECIFICATIONS B-40

C. CARD LOAD ROUTINE C-1 to C-6

C1. GENERAL C-1

C2. PARAMETERS FOR THE LOAD ROUTINE C-1

C3. LOADING ADDITIONAL PROGRAMS C-2

C4. LOAD ROUTINE STOPS C-3

C5. DESCRIPTION OF OPERATION C-3
C5.1. Bootstrap Section C-3
C5.2. Clearing Section C-4
C5.3. Reader Section C-4
C5.4. Loader Section C-4

C6. PROGRAMMING CONSIDERATIONS C-4

C7. LOADING FROM CARD READER C-5

C8. 1001 LOADER LOADING PROCEDURE C-5

D. EXEC I D-1 to D-3

D1. GENERAL D-1

D2. MACRO INSTRUCTIONS D-1
D2.1. Message Macro (MSG) D-1
D2.2. Restart Macro D-2

D3. I/O CONTROL ROUTINE MESSAGES D-3

E. TRANSLATION TABLES E-1 to E-1

E1. GENERAL E-1

FIGURES

1-1. Source-to-Object Code Translation with Assembler 1-1

1-2. UNIVAC 9200/9300 Assembly System 1-3

2-1. Example of Source Code Statements 2-2

3-1. Example of Printer Output of a Program 3-9

5-1. Elements A and B Deck Structure 5-7

5-2. Linker Input 5-8

5-3. Header Processing 5-10

5-4. ESID Processing for Element A 5-11

5-5. ESID Processing for Element B 5-12

5-6. Linker Input Deck Sequence for Two-Pass Operation 5-14

5-7. Linker Input Deck Sequence for One-Pass Operation 5-15

A-1. Schematic of Preassembly Macro Pass Operation A-1

TABLES

2–1. Instruction Mnemonics 2–7

2–2. Symbols Used in Describing Operand Formats 2–8

2–3. Operand Specifications Using Implied Base Register and Length Notation 2–11

2–4. Characteristics of the Various Constants 2–14

3–1. Internal Code 3–15

B–1. UNIVAC 1001 Card Controller IOCS Initial Error Indications B–33

B–2. UNIVAC 1001 Card Controller IOCS Requested Error Indications B–34

1. INTRODUCTION

1.1. GENERAL

Use of this manual presupposes a familiarity with the instruction repertoire and instruction and data formats of the UNIVAC 9200/9300 as described in "UNIVAC 9200/9300 Systems Central Processor and Peripherals Programmers Reference," UP-7546 (current version).

1.2. THE PURPOSE OF AN ASSEMBLER

An Assembler is one result of the many and continuing efforts to improve communications between computers and computer users. The general direction of these efforts has been towards an intermediate language which is close to the language of the user and which relies heavily on the computer for translation into its language.

In an Assembler language all coding is represented in the form of statements which are understandable to the programmer. The Assembler then converts these statements into a binary form which is understandable to the computer. The programmer's statements, when keypunched, are called source code. The Assembler converts the source code into object code. Figure 1–1 shows the general flow of source-to-object code conversion with an Assembler.

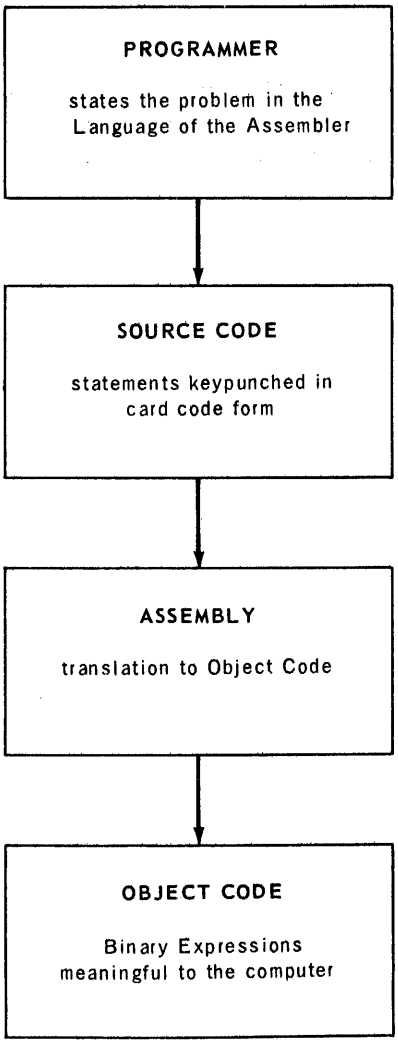


Figure 1–1. Source-to-Object Code Translation with Assembler

1.3. CARD ASSEMBLER FOR THE UNIVAC 9200/9300

The Card Assembler for the UNIVAC 9200/9300 System is an efficient, easy-to-use software aid that satisfactorily handles most of the programming problems encountered by the user. Each machine instruction and data form have simple, convenient representations in the assembly language. The rules which govern the use of the language are not complex; they may be learned quickly and applied easily.

A program in Card Assembler language for the UNIVAC 9200/9300 is written on a standard Univac coding form. The information on the form is keypunched, and the resulting source deck is read twice by the Assembler. Output cards, or an object deck, are produced by the Assembler in relocatable object code or absolute object code. The object deck is ready for loading into the UNIVAC 9200/9300 by means of the Card Program Loader routine. The basic flow of the UNIVAC 9200/9300 Card Assembler and associated software is shown in Figure 1–2. Input to the Assembler is a card deck keypunched from an Assembler coding form or is the output from the Preassembly Macro Pass.

The macro library is in macro code. Parameters are established for the macros by means of macro instructions. The Preassembly Macro Pass (described in Appendix A) converts the macro code into source code in preparation for assembly.

The assembly operation is a conventional two-pass procedure which produces a card deck in relocatable object code. The outputs of several separate assemblies may be combined by means of a Linker. The Linker output is in absolute object code. When a program is ready to be run, the relocatable or absolute object deck is loaded by a Card Program Loader subroutine.

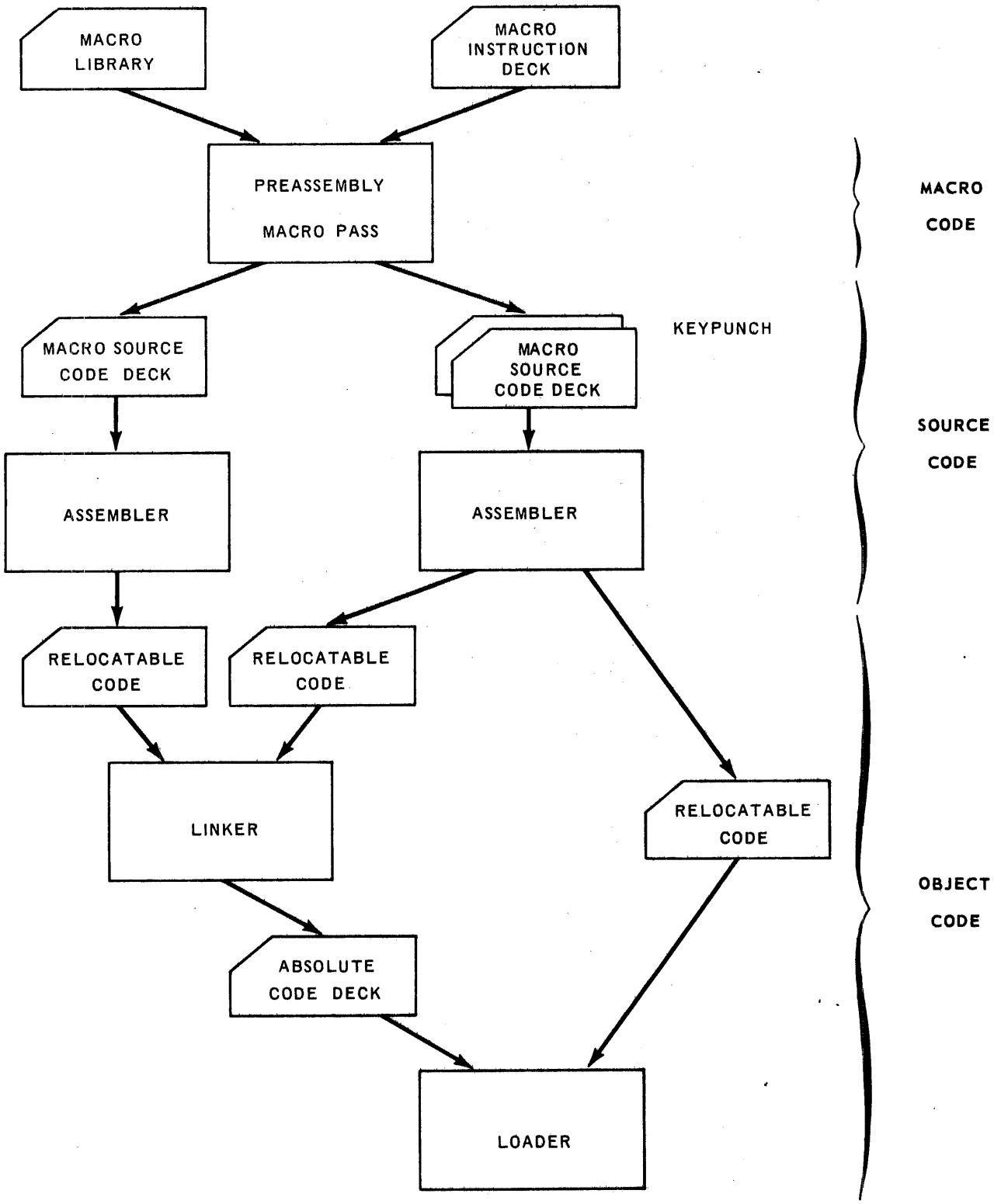


Figure 1–2. UNIVAC 9200/9300 Assembly System

1.4. ASSEMBLY LANGUAGE CHARACTERISTICS

The succeeding sections of this manual describe in detail the use of the Assembler coding form and the operational characteristics of the Assembler. These characteristics are summarized briefly as follows:

Mnemonic Operation Codes – A fixed name, consisting of two, three, or four letters, is assigned to each machine instruction. The name is chosen to suggest the nature of the instruction, thereby helping the user to learn and remember the instruction.

Symbolic Addressing and Automatic Storage Assignment – Symbolic labels may be assigned to instructions or groups of data. An instruction may then reference the labeled data by label rather than by storage address. In many cases, other data required by the instruction (such as operand length) may be supplied automatically by the Assembler. Another major task of the Assembler is to keep track of all storage locations used and to assign all incoming instructions and data to specific locations. The Assembler also handles all base register and displacement calculations.

Flexible Data Representation – Data may be represented in the Assembler in decimal, hexadecimal, or character notation, thus allowing the programmer to choose the most suitable form for each constant.

Relocatable Programs and Program Linking – Programs are prepared by the Assembler in an absolute or relocatable form. In relocatable form, the actual storage locations to be occupied by a program need not be specified at assembly time, or if specified, they may easily be altered before loading. Provisions are made for linking together, loading, and running as one program the results of separate assemblies, thereby reducing the machine time required to make changes to one part of a program.

Program Listing – One of the outputs of the Assembler is a printed listing of source and object codes. This listing includes flags marking any errors detected by the Assembler. Source code errors do not cause the Assembler to stall. The Assembler continues to process the rest of the source code performing its usual error checks, thus minimizing the number of assemblies required to produce error-free code.

2. THE ASSEMBLER LANGUAGE

2.1. CHARACTER SET

The character set used in writing statements in the Assembler language consists of:

Letters	A, B, C, . . . , Z
Digits	0, 1, 2, . . . , 9
Special Symbols	* + - , ( ) ' blank

2.2. STATEMENT FORMAT

Statements in the Assembler language are written on a standard coding form. Information for the Assembler and comments are written in columns 1 through 71. Column 72 must be blank. Columns 73 through 80 may contain program identification and sequencing information. The information in columns 1 through 71 consists of the following fields.

2.2.1. Label Field

The label field begins in column 1 and is terminated by a blank column. There may be no embedded blanks. The field may either be blank or contain a symbol whose value is to be defined. More detailed information about symbols is contained under headings 2.3.6 and 3.1.1.

2.2.2. Operation Field

The operation field begins with the first nonblank after the label field and is terminated by a blank. It contains either the name of an assembler directive or the mnemonic operation code for a machine instruction.

2.2.3. Operand Field

The operand field begins with the first nonblank after the operation field and is terminated by a blank not contained in a character representation. This field contains information which defines the operands of a machine instruction or which supplies the specifications required with an assembler directive.

2.2.4. Comments Field

The comments field begins with the column after the blank that terminates the operand field and ends at column 71. It may contain any combination of characters including blanks. It is not processed by the Assembler other than including it on the assembly listing. It may contain remarks to clarify the purpose or operation of the associated coding. A line may consist entirely of comments from columns 2 through 71 if column 1 contains an asterisk.

	LABEL (1)	ƀ OPERATION ƀ (10)	OPERAND (16) ƀ
1.	*THIS IS A COMMENT LINE.		
2.	TAG	BAL	15,TAG2
3.	LH 15,TAG3 THE OPERATION CODE IS LH		
4.		LH	15,TAG3

Figure 2–1. Example of Source Code Statements

Although the assembler language is free form, it is recommended that source code statements be written with the first character of the operation code in column 10 and the first character of the operand field in column 16. Tabulating the statements in this fashion creates a program listing which is neater in appearance and easier to read. The standard coding form is ruled to conform to this convention. Thus, although the statements on lines 3 and 4 of Figure 2–1 are equivalent to the Assembler, the form of line 4 is preferred to that of line 3.

The Assembler ignores the presence of any blank cards in the source code deck.

2.3. EXPRESSIONS

The operand field of a statement in the assembler language ordinarily consists of one or more expressions. Expressions may be grouped by parentheses and are separated by commas. For example, the basic operand formats for computer instructions are shown in Table 2–3. In this table, each subscripted letter represents an expression. An expression may be a single term or a number of terms connected by operators. The permissible operators are a plus sign (+) representing addition and a minus sign (–) representing subtraction. A leading minus sign is also allowed to produce the negative of the first term. All operations are performed in two's-complement binary notation. A term may be one of the following:

A decimal, hexadecimal, or character representation of an actual value.

A location counter reference.

A symbol.

2.3.1. Decimal Representation

A value may be represented directly by a string of up to five digits, 0 through 9, forming a decimal number from 0 through 32767. Such a number is converted to a binary value occupying one or two bytes depending on the type of field for which it is intended. Following are some decimal representations.

Decimal Representation	Binary Value
0	00000000
13	00001101
257	00000001 00000001
32767	01111111 11111111

2.3.2. Hexadecimal Representation

A hexadecimal representation consists of a string of digits preceded by X' and followed by ' (apostrophe). Each hexadecimal digit represents a half byte of information. The hexadecimal digits and their values are:

0 – 0000	8 – 1000
1 – 0001	9 – 1001
2 – 0010	A – 1010
3 – 0011	B – 1011
4 – 0100	C – 1100
5 – 0101	D – 1101
6 – 0110	E – 1110
7 – 0111	F – 1111

Some examples of hexadecimal representations and their values are:

Hexadecimal Representation	Binary Value
X'D'	00001101
X'101'	00000001 00000001
X'7FFF'	01111111 11111111

2.3.3. Character Representation

A character representation consists of a string of characters preceded by C' and followed by '. The following are valid character representations.

Character Representation	EBCDIC Value
C'D'	11000100
C'GROSS'	1100011111011001110101101110001011100010
C'9'	11111001

In a character representation, an apostrophe is represented by two successive apostrophes, and an ampersand by two successive ampersands.

In an expression, a self-defining term in character representation can be a maximum of one character in length.

2.3.4. Location Counter

An indication of the next location available for assignment is maintained as a counter called the location counter. After the Assembler processes an instruction or constant, it adds the length of the instruction or constant processed to the location counter.

Each instruction or address constant must have an address which is a multiple of two. Such an address is said to fall on a halfword boundary. If the value of the location counter is not a multiple of two when assembling such a constant or an instruction, a one is added to the location counter before assigning an address to the current line. Storage locations reserved by this process receive binary zeros when the program is loaded.

The current value of the location counter is available for reference in the Assembler language and is represented by the single special character * (asterisk). If written in a constant representation or in an instruction operand expression, this symbol is replaced by the storage address of the leftmost byte allocated to that instruction or constant. Thus the instruction

```
BC          15,*
```

represents a one-instruction loop.

2.3.5. Relative Addressing

An instruction may address data in its immediate vicinity in storage in terms of its own storage address. This is called relative addressing and is achieved by an expression of the form *+n or *-n where n is the difference in storage addresses of the referring instruction and the instruction or constant being accessed. Relative addressing is always in terms of bytes, not words or instructions. For example, in the coding

```
LABEL     OPERATION     OPERAND
1         10            16
          CH            15,LMIT
          BC            7,*+12
          AH            15,TWO
          BC            15,*-12
          MVC           A,B
```

the address *+12 in the second line is the address of the instruction in the last line and the address *-12 in the fourth line is the address of the instruction in the first line since each of the first four instructions is four bytes long.

2.3.6. Symbols

A symbol is a group of up to four alphanumeric characters. The first, or leftmost, must be alphabetic. Special characters or blanks may not be contained within a symbol. The following are examples of valid symbols:

A	LOSS
A72Z	PRFT
CAT	

The following are not valid symbols for the reasons stated:

GROSS	More than four characters
N PA	Embedded blank
SR)N	Special character

A symbol may be assigned any value from 0 through 32767. It is assigned a value, or defined, when it appears in the label field of any source code statement other than a comment. A symbol appearing in the label field of an EQU or ORG directive is assigned the value of the expression in the operand field. In all other cases the value assigned is the current value of the location counter after adjustment to a halfword boundary, if necessary. The value is assigned to the current label before the location counter is incremented for the next instruction, constant, or storage definition. Thus, if a symbol appears in the label field of a statement defining an instruction, constant, or storage area, the symbol is assigned a value equal to the storage area address of that instruction, constant, or storage area.

2.3.7. Relocatable and Absolute Expressions

A single term may be either relocatable or absolute. Decimal, character, and hexadecimal representations are all absolute terms. A location counter reference within a section of relocatable code yields a relocatable value. If a symbol is defined by appearing in the label field of a source code statement within a section of relocatable code, its value will be relocatable.

An expression is relocatable in the following cases: if it consists of an absolute expression plus a relocatable term; if it can be reordered to have that form; or if it consists solely of a relocatable term. Some examples of relocatable expressions are:

R

A+R

R+A

R−R+A+R

where R represents a relocatable term and A an absolute term.

An expression is absolute if all of the terms in the expression are absolute or if it consists only of absolute terms plus an even number of relocatable terms of which exactly half are preceded by minus signs. Some examples of absolute expressions are:

A

A+A−A

A−A+A+A

R+A−R

R−R+A

An expression may be negatively relocatable under certain circumstances (see 2.5.1). Such an expression consists of an absolute expression minus a relocatable expression, or an expression which may be reordered to that form. Some examples are as follows:

A−R A−R−R+R R−R+A−R

2.3.8. Length Attribute

The Assembler associates a length attribute with a symbol defined in the label field of a source code line representing an instruction, constant, or storage definition. The length attribute of such a symbol is the number of bytes assigned to the instruction, constant, or storage area involved. The length attribute of an expression is also determined by the Assembler and is a function of the leading term of the expression. If the first term of an expression is an absolute value, a length attribute of one byte is assigned to the expression. If the leading term is a symbol, the number of bytes attributed to the expression is the same as the length attributed to the symbol. Thus, if TAG appears in the label field of an LH instruction (Load Halfword), it would have a length attribute of 4 since LH is a 4-byte instruction. In referencing the same label, the expression TAG+195 also has a length attribute of 4; but the expression 195+TAG has a length attribute of 1 because the leading term is a constant.

When a location counter reference appears as the first term of an expression, its length attribute is defined as having either the length of the instruction in which it appears or as a length attribute of 1 (when the reference to the location counter occurs in an EQU statement).

TAG	EQU*	The length attribute for TAG is 1
MVC	*+12,ABC	This instruction will move 6 bytes from ABC to *+12.

2.4. MACHINE INSTRUCTIONS

A list of the standard machine instructions giving the numeric and hexadecimal operation codes with the instruction type is shown in Table 2−1.

The machine instruction format consists of a label (optional), a mnemonic operation code, and an operand. If a symbol is used in the label field of a machine instruction, it is assigned the address of the leftmost character of the instruction and receives a length attribute equal to the length of that instruction. There are four types of instruction formats. These are shown below together with a brief explanation of the functions performed by the instructions within each format type. Table 2−2 defines the symbols used in the instruction type formats.

MNEMONIC	FUNCTION	HEXADECIMAL OPERATION CODE	FORMAT
AH	ADD HALFWORD	AA	RX
AI	ADD IMMEDIATE	A6	SI
AP	ADD (PACKED) DECIMAL	FA	SS2
BAL	BRANCH AND LINK	45	RX
BC	BRANCH ON CONDITION	47	RX
CH	COMPARE HALFWORD	49	RX
CLC	COMPARE LOGICAL CHARACTER	D5	SS1
CLI	COMPARE LOGICAL IMMEDIATE	95	SI
CP	COMPARE (PACKED) DECIMAL	F9	SS2
DP	DIVIDE (PACKED) DECIMAL	FD	SS2
ED	EDIT	DE	SS1
HPR	HALT AND PROCEED	A9	SI
LH	LOAD HALFWORD	48	RX
LPSC	LOAD PROGRAM STATE CONTROL	A8	SI
MP	MULTIPLY (PACKED) DECIMAL	FC	SS2
MVC	MOVE CHARACTERS	D2	SS1
MVI	MOVE IMMEDIATE DATA	92	SI
MVN	MOVE NUMERICS	D1	SS1
MVO	MOVE WITH OFFSET	F1	SS2
NC	*AND* CHARACTERS	D4	SS1
NI	*AND* IMMEDIATE DATA	94	SI
OC	*OR* CHARACTERS	D6	SS1
OI	*OR* IMMEDIATE DATA	96	SI
PACK	PACK	F2	SS2
SH	SUBTRACT HALFWORD	AB	RX
SP	SUBTRACT (PACKED) DECIMAL	FB	SS2
SPSC	STORE PROGRAM STATE CONTROL	A0	SI
SRC	SUPERVISOR REQUEST	A1	SI
STH	STORE HALFWORD	40	RX
TIO	TEST I/O	A5	SI
TM	TEST UNDER MASK	91	SI
TR	TRANSLATE	DC	SS1
UNPK	UNPACK	F3	SS2
XIOF	EXECUTE INPUT/OUTPUT FUNCTION	A4	SI
ZAP	ZERO ADD (PACKED) DECIMAL	F8	SS2

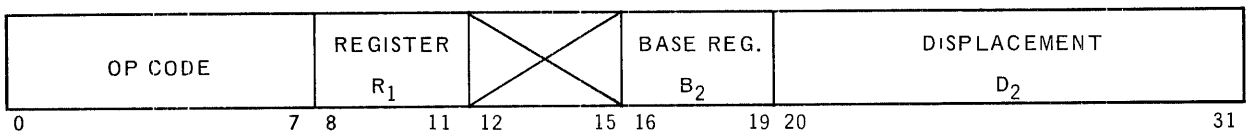
Table 2–1. Instruction Mnemonics

SYMBOL	MEANING
R_1	The number of the register addressed as operand 1
I_2	The immediate data or device address used as operand 2 of an SI instruction.
L	The length of the operands *
L_i	The length of operand i *
S_i	The storage address of operand i
B_i	The base register for operand i
D_i	The displacement for operand i

* *This is the true length of the operand, not the length less one, as required in object code. The Assembler makes the necessary reduction of one in the length when converting source to object code.*

Table 2–2. Symbols Used in Describing Operand Formats

2.4.1. RX – Register to Storage Instructions



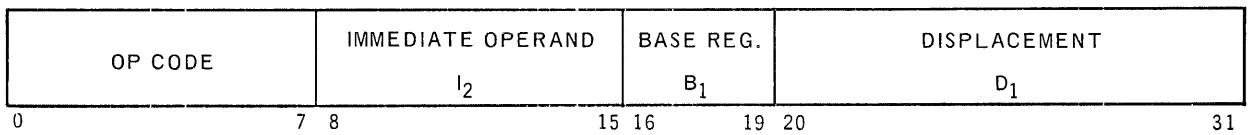
RX

Complete Operands Form: $R_1,D_2(,B_2)$ 4-byte instruction

In general, instructions in this format are used to process data between registers and storage and include such functions as load, store, compare, add, subtract, and branch. The mnemonic codes for instructions using this type of format are:

AH	Add Halfword
BAL	Branch and Link
BC	Branch on Condition
CH	Compare Halfword
LH	Load Halfword
SH	Subtract Halfword
STH	Store Halfword

2.4.2. SI – Instruction to Storage Instructions



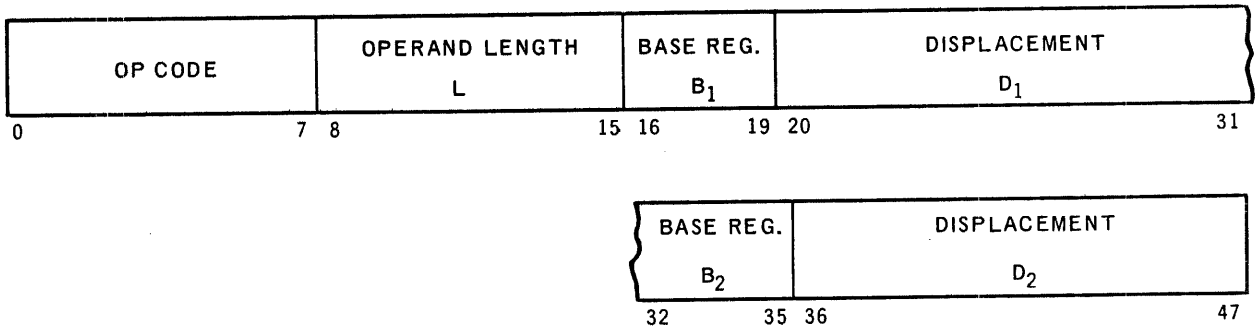
SI

Complete Operands Form: $D_1(B_1),I_2$ 4-byte instruction

In general, instructions with this format are used for processing with control data contained in the instruction. The mnemonic codes for instructions using this type of format are:

AI	Add Immediate
CLI	Compare Logical Immediate
HPR	Halt and Proceed
LPSC	Load Program State Control
MVI	Move Immediate Data
NI	AND Immediate Data
OI	OR Immediate Data
SPSC	Store Program State Control
SRC	Supervisor Request
TIO	Test I/O
TM	Test Under Mask
XIOF	Execute I/O Function

2.4.3. SS1 – Storage to Storage Instructions



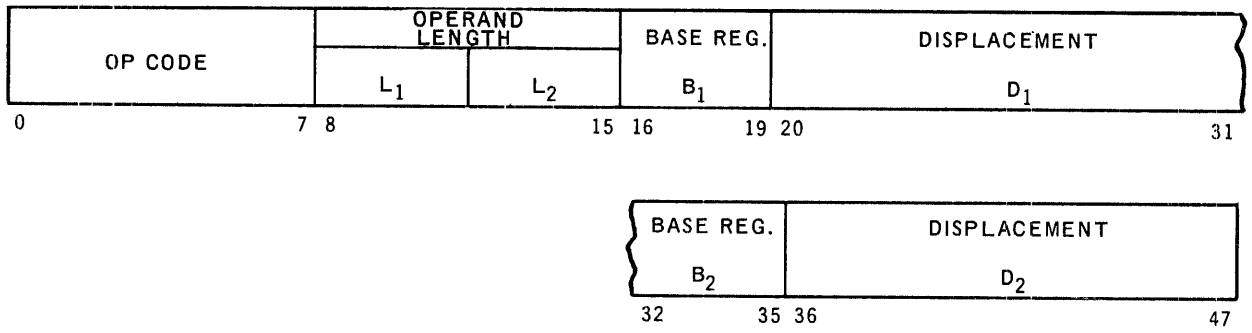
Complete Operands Form: $D_1(L,B_1),D_2(B_2)$

6-byte instruction

The instructions in this format are used to process data in storage when the operands are of equal length, and include such functions as comparisons, transfers, translations, and logical operations. The mnemonic codes for instructions using this type of format are:

CLC	Compare Logical Character
ED	Edit
MVC	Move Characters
MVN	Move Numerics
NC	AND Characters
OC	OR Characters
TR	Translate

2.4.4. SS2 – Storage to Storage Instructions



SS2

Complete Operands Form: $D_1(L_1,B_1),D_2(L_2,B_2)$

6-byte instruction

The instructions in this format are used to process operands of unequal length and to process packed decimal values. The various functions include decimal operations (add, subtract, compare), shift operations, and pack and unpack operations.

The mnemonic codes for instructions using this type of format are:

AP	Add Packed Decimal
CP	Compare Packed Decimal
DP	Divide Packed Decimal
MP	Multiply Packed Decimal
MVO	Move With Offset
PACK	Pack
SP	Subtract Packed Decimal
UNPK	Unpack
ZAP	Zero Add Packed Decimal

2.4.5. Implied Base Register and Length

Where an operand is described in terms of a storage address and a length, the expression used may be simplified from that shown in the instruction format by implying the base register and the length. Information supplied in the USING and DROP directives enable the Assembler to separate a storage address into a base register and a displacement. If a length attribute is associated with the expression but is not specified in the statement, a value equal to the length of the operand is supplied by the Assembler. Table 2–3 lists the complete specification for the operand referencing memory, applicable instruction types, and the operand format as it may be written utilizing an implicit base register and/or length representations.

APPLICABLE INSTRUCTION TYPES	COMPLETE SPECIFICATION FOR ONE OPERAND	OPERAND SPECIFICATION USING		
		IMPLIED BASE REGISTER NOTATION	IMPLIED LENGTH	IMPLIED BASE REGISTER AND LENGTH
RX	$D_2(,B_2)$	S_2	NA	NA
SI	$D_1(B_1)$	S_1	NA	NA
SS1	$D_1(L,B_1)$	$S_1(L)$	$D_1(,B_1)$	S_1
SS1	$D_2(B_2)$	S_2	NA	NA
SS2	$D_1(L_1,B_1)$	$S_1(L_1)$	$D_1(,B_1)$	S_1
SS2	$D_2(L_2,B_2)$	$S_2(L_2)$	$D_2(,B_2)$	S_2

Table 2–3. Operand Specifications Using Implied Base Register and Length Notation

Example: To move 80 characters from the field labeled OPA defined as a 90-character field to the field labeled OPB and defined as an 80-character field, the instruction could be written as follows:

```
MVC         OPB,OPA
```

The length attribute of OPB is implied.

If 90 characters were to be moved, the instruction would be written

```
MVC         OPB(90),OPA
```

2.5. DATA AND STORAGE FORMATS

The formats for data and storage statements are similar to those for a machine instruction. A symbol may be used in the label field. It is assigned the address of the leftmost character of the constant or storage area being specified and is attributed with a length equal to that of the specified constant or storage area. The operation code is either DC (Define Constant) or DS (Define Storage). The operand has various formats which are explained below.

2.5.1. DC – Define Constant

There are three types of constants: C for character representation; X for hexadecimal; and Y for expression. To define a constant, the assembly directive DC is written in the operation field. The statement has the form:

LABEL	OPERATION CODE	OPERAND
Symbol	DC	tLn'c'

or

LABEL	OPERATION CODE	OPERAND
Symbol	DC	Y(e)

or

LABEL	OPERATION CODE	OPERAND
Symbol	DC	YL1(e)

LABEL	OPERATION CODE	OPERAND
Symbol	DC	YL2(e)

where: n is a decimal number ≤ 16 specifying the number of bytes the constant is to occupy,

t is X or C denoting hexadecimal or character representation, respectively,

c is the actual character or hexadecimal representation for the constant, and

e is any acceptable expression as previously defined.

2.5.1.1. Character Representation

A character representation is a string of as many as 16 characters, including blanks, enclosed by apostrophe marks. The apostrophe mark itself is represented by two successive apostrophes and an ampersand by two successive ampersands. In each of these cases the two characters count only as one towards the limit of 16. Thus, to represent a character constant of 16 apostrophes, 32 successive apostrophes would be written, preceded by and ended with an apostrophe. The length specification may be omitted, in which case the length of the constant is determined implicitly from the number of characters between the apostrophe marks. If the number of characters in apostrophes is greater than the length n, the rightmost characters are truncated to fit the field in the area reserved for it. If the number of characters between apostrophes is less than the length, the value is padded with blanks on the right to fill the field.

For example, the following lines each result in a two-byte constant consisting of the letter A followed by blank. The third representation is flagged with an error indication.

```
LABEL        OPERATION        OPERAND
1            10               16
             DC               CL2'A'
             DC               C'A'
             DC               CL2'AB'
```

2.5.1.2. Hexadecimal Representation

A hexadecimal representation is a string of as many as 32 hexadecimal digits enclosed by apostrophe marks. If the digit string is less than twice the length specification, the field is padded with hexadecimal zeros on the left. If more than twice the length specification, the representation is truncated on the left to produce a value equal to the length. The length specification may be omitted, in which case the length of the constant is determined as the smallest number of bytes which will contain the constant specified. If necessary, the field is padded on the left with one hexadecimal zero.

The following illustrates the values of source statements which represent valid hexadecimal constants, three bytes in length:

CONSTANT REPRESENTATION	VALUE		
DC XL3'1'	00000000	00000000	00000001
DC X'123A5'	00000001	00100011	10100101
DC X'1F3456'	00011111	00110100	01010110

2.5.1.3. Expression Constants

Constants of type Y provide a way to write a constant involving a relocatable expression. If the length specification L1 is not present, the expression defining an expression constant may have any value from −32,768 to 32,767 inclusive and may be absolute, relocatable, or negatively relocatable. (A negatively relocatable expression consists of an absolute expression minus a relocatable expression, or an expression that can be reordered to that form.) An expression constant in which the length specification L1 or L2 is not present provides a convenient notation for representing a complete storage address. It is for this reason that constants of this type are called address constants.

An address constant always occupies two bytes of storage and location counter adjustment to a halfword boundary is performed by the Assembler before storage locations are assigned to the constant. No such adjustment is performed for hexadecimal or character constants.

For example, an address constant designed to generate the address assigned to the label 'TAG' would take the following form.

```
DC      Y(TAG)
```

An expression constant in which the length specification L1 is present may have any value from 0 through 255 and may be absolute, relocatable, or negatively relocatable. It always occupies one byte of storage, and no location counter adjustment is made before assigning a memory location to the constant. It is useful when an externally defined symbol is assigned to only one byte.

An expression constant in which the length specification L2 is present is the same as an expression constant in which no length specification is present, except in the former case, no halfword boundary adjustment is made.

A summary of constant types, lengths, padding and truncation rules appears in Table 2–4.

CONSTANT TYPE	EXPLICIT LENGTH	IMPLICIT LENGTH	TRUNCATION OR PADDING
C	variable 1–16	maximum 16	on right side
X	variable 1–16	maximum 16	on left side
Y	not stated	2	on left side
	1	none	on left side
	2	none	on left side

Table 2–4. Characteristics of the Various Constants

2.5.2. DS – Define Storage

The format of the assembler language statement to reserve storage is as follows:

LABEL	OPERATION CODE	OPERAND
Symbol (Optional)	DS	dCLn

or

LABEL	OPERATION CODE	OPERAND
Symbol (Optional)	DS	dH

where: d is a non-negative integer called the duplication factor, the number of fields to be reserved (d may be a maximum of 256),

n is a decimal number representing the length of the field to be reserved (n may be a maximum of 256 and a minimum of one),

H represents a field whose length is two bytes and whose storage address must be on a halfword boundary.

The statement DS 0H causes the location counter to be adjusted to a multiple of two without reserving storage. A duplication factor of zero may be used with any storage definition statement to define the address and length of a field without reserving storage for it. The duplication factor may be omitted, in which case a factor of one is assumed.

Thus:

```
CARD    DS    0CL80
FRST    DS    CL40
LAST    DS    CL40
```

would define an 80-byte field named CARD, a 40-byte field named FRST whose address is the same as that of CARD, and a field named LAST whose length is 40 bytes and whose address is 40 greater than that of CARD and FRST.

The location counter is not increased in assembling CARD (because duplication factor is 0) but is with FRST and LAST. Therefore, 40 + 40 = 80 spaces are reserved, with FRST and CARD assigned the starting location and LAST assigned the midpoint. When the duplication factor is specified, it defines the number of fields of length n (for C) or the number of pairs of bytes (for H) to be reserved. For example,

```
TAG     DS    13H
```

reserves 13 pairs of bytes. The symbol, TAG, refers to the first pair of bytes only and not to the entire 26 bytes. TAG would have a length attribute of two in this instance. For example,

```
TAG1    DS    10CL10
```

reserves 10 groups of 10 bytes each, or 100 bytes. The symbol TAG1 refers to the first group of ten bytes and not to the entire 100 bytes. In this instance TAG1 would have a length attribute of ten.

3. ASSEMBLER DIRECTIVES AND SYSTEM CODES

3.1. DIRECTIVES

In addition to the representation of machine instructions, constants, and storage, the Assembler language includes several assembler directives. These are instructions to the Assembler to perform certain functions and provide the user of the Assembler language with control of the operation of the Assembler.

The assembler directives, grouped by function, are as follows:

Symbol Definition
- EQU

Assembler Control
- START
- END
- ORG

Base Register Assignment
- USING
- DROP

Program Linking
- ENTRY
- EXTRN

Assembler directives, except START, may use a symbol in the operand field, and, with the exception of ENTRY, EXTRN, USING and DROP, the symbol must have appeared in the label field of a previous statement.

3.1.1. Symbol Definition

EQU – Equate

The value and length attribute of a symbol may be defined explicitly. The statement to accomplish this has the following form:

LABEL	OPERATION	OPERAND
Symbol	EQU	e_1, e_2

where: e_1 and e_2 are expressions and must have been previously defined.

The symbol is defined to have a length attribute equal to the value of the second expression in the operand. The second expression in the operand may be omitted, in which case the symbol is defined to have the length attribute of the first expression.

The symbol in the label field is defined to have the value of the first expression in the operand field. If the value of the first expression in the operand field is not between 0 and 32767, the statement will be flagged with an error indication and the symbol will remain undefined.

Thus, if the value of the location counter is 2000 when the following lines are encountered,

LABEL (1)	OPERATION (10)	OPERAND (16)
TAG	DS	25CL10
HIDE	EQU	100+TAG,150
SEEK	EQU	TAG+270-*

TAG has a relocatable value of 2000 and a length attribute of 10.

HIDE has a relocatable value of 2100 and a length attribute of 150.

SEEK has an absolute value of 20, and a length attribute of 10.

3.1.2. Assembly Control

Assembler directives are available to control the program name and initial location, alter the location counter in a specified manner, and indicate the end of the program statement and the instruction with which execution of the object program is to begin.

3.1.2.1. START – Program Start

The START directive defines the program name and tentative starting location. It must precede all other program statements in the source code deck except comments. The format of the START directive is as follows:

LABEL	OPERATION	OPERAND
Symbol	START	Decimal or Hexadecimal representation

The expression in the operand field is evaluated and incremented if necessary to make it a multiple of four. The result becomes the initial setting of the location counter and is the value of the symbol in the label field. This symbol becomes the Program IDentification (PID) and is available as an entry point without being separately defined as such (see 3.1.4). Although the operand of the START directive is an absolute value, it is treated as relocatable.

Thus the value of the location counter and the coding which follows a START directive are both relocatable. Any one of the statements below would result in the program having the name SORT, being assigned to locations starting at 1068, and having the symbol SORT defined with the relocatable value 1068.

```
SORT    START   1065
SORT    START   1068
SORT    START   X'42C'
```

A START directive preceded by one or more statements other than comments is ignored and flagged as an error. A START directive whose operand field does not have a value from 0 to 32764 is ignored and flagged as an error. If there is no valid START directive, the program name is left blank and the location counter is set to 0.

3.1.2.2. END – Program End

The END directive indicates to the Assembler the end of the program being assembled. The format of the END directive is

LABEL	OPERATION	OPERAND
Symbol (optional)	END	Expression (optional)

With an END directive the Assembler stops reading cards, punches any remaining data which has accumulated, and then punches a Transfer Card. If the operand field of the END directive contains an expression, this expression is punched into the Transfer Card to signify to the load routine the address at which to begin program execution. If there is no expression in the operand field of the END directive, the corresponding field of the Transfer Card is blank. In that case when the load routine encounters the Transfer Card, it transfers control to the first location loaded.

If a symbol appears in the label field of the END directive, it is assigned the current value of the location counter. This is normally one greater than the highest address assigned to the program being assembled.

3.1.2.3. ORG – Set Location Counter

The ORG directive is used to set the location counter to a specified value. The format of the ORG directive is as follows:

LABEL	OPERATION	OPERAND
Symbol (optional)	ORG	e_1, e_2

The value to which the location counter is set is determined by the values of the expressions in the operand field. If e_2 is not specified, then the location counter is set to the value of e_1. If e_2 is expressed, the location counter is set to the next value greater than or equal to the value of e_1 which is a multiple of e_2. Examples follow:

OPERAND	RESULTING LOCATION COUNTER VALUE
1000	1000
1000,2	1000
1000,16	1008

The value of e_2 must be a power of two.

If a symbol appears in the label field, its value is also the value to which the location counter is set and the symbol is assigned a length attribute of one. The value must be either an absolute value between 0 and 32767 or a relocatable value between the initial location counter setting and 32768. If the value does not lie within this range, the ORG directive is ignored and the line is flagged with an error indication. With the ORG directive it is possible to set the location counter to a value which is not a halfword boundary.

The ORG directive to set the location counter to a value 603 less than its current setting would be as follows:

LABEL	OPERATION	OPERAND
	ORG	*–603

The ORG directive may be used to reserve a number of locations which are not expressed as a single decimal integer. For example, to reserve A minus B bytes of storage where A and B are previously defined symbols, the statement is written as follows:

LABEL	OPERATION	OPERAND
	ORG	*+A–B

Bytes of storage reserved either with a DS or ORG directive are *not* set to zero when the program is loaded.

If e_1 is a relocatable expression, the value to which the location counter is set and the coding that follows the ORG directive are both relocatable. If absolute, the value to which the location counter is set and the coding that follows the ORG directive are both absolute.

3.1.3. Base Register Assignment

The Assembler assumes the responsibility for converting storage addresses to base register and displacement values for insertion into instructions being assembled. To do this the Assembler must be informed of the available registers and the values assumed to be in those registers. The assembly directives USING and DROP are available for this purpose.

3.1.3.1. USING – Assign Base Register

The USING directive informs the Assembler that a specified register is available for base register assignment and that it contains a specified value. The format of the USING directive is as follows:

LABEL	OPERATION	OPERAND
Symbol (optional)	USING	R,A

where: R is a relocatable expression and A is an absolute expression.

It is also possible to specify an absolute value for the first expression in the operand of a USING directive.

The first expression represents the value the Assembler assumes is in the specified register at object time. The second expression in the operand field must be a number from 8 through 15 which denotes the general register specified.

3.1.3.2. DROP – Unassign Base Register

The format of the DROP directive is

LABEL	OPERATION	OPERAND
Symbol (optional)	DROP	Absolute expression

This directive informs the Assembler that the specified base register no longer contains a value available to the Assembler for computing base register and displacement values. The expression in the operand field of the DROP directive is a number from 8 through 15 which denotes the general register no longer available.

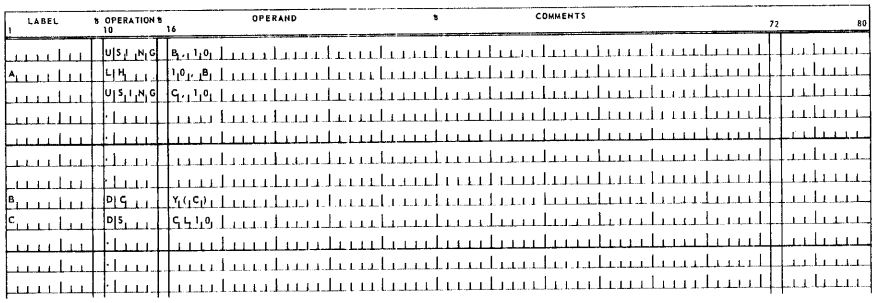
3.1.3.3. Function of USING and DROP Directives

The Assembler maintains a table of the available registers and the values they contain at object time. This table is referred to as the USING table. A USING directive adds a register and value to the USING table or revises the value for a register already in the table. A DROP directive removes a register and its associated value from the table. If the operands of a USING or DROP directive are not valid, the line is flagged with an error indication.

If an operand address is given as a relative address instead of as a base register and displacement specification, the Assembler searches the USING table for a value yielding a valid displacement, that is, a displacement of 4095 or less. If there is more than one such value, that value which yields the smallest displacement is chosen. If no value yields a valid displacement, the operand address is set to zero and the line is flagged with an error indication. If more than one register contains the value yielding the smallest displacement, the highest numbered register is selected.

An absolute address with no base register indicated is treated as an absolute, direct address.

The placement of a USING directive determines the instructions whose operand addresses may be decomposed based on that USING statement. The first operand of the USING statement determines the portion of the program which may be addressed using the specified register. Thus, if a program contains the coding



the B2 and D2 fields of the instruction labeled A will contain 10 and 0, respectively. Moreover, if the program contains no USING directives for register 10 other than the ones shown, then the second line labeled A is the only line in the program for which the Assembler would consider 10 as a register available for addressing the line labeled B.

The load routine stores in register 13 the starting address of the program just loaded. All other registers must be loaded by the program itself in a manner consistent with the information given to the Assembler in the USING directives. The following example shows how this is done.

LABEL	OPERATION	OPERAND
	USING	A,13
A	LH	12,B
	USING	C,12
B	DC	Y(C)
C	DS	CL10
	END	A

Lines two and three of the above example exemplify the following general rule:

> The loading of a value into a general register must precede the USING directive which informs the Assembler the value is available.

It is also possible to specify an absolute value for the first expression in the operand of a USING directive. The entry in the USING table made in response to such a USING directive is not used to decompose relative addresses. It is used instead to decompose absolute addresses. For example, given the following coding

```
        USING   4000,15
A       LH      14,4096
```

B2 and D2 fields of the instruction labeled A will contain 15 and 96 respectively.

3.1.3.4. Direct Addressing

The machine instruction format provides for either base register and displacement addressing (indexed addressing) or direct addressing. Instructions using direct addressing have a faster execution time. To facilitate error checking by the Assembler, direct addressing is described to the Assembler in terms of the pseudo base registers 0, 1, 2, 3, 4, 5, 6, and 7 which contain the values 0, 4096, 8192, 12288, 16384, 20480, 24576 and 28672, respectively. Thus, the direct address 512 would be treated by the Assembler as an address consisting of a reference to the pseudo base register 0 and a displacement of 512. The address 4098 would yield a base of 1 and a displacement of 2. The additional forms of the USING directive which are available for direct addressing are, specifically

LABEL	OPERATION	OPERAND
	USING	*,0
	USING	*,1
	.	.
	.	.
	.	.
	.	.
	USING	*,7

The first line above makes direct addressing available for addresses in the range 0 to 4095. The second makes direct addressing available for addresses in the range 4096 to 8191, and so on. The DROP directive may also refer to the pseudo registers 0 through 7 to terminate direct addressing.

A program involving direct addressing may still be relocatable.

The asterisk (*) when used in the operand of the USING directive specifying a pseudo base register has a unique meaning and does not have the normal connotation of the current value of the location counter.

3.1.4. Program Linking

The Assembler provides, as part of its output, information which allows the results of separate assemblies to be linked together, loaded, and then executed as a single program. Proper sectioning reduces the machine time required to make changes to an existing program. If a change is required, only that part which is changed need be reassembled. The output is then linked with the remaining parts to produce the altered program. Proper sectioning of a program also reduces the number of symbols required in each of the separate assemblies.

A symbol defined in the label field of element A and addressed in element B is said to be externally defined in element A and referenced in element B. Thus, by using the ENTRY and EXTRN directives, proper linkage is supplied when the separate elements are assembled. This information is handed on to the Linker program by the External Definition Cards and the External Reference Card which are outputs of the Assembler.

3.1.4.1. ENTRY – Externally Defined Symbol Declaration

That portion of a program submitted as input to a single assembly is called an element. Each element must declare the symbols defined within that element and to which reference is made by other elements. Each symbol is referred to as being externally defined and is declared by the ENTRY directive. The ENTRY directive has the format

LABEL	OPERATION	OPERAND
	ENTRY	Symbol

The symbol in the operand field is declared to be externally defined. Its name and assigned value are included in the output of the Assembler as an External Definition Card.

3.1.4.2. EXTRN – Externally Referenced Symbol Declaration

The Assembler must also be informed of all symbols referred to in the element being assembled but which are defined in some other element. A reference to such a symbol is called an external reference, and such symbols are declared in the EXTRN directive. The format of the EXTRN directive is

LABEL	OPERATION	OPERAND
	EXTRN	Symbol

The symbol in the operand field is declared to be a symbol defined in some other element. A symbolic name and the External Symbol Indentification assigned by the Assembler are included as input of the Linker as an External Reference Card.

3.1.5. Assembler Program Listing

Figure 3–1 is a comprehensive example of coding in UNIVAC 9200/9300 Assembler language. The listing shown is a reproduction of an actual printout from the prototype UNIVAC 9200/9300 System. The example is coding for the self-loading memory dump routine described in "UNIVAC 9200/9300 Programming Utility Reference Manual," UP-4120 (current version).

```
0001                    *          THIS MEMORY DUMP ROUTINE IS ADJUSTED TO THE                MD 1010
0002                    *          MACHINE CONFIGURATION.                                     MD 1020
0003                    *     MDSLF POS=132,CH=63                                             MD 1030
0004                    *          THE FOLLOWING CODING IS EQUIVALENT TO THE SOURCE CODE      MD 1040
0005                    *          WHICH IS GENERATED BY THE ABOVE MACRO INSTRUCTION.         MD 1050
0006                    *          PARAMETER  EXPLANATION                                     MD 1060
0007                    *          PAR.1  POS = 96,120,OR 132 FOR PRINTER CH. POSITION.       MD 1070
0008                    *          PAR.2   CH = 63 OR 48 FOR AN ASSOCIATED PRINT BAR.         MD 1080
0009                    *          PAR.3  BGN = BLANK OR 128 THROUGH 32767 FOR THE            MD 1090
0010                    *                       LOCATION AT WHICH MEMORY DUMP IS TO BEGIN.    MD 1100
0011                    *          PAR.4  END = BLANK OR 128 THROUGH 32767 FOR THE            MD 1110
0012                    *                       LOCATION AT WHICH MEMORY DUMP IS TO END.      MD 1120
0013                    *          PAR.5  MEM = 8K,12K,16K,AND 32K, IF THE ENTIRE MEMORY      MD 1130
0014                    *                       IS TO BE DUMPED AND THE ROUTINE IS TO BE      MD 1140
0015                    *                       LOADED INTO THE HIGHEST MEMORY LOCATIONS.     MD 1150
0016                    *                       OTHERWISE BLANK.                              MD 1160
0017                    *          PAR.6 LOAD = 260 THROUGH 32190 FOR THE LOCATION            MD 1170
0018                    *                       AT WHICH MEMORY DUMP IS TO BE LOADED.         MD 1180
0019                    *                       BLANK IF MEM PARAMETER EXISTS.                MD 1190
0020                    *          PAR.7  RDR = BLANK OR 1001 FOR AN ASSOCIATED READER.       MD 1200
0021                    MD        START 0                                                     MD 2010
0022                              USING *,0                                                   MD 2011
0023                    000A      ORG   10                                                    MD 2012
0024                    000A 00000000000000000010 DC    XL10'1C'   LOADER SECTION 1ST CARD  N MD 2020
0025                    0014 00000000000000000000 DC    XL8'0'                               MD 2030
0026                    001C D201004600042 MVC   70(2),66   SET BASE ADDR FOR 2ND CARD  P,   MD 2040
0027                    0022 92500045      MVI   69,80     CON.1   SET 80 TO D.C. AREA  P   MD 2050
0028                    0026 A4010002      XIOF  2,1       READ CARD                    P   MD 2060
0029                    002A 4770003E      BC    7,62      IS XIOF ACCEPTED?            DY  MD 2070
0030                    002E A5010000      TIO   0,1       TEST I/O STATUS              P   MD 2080
0031                    0032 4720002E      BC    2,46      IS READER WORKING?           DN  MD 2090
0032                    0036 91C00000      TM    0,X'C0'   TEST STATUS BITS             P   MD 2100
0033                    003A 47800F70      BC    8,3952    IS THERE ANY ERROR?          DY  MD 2110
0034                    003E A90021C0      HPR   X'21C0',0  READER OFF NORMAL   RESTART H   MD 2120
0035                    0042 0F66          DC    Y(3942)   ADDRESS FOR 2ND CARD             MD 2130
```

Figure 3–1. Example of Printer Output of a Program

(Sheet 1 of 5)

```
0036  0044 000A0048                    DC    XL4'0A0048'          DEVICE CONTROL FOR 1ST CARD      MD  2140
0037  0F70                             ORG   3952                 LOADER SECTION  2ND CARD   N  MD  2150
0038  0F70 950C0FAF                    CLI   *+63,X'0C'           IS THIS A TYPE Y CARD?     DN  MD  2160
0039  0F74 47800FA0                    BC    8,*+44               IF Y CARD GO TO CON.4       C  MD  2170
0040  0F78 950A0FAF                    CLI   *+55,X'0A'           IS THIS A TYPE G CARD?     DY  MD  2180
0041  0F7C 47800F84                    BC    8,*+8                                            C  MD  2190
0042  0F80 47F00F96                    BC    15,*+22              IF NO GO TO CON.5           C  MD  2200
0043  0F84 D2000F910FB0                MVC   *+13(1),*+44         SET LENGTH FOR LOAD             MD  3010
0044  0F8A D2010F920FB2                MVC   *+8(2),*+40          SET ADDRESS FOR LOAD            MD  3020
0045  0F90 D2000FAC0FB8                MVC   *+28(1),*+40         LOAD TEXT                   P   MD  3030
0046  0F96 D2010046 0FAA               MVC   70(2),*+20           CON.5   SET BASE ADDRESS    P   MD  3040
0047  0F9C 47F00022                    BC    15,34                GO TO CON.1  (1ST CARD)     B   MD  3050
0048  0FA0 D2010FA60FBC                MVC   *+8(2),*+28          CON.4   SET START ADDRESS   P   MD  3060
0049  0FA6 47F00000                    BC    15,0                 GO TO MEMORY DUMP           B   MD  3070
0050  0FAA 0FAE                       DC    Y(*+4)               ADDRESS FOR SUBSEQUENT CARD     MD  3080
0051  0DBC                             ORG   *-496                MEMORY DUMP SECTION         N   MD  3090
0052  0DBC 0000                  MBGN  DC    Y(0)                 CONST. FOR BEGINNING ADDRESS    MD  3100
0053  0DBE 0FFF                  MEND  DC    Y(4095)              CONST. FOR ENDING ADDRESS       MD  3110
0054  0DC0 0080                  M?CO  DC    Y(128)               CONSTANT 128                    MD  3120
0055  0DC2 0F80                        DC    Y(M?PW+6)            STARTING ADDRESS FOR EDIT       MD  3130
0056  0DC4 0FC0                        DC    Y(M?PW+70)           ENDING ADDRESS FOR EDIT         MD  3140
0057  0DC6 0FF4                        DC    Y(M?PW+132-10)                                       MD  3150
0058  0DC8 405CF0F1F2F3F4F5F6    M?TB  DC    XL9'405CF0F1F2F3F4F5F6'  TRANSLATION TABLE FOR       MD  3160
0059  0DD1 F7F8F9C1C2C3C4C5C6          DC    XL9'F7F8F9C1C2C3C4C5C6'  63 CH BAR                  MD  3170
0060  0DDA 92080050              MENT  MVI   80,X'08'              M.D.ENTRY  SET LINE ADV BIT P  MD  3180
0061  0DDE 92000F6C                    MVI   M?CN+2,0                                           MD  3190
0062  0DE2 D2020F6D0F6C                MVC   M?CN+3(3),M?CN+2     SET VC B1,C1,F1,AND K1      P   MD  3200
0063  0DE8 48E00F6C                    LH    14,M?CN+2            SET REG 14 TO ZERO          P   MD  4010
0064  0DEC D2010F6A0DC4                MVC   M?CN(2),M?CO+4       SET LIMIT OF EDIT TO 4 GR.  P   MD  4020
0065  0DF2 A4030001              M?A   XIOF  1,3                  ISSUE PRINT ORDER 63 CH BAR P   MD  4030
0066  0DF6 47800E10                    BC    8,M?B0               IS ORDER ACCEPTED?          DN  MD  4040
0067  0DFA A5030F70                    TIO   M?CN+6,3             TEST I/O STATUS             P   MD  4050
0068  0DFE 47200DF2                    BC    2,M?A                IS PRINTER WORKING?         DN  MD  4060
0069  0E02 D2000E080F70          M?E   MVC   *+9(1),M?CN+6        SET STATUS BITS FOR DISPLAY P   MD  4070
0070  0E08 A9002300                    HPR   X'2300',0            PRINTER OFF NORMAL          H   MD  4080
```

Figure 3–1. Example of Printer Output of a Program

(Sheet 2 of 5)

```
0071   0E0C 47F00DF2          BC    15,M2A
0072   0E10 9501DF6C    M2E0  CLI   M2CN+2,1          GO TO A FOR RECOVERY          B   MD  4090
0073   0E14 47800EBA          BC    8,M2B2           IS VC B0 SET TO B2?           DN  MD  4100
0074   0E18 92EE0F7A          MVI   M2PW,X'EE'       IF YES GO TO B2               C   MD  4110
0075   0E1C D2820F7B0F7A      MVC   M2PW+1(132-1),M2PW B1                          N   MD  4120
0076   0E22 9501DF6B          CLI   M2CN+3,1         CLEAR STANDBY P.BUFFER AREA   P   MD  4130
0077   0E26 47800EC2          BC    8,M2C2           C0  IS VC C0 SET TO C2?       DN  MD  4140
0078   0E2A 9502DF6B          CLI   M2CN+3,2         IF YES GO TO C2               C   MD  4150
0079   0E2E 47800ECA          BC    8,M2C3           IS VC C0 SET TO C3?           DN  MD  4160
0080   0F32 48F00DC2    M2C1  LH    15,M2C0+2        IF YES GO TO C3               C   MD  4170
0081   0E36 F3420F7A0D3C      UNPK  M2PW(5),60(3)    SET STARTING ADDR OF EDIT     P   MD  4180
0082   0E3C 92EE0F7E          MVI   M2PW+4,X'EE'     EDIT ADDRESS                  P   MD  4190
0083   0F40 F3E7F000E000      UNPK  0(15,15),0(8,14)                               MD  4200
0084   0E46 F110F00EE007      MVO   14(2,15),7(1,14) EDIT DATA 7 BYTES             P   MD  5010
0085   0E4C 96F0F00E          OI    14(15),X'F0'     EDIT DATA 8TH BYTE            P   MD  5020
0086   0E50 96F0F00F          OI    15(15),X'F0'                                   MD  5030
0087   0E54 D2010F7EE006      MVC   M2CN+6(2),6(14)                                MD  5040
0088   0E5A A6120036          AI    62,18            STORE PREDECESSOR BYTES       P   MD  5050
0089   0E5E A6080038          AI    60,8             R15 + 18 TO R15               P   MD  5060
0090   0E62 47100EAE          BC    1,M2H            R14 + 8 TO R14                P   MD  5070
0091   0E66 49E00DBE          CH    14,MEND          IF R14 OVERFLOW GO TO H       DN  MD  5080
0092   0E6A 47200EAE          BC    2,M2H            IS R14 EQUAL TO MEM LIMIT?    DN  MD  5090
0093   0E6E 49F00F6A          CH    15,M2CN          IF YES GO TO H                C   MD  5100
0094   0E72 47400E40          BC    4,M2D            IS R15 EQUAL TO EDIT LIMIT?   DY  MD  5110
0095   0E76 49E00DC0          CH    14,M2C0          IF NO GO TO D                 C   MD  5120
0096   0E7A 47800E8C          BC    8,**+18          IS R14 EQUAL TO 128?          DN  MD  5130
0097   0E7E D2050F740F72      MVC   M2CN+10(6),M2CN+8 IF YES GO TO SS               C   MD  5140
0098   0E84 92060F6C          MVI   M2CN+4,0         EXTEND PREDECESSOR BYTES      P   MD  5150
0099   0E88 47F00F1E          BC    15,M2L           SET VC F0 TO F1               P   MD  5160
0100   0E8C 9201DF6B          MVI   M2CN+3,1         GO TO L                       B   MD  5170
0101   0E90 D201DF6AC0C0      MVC   M2CN(2),M2C0+6   SS      SET VC C0 TO C2       P   MD  5180
0102   0E96 48E00DBC          LH    14,MBGN          SET LIMIT OF EDIT             P   MD  5190
0103   0E9A 94FC0D3C          NI    61,X'FC'         SET BEGINNING ADDR TO R14     P   MD  5200
0104   0E9E 49E00DC0          CH    14,M2C0          AND ERASE 4 LSB               C   MD  6010
0105   0EA2 47A00F1E          BC    10,M2L           IS BG.AD. SMALLER THAN 128?   DY  MD  6020
                              IF NO GO TO L          C   MD  6030
```

Figure 3-1. Example of Printer Output of a Program

(Sheet 3 of 5)

```
0106  0EA6 48E00DC0                LH    14,M2C0                   SET BEGINNING ADDR TO 128    P   MD  6040
0107  0EAA 47F00F12                BC    15,M2L                    GO TO L                      B   MD  6050
0108  0EAE 92010F6C          M2H   MVI   M2CN+2,1                  SET VC R0 TO R2              P   MD  6060
0109  0EB2 92010F6F                MVI   M2CN+5,1                  SET VC K0 TO K2              P   MD  6070
0110  0EB6 47F00F12                BC    15,M2L                    GO TO L                      B   MD  6080
0111  0EBA 92000F6C          M2B2  MVI   M2CN+2,0                  SET VC R0 TO R1              P   MD  6090
0112  0EBE 47F00F18                BC    15,M2L+6                  GO TO TT                     B   MD  6100
0113  0EC2 92020F6D          M2C2  MVI   M2CN+3,2                  SET VC C0 TO C3              P   MD  6110
0114  0EC6 47F00E32                BC    15,M2C1                   GO TO C1                     B   MD  6120
0115  0ECA 48D0003C          M2C3  LH    13,60                     LOAD R13 FROM R14            P   MD  6130
0116  0ECE 48F00DC2                LH    15,M2C0+2                 PP   SET ST.ADDRESS OF EDIT  P   MD  6140
0117  0ED2 D507D000F72             CLC   0(8,13),M2CN+8            QQ   IS DATA EQUAL TO PRED?  DY  MD  6150
0118  0ED8 47600E32                BC    6,M2C1                    IF NO GO TO C1               C   MD  6160
0119  0EDC A612003E                AI    62,18                     R15 + 18 TO R15              P   MD  6170
0120  0EE0 A608003A                AI    58,8                      R13 +8 TO R13                P   MD  6180
0121  0EE4 47100E32                BC    1,M2C1                    IF R13 OVERFLOW GO TO C1     DN  MD  6190
0122  0EE8 49D00DBE                CH    13,MEND                   IS R13 EQUAL TO MEM LIMIT?   DN  MD  6200
0123  0EEC 47200E32                BC    2,M2C1                    IF YES GO TO C1              C   MD  7010
0124  0EF0 49F00F6A                CH    15,M2CN                   IS R15 EQUAL TO EDIT LIMIT?  DY  MD  7020
0125  0EF4 47400ED2                BC    4,M2C3+8                  IF NO GO TO QQ               C   MD  7030
0126  0EF8 48E0003A                LH    14,58                     LOAD R14 FROM R13            P   MD  7040
0127  0EFC 95010F6E                CLI   M2CN+4,1                  F0   IS VC F0 SET TO F2?     DN  MD  7050
0128  0F00 47800ECE                BC    8,M2C3+4                  IF YES GO TO PP              C   MD  7060
0129  0F04 92010F6E                MVI   M2CN+4,1                  F1   SET VC F0 TO F2         P   MD  7070
0130  0F08 92EF0F88                MVI   M2PW+14,X'FF'             FILL * INTO STANDBY          P   MD  7080
0131  0F0C D2690F920F80            MVC   M2PW+24(132-26),M2PW+6    PRINT BUFFER AREA            C   MD  7090
0132  0F12 DC830F7A0CDA      M2L   TR    M2PW(132),M2TB-238        TRANSLATE                    P   MD  7100
0133  0F18 A5030F70                TIO   M2CN+6,3                  TT   TEST I/O STATUS         P   MD  7110
0134  0F1C 47200F18                BC    2,M2L+6                   IS PRINTER WORKING?          DN  MD  7120
0135  0F20 91F90F70                TM    M2CN+6,X'F9'              IS THERE ANY ERROR?          DY  MD  7130
0136  0F24 47800F30                BC    8,M2K0                    IF NO GO TO VC K0            C   MD  7140
0137  0F28 92010F6C                MVI   M2CN+2,1                  SET VC R0 TO R2              P   MD  7150
0138  0F2C 47F00E02                BC    15,M2E                    GO TO E                      B   MD  7160
0139  0F30 95010F6F          M2K0  CLI   M2CN+5,1                  IS VC K0 SET TO K2?          DN  MD  7170
0140  0F34 47800F4A                BC    8,*+22                    IF YES GO TO K2              C   MD  7180
```

Figure 3–1. Example of Printer Output of a Program

(Sheet 4 of 5)

```
0141        0F38 95020F6F
0142        0F3C 47800F52
0143        0F40 D2830080 0F7A
0144        0F46 47F00DF2
0145        0F4A 92020F6F
0146        0F4E 47F00F40
0147        0F52 923C0050
0148        0F56 A4030003
0149        0F5A A5030F70
0150        0F5E 47200F5A
0151        0F62 A9002FFF
0152        0F66 47F00DDA
0153        0F6A
0154        0F7A
0155             00000A0000DDA
```

```
            CLI   M?CN+5,2
            BC    8,M?K3
M?K1        MVC   128(132),M?PW
            BC    15,M?A
            MVI   M?CN+5,2
            BC    15,M?K1
M?K3        MVI   80,X'3C'
            XIOF  3,3
            TIO   M?CN+6,3
            BC    2,*-4
            HPR   X'2FFF',0
            BC    15,MENT
M?CN        DS    CL16
M?PW        DS    CL132
            END   MENT
```

```
IS VC K0 SET TO K3?          DN  MD  7190
IF YES GO TO K3              C   MD  7200
LOAD DATA INTO PRINT BUFFER  P   MD  8010
GO TO A                      B   MD  8020
K2   SET VC K0 TO K3         P   MD  8030
GO TO K1                     B   MD  8040
SET LINE ADV BITS FOR H.P.   P   MD  8050
AND PAPER FEED TO H.P. POS   C   MD  8060
IS PRINTER WORKING?          DN  MD  8070
IF YES REPEAT TIO            C   MD  8080
SUCCESSFUL STOP              H   MD  8090
RETURN TO MENT               B   MD  8100
WS FOR VC AND PREDEC. BYTES      MD  8110
STANDBY PRINT BUFFER AREA        MD  8120
                                 MD  8130
```

Figure 3–1. Example of Printer Output of a Program

(Sheet 5 of 5)

3.1.6. Assembler Control Card

On the first pass, the source code deck may be preceded by a control card which has the following form:

LABEL	OPERATION	OPERAND
	CTL	ABS , p, q

where ABS indicates the output element is to be in absolute code form and is not to contain any external reference, p is a decimal number representing the largest address available on the computer on which the assembly is being done, and q is a decimal n umber representin g the largest address available on the computer for which the element is being assembled. Any field in the operand may be omitted. If ABS is omitted, the output element is in relocatable code form. If p is omitted, the memory size of the computer on which the element is being assembled is assumed to be 16,384. If q is omitted, the memory size of the computer for which the element is being assembled is assumed equal to the memory size of the computer on which the assembly is being done. The CTL card may be omitted, in which case the result is the same as indicated for each field omitted.

3.1.7. Operand Format

In general, operands take the format of a series of expressions separated by commas. If an expression is not expressed, the comma indicating its position must nevertheless be present. An exception to this rule is the last expression in the operand – if it is not expressed, its preceding comma may also be dropped.

3.2. SYSTEM CODES

Table 3–1 shows the relation the Assembler assumes between card code, internal computer code, and printer graphic. The Assembler reads a source code card in compressed form and then translates it to the internal code shown in Table 3–1. If keypunch equipment is used which sets up a different relationship between card code and printer graphic than the one shown in Table 3–1, a different translation table may be substituted at linker time for use by the Assembler in translating source code cards. This translation table may set up any relation between card code and printer graphic that is desired; however, the relation between internal code and printer graphic shown in Table 3–1 must remain inviolate, since this is the only way the Assembler can "read" the source code. The Assembler prints its listing directly from the internal code. This operation, in effect, assumes a 63-character print bar. If a 48-character print bar is used while assembling, the Assembler may be modified at linker time to translate printer output from internal code to 48-character print bar code before printing.

The Assembler punches all output cards in a compressed "object code" form which may be handled directly by the Linker or the absolute loader.

Some users may provide programs via the Assembler to be used to process data represented in an internal code different from the one used by the Assembler. In such a case, the user must take special care in the representation of his constants. For example, the Assembler assigns the internal code 11000001 to the graphic "A". If, at the time an object program is run, the internal code for the data assigns the code 11000000 to the graphic "A", a test for equality against a constant represented as C'A' in source code language may not be performed as desired.

In general, when data to be processed by an object program is represented in an internal code other than that used by the Assembler, all difficulties can be avoided by representing all constants in the source code in hexadecimal.

TWO MOST SIGNIFICANT BITS OF ZONE – 00

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

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

TWO MOST SIGNIFICANT BITS OF ZONE – 01

DIGIT	TWO LEAST SIGNIFICANT BITS OF ZONE			
	00	01	10	11
0000	ŧ	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 "

Table 3–1. Internal Code (Sheet 2 of 4)

TWO MOST SIGNIFICANT BITS OF ZONE – 10

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

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

TWO MOST SIGNIFICANT BITS OF ZONE – 11

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

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

4. OPERATING PROCEDURES

4.1. GENERAL OPERATING INSTRUCTIONS

A source code deck ready for assembly must pass through the computer twice. The Assembler Load deck marked FIRST PASS precedes the source code deck on the first pass; the Assembler Load deck marked LAST PASS precedes the source code deck on the second pass.

All printing and punching is done during the second pass. Consequently, the second pass may be repeated as often as required. The LAST PASS Assembler Load deck must precede the source code deck each time. When the card reader is used for input, feed the first card, then depress the PROC CLEAR and START keys.

In addition to the regular I/O displays listed in Appendix B, the following displays may be used.

DISPLAY	REASON AND ACTION
1F02	Symbol table is full. Use larger memory size if possible (specification p on the CTL card) and start over. When p is 8191, the Assembler can handle about 200 tags. Press RUN to continue (all subsequent tags will be undefined).
1FFF	LAST PASS is completed. Two blank cards must follow the END card on the LAST PASS in order to get this display.

4.1.1. Card Controller Operating Instructions

The following sections provide instructions for operating the Card Assembler when the UNIVAC 1001 Card Controller is being used as the input device. Instructions are given for starting the run from the beginning, and for rerunning the second pass.

4.1.1.1. Start Instructions

To start the Assembler, perform the following steps:

1. Place the source code between the FIRST PASS and LAST PASS load decks.
2. Place the entire deck in the Card Controller primary read hopper.
3. On the Card Controller:
   a. Set ALT1 to the ON position; set all other ALT switches to the OFF position.
   b. Depress the LOAD PR1, CLEAR, START and RUN switches.
4. On the UNIVAC 9200/9300 console:
   a. Enter hexadecimal B8 in the DATA ENTRY switches.
   b. Press the CHAN CLEAR, PROC CLEAR, LOAD ON, RUN, LOAD OFF, and RUN switches.
5. On the Card Controller, set ALT1 to the OFF position.

4.1.1.2. Second Pass Rerun Instructions

To rerun the second pass of the Assembler, perform the following steps:

1. On the Card Controller:
   a. Place the LAST PASS load deck, followed by the source deck, into the primary feed hopper.
   b. Set all ALT switches to the OFF position.
   c. Press the LOAD PR1, CLEAR, START, and RUN switches.
2. On the UNIVAC 9200/9300 console, press the CHAN CLEAR, PROC CLEAR, and RUN switches.

4.2. ASSEMBLER CARD OUTPUT

The object code produced by the Assembler is punched into six different card types: Element Definition Cards, External Definition Cards, Program Reference Cards, External Reference Cards, Text Cards, and Transfer Cards. These card types have the following functions:

- The Element Definition Card contains the name, the size, and the origin of the element as assigned by the Assembler.
- An External Definition Card specifies the value of a symbol which may be referenced by other elements.
- The Program Reference Card contains the name of the element and the number by which this name is identified in the relocation information for the element.
- An External Reference Card contains a label to which the element refers but which it does not define. The card also contains a number by which this label is identified in the relocation information for the element.
- A Text Card contains the instructions and constants of the element, an address indicating where the instructions and constants are to be loaded into memory for execution, and the relocation information pertaining to the instructions and constants. The loading address for the instructions and constants is assigned by the Assembler to conform with the origin of the element as described in the Element Definition Card. The relocation information performs two functions:
  - It permits the relocation of the instructions and constants to an origin other than the one given to the element by the Assembler.
  - It provides the information required by the Linker to resolve any external references made in the instructions or constants with the corresponding external definitions made in other elements.

- The Transfer Card is generated by the END assembler directive. If the END directive specifies the address at which execution is to begin, this address appears in the Transfer Card.

The order and number of these cards in the Assembler object code output deck is as follows. First there is a single Element Definition Card. Then there are as many External Definition Cards as there are ENTRY assembler directives in the source code. Then there is a single Program Reference Card followed by as many External Reference Cards as there are EXTRN assembler directives in the source code. Then there are as many Text Cards as are required to contain the instructions and constants represented in the source code deck. Finally, there is a single Transfer Card.

If the output of an assembly contains no External Reference Cards, it may be loaded directly into the UNIVAC 9200/9300 via the Card Program Loader. In this instance, the text is loaded at the addresses indicated in the Text Cards, and job execution begins at the point indicated in the Transfer Card. The Element Definition Card, any External Definition Cards, the Program Reference Card, and the relocation information in the Text Cards are ignored by the Program Loader.

The format of these assembler output cards is as follows.

4.2.1. Element Definition Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12–2–9 punch
2	Type	A (Hollerith)
3	Length	26
6	Absolute/relocatable	12 punch if absolute program, relocatable otherwise.
7	Hole Count	Sum of the bytes punched in columns 8–72.
8	ESID	External Symbol Identification assigned by the Assembler to the name in columns 17–24.
13–16	Start Address	The base of this element as assigned by the Assembler.
17–24	Name	The name assigned to this element. (The name is left justified and is punched in EBCDIC.)
33–36	Length	The number of bytes of memory needed by the relocatable portions of this element.

4.2.2. External Definition Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12-2-9 punch
2	Type	H (Hollerith)
3	Length	13 (or number of columns used less one from Col. 11).
7	Hole Count	Sum of the bytes punched (columns 8–72).
9	RLD Length	Number of columns of RLD information on card (indicates 3 or 0).
10	Last RLD	Column 11 relative number indicating the most significant column of the last item of RLD information on the card. The value is 59 if there is relocation data; otherwise zero.
14–16	Symbol address	The Assembler assigned value of the symbol field.
17–24	Symbol	Symbolic name to be referenced by other program(s) (punched in EBCDIC).
70–72	RLD	Relocation field. See the description of this field for the Text Card. If present, column 72 contains a 3 and the least significant digit of column 71 also contains a 3 indicating that columns 14–16 are to be modified.

4.2.3. Program Reference Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12-2-9 punch
2	Type	J (Hollerith)
3	Length	13 (or number of columns used less one from Col.11).
7	Hole Count	Sum of the bytes punched (columns 8–72).
8	Program ESID	External Symbol Identification assigned by the Assembler to the program name.
13–16	Assembled Start Address	The base of this program as assigned by the Assembler.
17–24	Name	Element name (same as columns 17–24 of the Element Definition Card).

4.2.4. External Reference Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12-2-9 punch
2	Type	K (Hollerith)
3	Length	13 (or number of columns used less one from Col. 11).
7	Hole Count	Sum of the bytes punched (columns 8–72).
8	Name ESID	External Symbol Identification assigned by the Assembler to this symbolic name.
17–24	Name	Symbolic name being referenced by this card (punched in EBCDIC).

4.2.5. Text Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12-2-9 punch
2	Type	Q (Hollerith)
3	Text Length	Indicates the number of columns less one of text information on the card.
4–6	Load Address	The Assembler assigned location where the text is to be loaded.
7	Hole Count	Sum of the bytes punched (columns 8–72).
8	Program ESID	External Symbol Identification assigned by the Assembler to the program name to which this load address is relative.
9	RLD length	Number of columns of RLD information on this card.
10	Last RLD	Column 11 relative number indicating the most significant column of the last item of RLD information on the card. This number is 59 if there is RLD data, otherwise zero.
11 & following	TXT	The value to be loaded at the load address. The TXT field contains information from columns 11 through 11+n, where n is the number contained in column 3.
72 & preceding	RLD	RLD fields begin in column 72 and occur from right to left on the card for the number of columns indicated in column 9. Each RLD field is composed of three columns.

Example of RLD field:

Column 70 contains a name ESID. This points to a value in the linker reference table to be applied to the TXT on this card.

Column 71 contains a flag. The four most significant bits indicate the operation. All zero bits indicate that the reference table value is to be added to the text value to obtain the new text value. If the four most significant bits of the flag column are 0001, the reference table value is subtracted from the card text value to obtain the new text value.

The three least significant bits of the flag column indicate (in binary) the length of the text field in bytes. The remaining bit is a one if the field to be modified contains an additional halfbyte. Thus, the four least significant bits would contain the value eight for a four-bit field. If all four bits are zero, the field is four bits long and is in the left halfbyte.

Column 72 contains column position. A binary number (relative to column 11) pointing to the most significant column of the text information to be modified. (Column 11 is numbered as zero, column 12 as one, and so on.)

4.2.6. Transfer Card

COL.	FIELD NAME	CONTENTS
1	Load Key	12-2-9
2	Type	Y (Hollerith)
3	Length	5 (or number of columns less one from Col. 11).
7	Hole Count	Sum of the bytes punched (columns 8–72).
9	RLD Length	Number of columns of RLD information on the card. (Indicates 3 or 0.)
10	Last RLD	Column 11 relative number indicating the most significant column of the last item of RLD information on the card. (Contains 59 if there is relocation data, otherwise 0.)
11–13	Card Count	The number of reference type K or text type Q cards which were produced by the assembler for this element. (Carried in binary.)
14–16	Start Address	
70–72	RLD	Relocation field. Column 72 contains column 11 relative indicator of the first column of the start address (indicates Col. 14). The most significant 4 bits in column 71 are 0001 if the reference table address is to be subtracted from the card start address or 0000 if the reference table address is to be added to the card start address to obtain the relocated start address. The least significant 4 bits in column 71 indicate that the start address on the card is 3 bytes long. Column 70 contains the ESID that points to the value in the reference table to be applied to the card's start address field.

For all assembler output cards, the PID is left justified in columns 73–76, and a sequence number is punched in columns 77–80. Both the PID and sequence number are punched in Hollerith.

4.3. CARD ASSEMBLER PRINTED OUTPUT

The first page printed during assembly is a list explaining the one-character error codes that may occur on the succeeding assembly listing. Up to five of the codes may appear on one line in the assembly code field. The assembly listing contains the following:

PRINT POSITION	FIELD
0–3	Assembler assigned line number
4–8	Assembly codes
10–13	Assembler assigned address of the object code or (in the case of an EQU or an ORG line) the value assumed by the assembler.
15–46	Assembler produced object code (in hexadecimal).
48–127	Input card

The error codes and their meanings are as follows:

C Cover error, no USING covering relocatable operand address.

D Doubly defined label or reference to doubly defined label.

E Expression too large or improper syntax.

H Halfword boundary error on RX or AI operand.

I Instruction error.

L Location counter too large.

O Org error, 2nd definition of a label.

R Relocatable terms in the expression are improper or too many.

S Sequence break in columns 76 to 80.

T Truncation of oversize term.

U Undefined label referenced in this line.

X Continuation (no blank col 72 on noncomment card)-not permitted.

4.4. LINKING THE CARD ASSEMBLER

The source code for the Assembler consists of nine elements. From these elements the Linker produces three phases.

The first phase (having 1 punched in column 77) is the Assembler's FIRST PASS.

The second and third phases together (having 2 or 3 punched in column 77) become the Assembler's LAST PASS.

The example shown below would produce a card assembler for the following machine configuration:

Standard card reader

Serial punch

Standard printer/63 character print bar

LINKER INPUT	LINKER OUTPUT
CTL 2,16383,16383	
PHASE A91,510,A	
RDTT EQU 0,TBRD	
PRTT EQU 0,TBRD	
BLNK EQU 64	
PRTR EQU 15	
FONT EQU 0	
TBPR EQU 0	
Loader module (LD)	PHASE1 A91 – First Pass
Punch module (XPCH)	
Reader module (XRDR)	
Printer module (XPRT)	
Read translate table module (TBRD)	
Assembler first module (FIRP)	
PHASE A92,O,L,INTF	
Assembler second module (MIDP)	PHASE2 A92 – Last Pass
Punch translate table module (TBPU)	
PHASE A93,O,L,INTF	
Assembler third module (LASP) – END	PHASE3 A93 – Last Pass

In order to produce an assembler for a machine configuration using a row punch instead of a serial punch:

- Substitute an XPRW module for the XPCH module.
- After the first phase card, place the following card:

LABEL	OPERATION	OPERAND
CHNL	EQU	n

where: n is the channel number of the row punch.

In order to make an assembler that would use a UNIVAC 1001 Card Controller instead of the standard card reader:

- Substitute an XRDC module for the XRDR module.
- Substitute an LDCC module for the LD module.
- Change the second parameter of the first phase card from 510 tc 600.
- Include an EQU card of the following format:

LABEL	OPERATION CODE	OPERAND
RDCN	EQU	n

where: n is the number of the channel in which the Card Controller is located.

In order to make an assembler that would print on a 48 character bar:

- Substitute PRTT EQU 0, TBPR for PRTT EQU 0, TBRD.
- Substitute BLNK EQU 16 for BLNK EQU 0.
- Substitute PRTR EQU 0 for PRTR EQU 15.
- Substitute FONT EQU 128 for FONT EQU 0.
- Remove TBPR EQU 0.
- Include the print translation table module (TBPR) immediately after the XPRT module.

5. LINKER

5.1. INTRODUCTION

When a job consists of more than one element, the elements, which are the output of separate Assembler runs, must be combined before they may be loaded as an executable object program. This combining, or linking, is done by a utility program called the Linker. The Linker inserts the storage addresses for references made from one element to another and modifies addresses if an element is relocated.

A provision is included for dividing the output elements into separate loads or "phases". Another provision allows corrections, stated in hexadecimal, to be made to any of the elements being linked. These corrections must be in terms of the ultimate absolute addresses assigned to each field being changed.

Most of the input to the Linker consists of the output of one or more Assembler runs. However, control cards are supplied by the user to specify:

- the initial storage address to be allocated to the output element (PHASE card)
- the start of a new phase of the output (PHASE card)
- additional external definitions (EQU card)
- corrections to one or more of the elements being linked (REP)
- the end of the input stream (END)

The Linker provides an output listing including:

- the control cards on its input,
- the names and external definitions of the elements being linked and the values allocated to each, as well as the number of the phase in which it is included. Phases are numbered consecutively from one in the order in which they appear in the input.

Error indications are included in the listing, and most errors cause termination of the punched output. The punched card output is in the same form as the assembler output cards, except that no relocation data is punched. The output for each phase consists of Text Cards and a Transfer Card.

If necessary, the Linker increments the address to be assigned to each input element so that the base address is a multiple of four.

The Linker is capable of either a one- or two-pass operation. At the end of pass one a stop occurs with a display indicating readiness for pass two. At the end of pass two a stop with a display requiring a reply occurs. When the start button is depressed, the Linker interrogates this reply to determine its subsequent action, which is to process another set of input or to terminate processing.

The Linker is assembled separately from its input/output but is linked to the input/output, allowing for input from the standard card reader or the UNIVAC 1001, output to serial or row punch, and choice of input translation table and the option of a translation for the 48-character printer.

5.2. LINKER INPUT

The major input to the Linker consists of the output of one or more assemblies. The input to the Linker is normally formed by placing one element behind the other in the order they are to have in storage. Then a PHASE card is placed at the beginning of the deck to define the initial storage location and an END card at the end to signal the end of the input. If the output element is to consist of more than one phase, each input element must be entirely in one phase, with a PHASE card inserted in front of the first Element Definition Card in the phase. Each such PHASE card indicates the initial address to be allocated to that phase. When the Linker input is arranged in this manner, all elements comprising one phase must follow the PHASE card defining that phase and precede the PHASE card defining the next phase. Each element in the input must have a unique name.

The order of the input must also be such that the element using an externally defined symbol must precede all elements referring to that symbol. If there are any symbols for which this is not possible, their definitions may be supplied by EQU cards. If this is not desirable, the Linker provides the option of a two-pass operation. The first pass recognizes the headers (Element Definition and External Definition Cards) and stores the external definitions. The second pass processes the External Reference, Text, and Transfer Cards, and produces the output element.

If desired, a two-pass operation may be avoided by separating the headers of the input elements and presenting them first. The procedure is as follows:

1. *Put together the input elements as described above, but without control cards;*
2. *Sort out the header cards (12 punch in column 2);*
3. *Place the header cards in front of the remaining deck;*
4. *Insert the required control cards.*

Each PHASE card should precede the Element Definition Card for the first element in the phase being defined. EQU cards follow a PHASE Card, Element Definition, or External Definition Cards. REP cards must immediately precede the Transfer Card of the element they are to alter.

The Linker ignores the presence of any blank cards in its input deck.

5.3. LINKER CONTROL CARD FORMATS

The control card identifier (CTL, PHASE, EQU, REP, or END) is left justified in columns 10–14. Columns 1 to 9 are blank except for the EQU card on which columns 1 to 4 contain the symbol being defined. The specifications contained on each control card begin in column 16 and are terminated by a blank.

5.3.1. CTL

The CTL card is the first card of the Linker input The specifications consist of three fields separated by a comma:

n,p,q

where n=1 is a one-pass operation of the Linker;

n=2 is a two-pass operation of the Linker;

p is a decimal number representing the largest address available during linking.

q is a decimal number representing the largest address available to the output element.

Any field may be omitted. The effect is as follows:

n omitted : one-pass operation.

p omitted : largest address available for linking is not to change. The initial value is 16383.

q omitted : maximum address to be allocated is not to change. The initial value is 16383.

The CTL card may be omitted, in which case the result is the same as indicated above for each field omitted.

If the Linker is to perform a two-pass operation and produce code for a 16K system on a 16K system, the CTL card would be

```
CTL   2,16383,16383
```

5.3.2. PHASE

A PHASE card defines the name and initial storage address for the output element and must be the first or second card of the Linker input, preceded only by the CTL card. A PHASE card also precedes the Element Definition Card (type A) for the first element of each subsequent load. It specifies the name of the phase and its starting address.

The operand specifications field has the form

phase-name,displacement,flag,symbol

where phase-name is a group of up to four alphabetic characters representing the name of the phase

displacement is a decimal number (may be preceded by minus) or a hexadecimal number in the form X'nnnn'

flag — is C or A for the first PHASE card and C, A, or L for any others.

- C – load address equals the highest core address minus the displacement field.
- A – load address is the actual value given in the displacement field.
- L – load address is obtained by adding the displacement to the value of the symbol.

symbol — is any previously defined symbol.

5.3.3. EQU

An 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 operand specification field of the EQU card has the form:

```
value
```

or

```
value,symbol
```

where value — is a decimal number, a decimal number preceded by a minus sign, or a hexadecimal number in the form X'nnnn'

symbol — is any symbol which has been defined previous to the EQU card in the input deck.

In the first form above, the binary value represented by the value field becomes the value assigned to the symbol appearing in the label field of the EQU card. For an EQU card with a specification field of the second form above, the value of the previously defined symbol is added to this value to yield the value of the symbol being defined.

An EQU card must follow a PHASE card, an Element Definition Card, an External Definition Card, or another EQU card. It must precede the body of the first element containing a reference to the symbol defined. The symbol, contained in the specification field, must have been previously defined.

If the Linker control deck contains more than one EQU card defining the same symbol, an error indication is made on the listing. However, such an error does not terminate the punching of output. Instead, the Linker continues to treat the definition given in the first such EQU card as the definition for the symbol.

5.3.4. END

The END card indicates the end of the input to the Linker and is the last card in the deck.

The operand specification field has the same form as that of the EQU card, and is processed in the same way to produce a single value which is interpreted as the address at which to begin executing the last phase being produced by the Linker. as such, this value is punched into the Transfer Card at the end of the output element.

If the output of the Linker consists of more than one phase, the transfer address of each phase but the last is determined as follows:

a. Normally, the transfer address of the phase is the address from the first Transfer Card in the input to the phase.

b. If no Transfer Card in the input contains an address, the transfer address is the lowest address assigned to the phase.

The specification field of the END card may also be blank. In this case the transfer address punched into the terminal Transfer Card of the output element is the address from the first Transfer Card of an input element in that phase containing an address. If no Transfer Card of an input element contains an address, the lowest address assigned to that phase is punched into the terminal Transfer Card.

5.3.5. REP

The REP (Replace) cards specify changes which are to be made to an assembled element. The REP cards are placed immediately in front of the Transfer Card of the element to be altered. Addresses and data are specified in hexadecimal in the same form they are to have in the output element. No relocation or linking facilities are provided by the Linker for this data.

The form of the operand specifications field is:

address,data,data,...

where: address is a field of from one to four hexadecimal digits specifying the storage address of the leftmost byte of data to be altered as a result of this card.

data is a field of from one to four hexadecimal digits specifying data to be right justified in a halfword of storage. The address field is followed by a variable number of such data fields specifying the contents of successive halfwords of memory. The fields are separated by commas and terminated by a blank.

5.3.6. MOD

The MOD (modular set) card instructs the linker to set the location counter to the value calculated from the operand specifications. The operand specifications field has the following form:

a, b

where: a must be a power of 2, and may be a decimal or hexadecimal expression.

b may be omitted, or it may be a decimal or hexadecimal expression.

The location counter is set to the next number which is the value of "b" more than a multiple of the value of "a" and which is greater than or equal to its present value.

Examples:

```
MOD  8
```

If the present value of the location counter is 4024, then the value is not modified. If the current value is 4025, it is modified to 4032.

```
MOD  8, 3
```

If the current value of the location counter is 4028, its value is modified to 4035.

The MOD card must be placed immediately before the object deck of a module.

5.4. EXAMPLE

Assume two separately assembled elements, A and B. A was assembled at an origin of 0 and has a length of 100, while B was assembled at an origin of 400 and has a length of 200. Further, A externally defines one entry point M, which is assigned an element relative address of 50, and makes external references to symbols X, Y and Z. B on the other hand externally defines symbols X, Y, and Z and makes an external reference to M. Symbols X and Y are entry points with relative addresses of 475 and 550, respectively, while Z is defined as having an absolute value of 25. Finally, neither the A nor the B element Transfer Card specifies a starting address. The object code decks for elements A and B have the following construction.

Element A

a. One Element Definition Card specifying that this element is named A and has an origin of 0 and a length of 100.

b. One External Definition Card specifying M as an externally defined symbol with an element relative value of 50.

c. One Program Reference Card specifying that this element is named A, that this name has an External Symbol Identification (ESID) number of 1, and that element A has an origin of 0.

d. Three External Reference Cards specifying that X, Y, and Z are externally referenced symbols which have ESIDs of 2, 3, and 4, respectively.

e. Text Cards containing the instructions and constants of element A and the relocation information for these instructions and constants. Two examples may clarify the nature of this relocation information:

   1. An instruction may refer to some other part of element A. This reference is relative to the origin of the element. If the origin moves, the reference must be adjusted accordingly. The associated relocation information indicates where this reference is made in element A and specifies the ESID of element A, indicating that this is an element relative reference.

   2. An external reference may be made. In this case, the reference is undefined. The associated relocation information indicates where in element A this reference is made and specifies the ESID identifying the undefined symbol referenced.

f. One Transfer Card.

Element B

a. One Element Definition Card specifying that this element is named B and has an origin of 400 and a length of 200.

b. Three External Definition Cards.
   1. One specifies that X is an externally defined symbol and that it has an element relative value of 475.
   2. One specifies Y with an element relative value of 550.
   3. One specifies Z with an absolute value of 25.

c. One Program Reference Card specifying that the element is named B, that it has an ESID of 1, and that it has an origin of 400.

d. One External Reference Card specifying M as an externally referenced symbol with an ESID of 2.

e. Text Cards containing the instructions and constants of element B and the relocation information for these instructions and constants.

f. One Transfer Card.

These two decks are represented schematically in Figure 5–1. Suppose elements A and B are to be linked into one job having an origin of 1000 and whose initial execution address is to be the beginning of element A. The origin would be specified in a PHASE card, the transfer address in an END card. The input to the Linker for a one-pass operation would appear as shown in Figure 5–2.

The Linker reads the PHASE card and sets the location counter to 1000 in preparation for creating a job to be loaded beginning at memory location 1000. The Linker then reads the header cards and sets up the *reference table*. Each entry in the reference table consists of three fields.

1. The name which this entry describes.
2. The location assigned to this name.
3. The *relocation factor* for this name. The relocation factor is the amount by which the value assigned to the name by the Assembler must be adjusted to arrive at the value to be assigned to the name by the Linker.

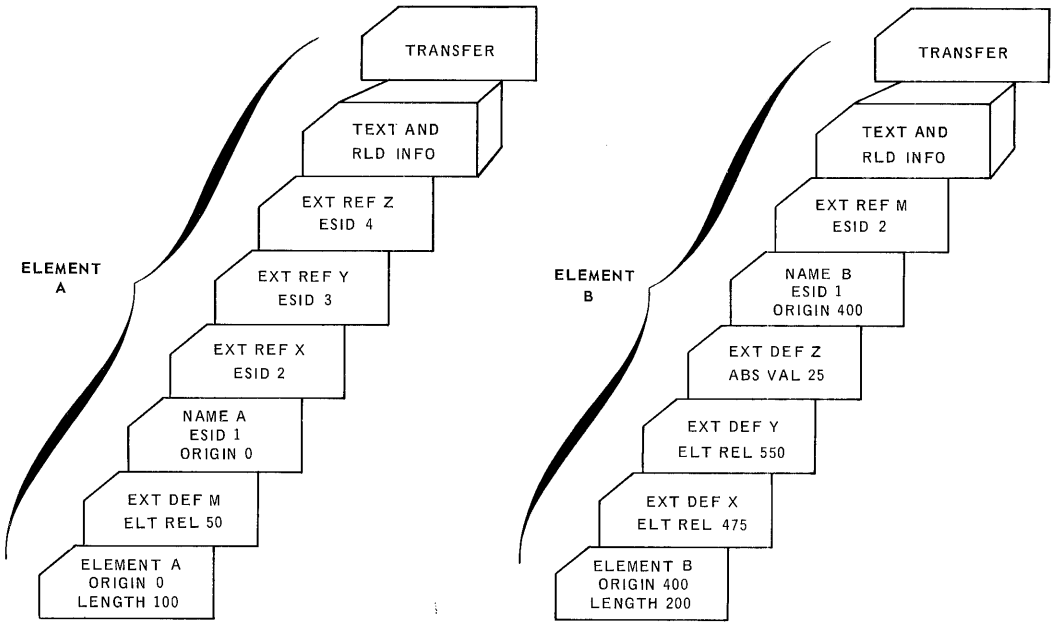


Figure 5–1. Elements A and B Deck Structure

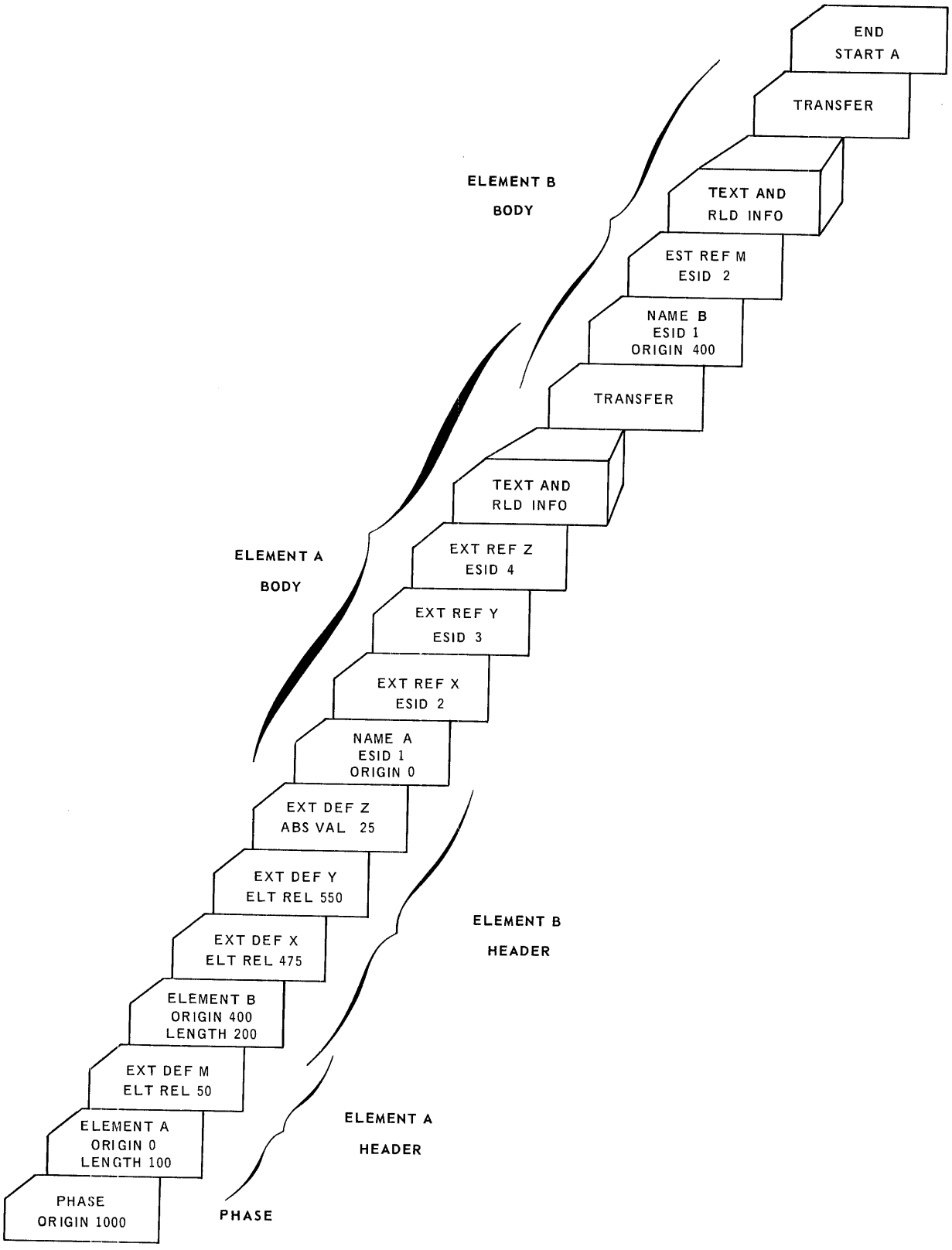


Figure 5–2. Linker Input

For example, the name "A" is to be assigned a value of 1000 by the Linker. It was assigned a value of 0 by the Assembler; therefore, its relocation factor is 1000.

As a second example, consider the name "B".

1. Since element A begins in location 1000 and is 100 bytes long, the name "B" is assigned a value of 1100 by the Linker.

2. While the body of element A is being processed, the name "B" has a relocation factor of 1100, since the name "B" is undefined in element A.

3. While the body of element B is being processed, the name "B" has a relocation factor of 700, since in element B the Assembler assigned a value of 400 to the name "B".

The reference table produced as a result of processing the header cards in Figure 5–2 is shown in Figure 5–3.

The Linker then reads the Program and External Reference Cards for element A. The information from these reference cards is used by the Linker to build an ESID table. Each entry in the ESID table consists of two fields:

1. The ESID from the reference card.

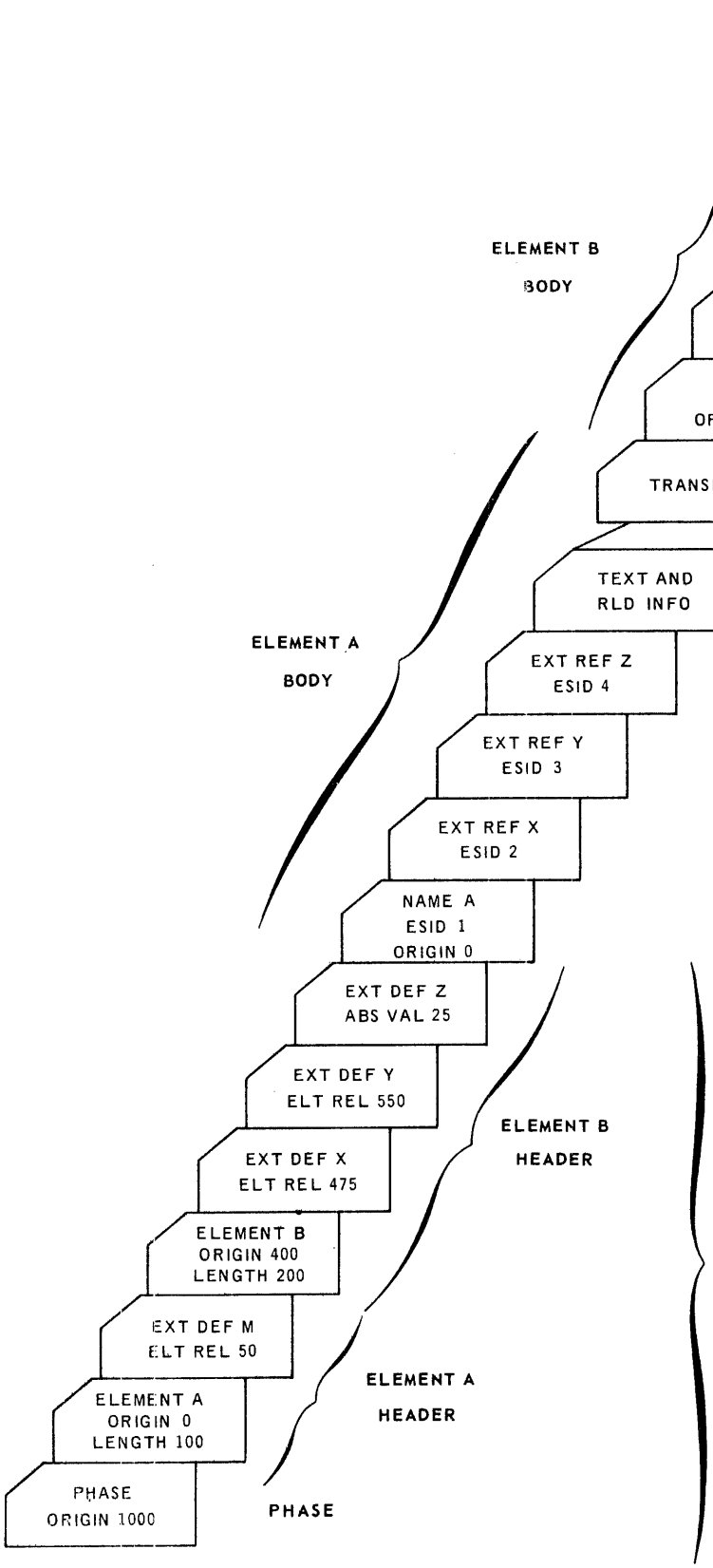
2. The reference table entry number of the symbol to which the ESID is assigned.

The Program Reference Card is also used to determine the relocation factor for the element name. The result of processing the reference cards is shown in Figure 5–4.

The Linker then processes the text of element A. For each instruction or constant on the input text cards it produces an instruction or constant on an output Text Card. The absolute portions of the text are produced unaltered. The address at which the text is to be loaded is adjusted by the relocation factor for element A.

If a portion of the text is relocatable, then there is associated with it relocation information specifying an ESID of 1. In this case, the Linker looks up in the ESID table the associated reference table entry number. It then looks up in the reference table the relocation factor (1000) and adjusts the text by the relocation factor. The input text is then relocated to the origin specified by the PHASE card, and this relocated text is produced as output.

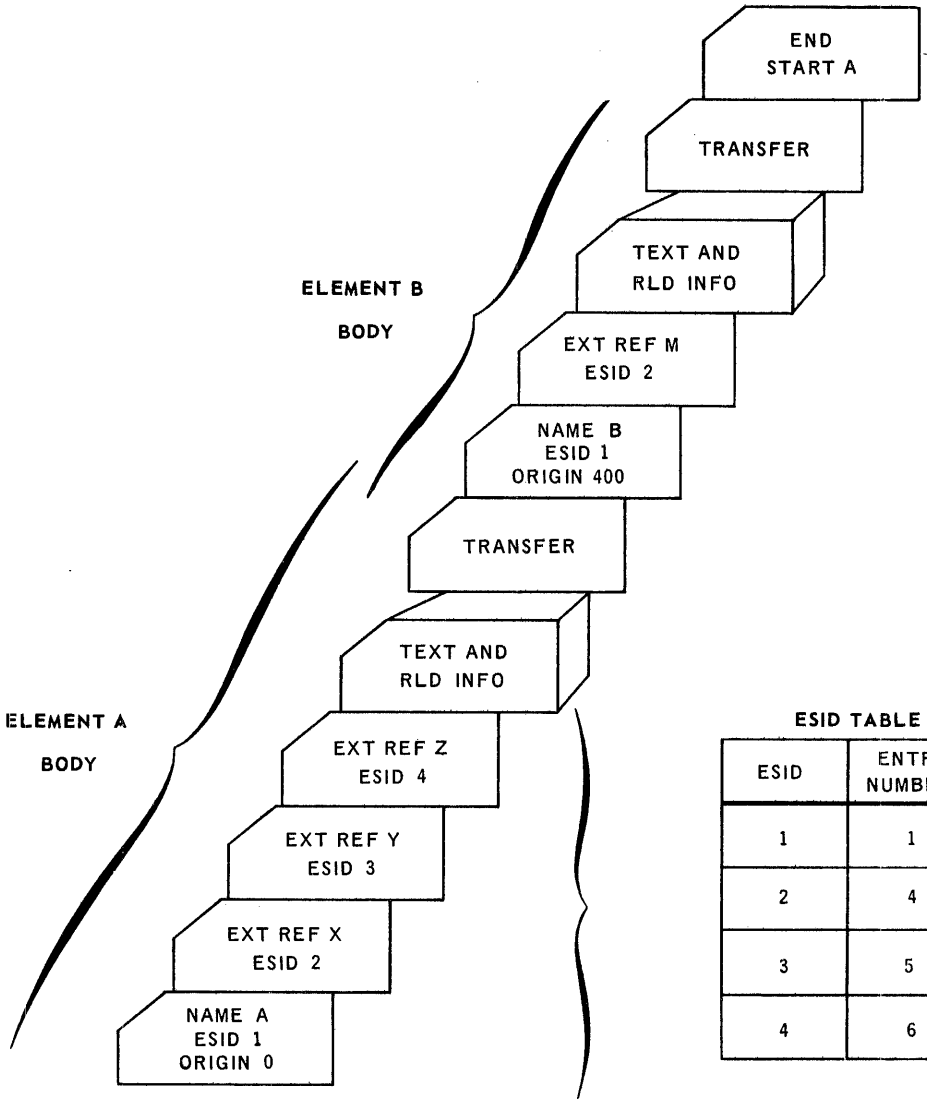
The Linker performs a similar function if a portion of the text makes an external reference. (Assume the reference is made to the symbol Y.) There is associated with this text relocation information specifying the ESID of the external reference (3). The text is adjusted by the relocation factor (1250) determined by the relation between ESID and reference table entry number (5). This defines the external reference, and the resolved text is produced as output.



REFERENCE TABLE

ENTRY NUMBER	NAME	VALUE	RELOCATION FACTOR
1	A	1000	1000
2	M	1050	1050
3	B	1100	1100
4	X	1175	1175
5	Y	1250	1250
6	Z	25	25

Figure 5–3. Header Processing



ESID TABLE

ESID	ENTRY NUMBER
1	1
2	4
3	5
4	6

REFERENCE TABLE

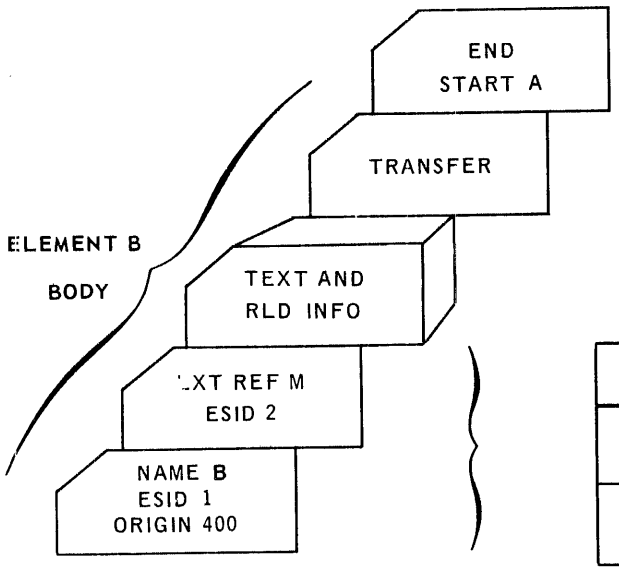
ENTRY NUMBER	NAME	VALUE	RELOCATION FACTOR
1	A	1000	1000
2	M	1050	1050
3	B	1100	1100
4	X	1175	1175
5	Y	1250	1250
6	Z	25	25

Figure 5-4. ESID Processing for Element A

The Linker recognizes the end of element A by means of the Transfer Card. It then reads the Program and External Reference Cards for element B and adjusts the reference and ESID tables accordingly. The result of this adjustment is shown in Figure 5–5. Note that the relocation factor for the name "B" is changed.

The Linker then uses the ESID and reference tables to process the text of element B and produces the related output text completely relocated and with all external references defined. In response to the END card, the Linker produces a Transfer Card with a value of 1000 (the value of the name "A") in it for a Transfer Address. Thus, the output of the Linker is a deck of Text Cards with no relocation information, followed by a Transfer Card.

If a third element were to follow element B as input to the Linker, the relocation factor for the name "B" would be set back to 1100 by the Linker before it processed this third element.



ESID TABLE

ESID	ENTRY NUMBER
1	3
2	2

REFERENCE TABLE

ENTRY NUMBER	NAME	VALUE	RELOCATION FACTOR
1	A	1000	1000
2	M	1050	1050
3	B	1100	700
4	X	1175	1175
5	Y	1250	1250
6	Z	25	25

Figure 5–5. ESID Processing for Element B

5.5. ONE- AND TWO-PASS LINKING

One-pass linking of an input deck is possible when all symbols on External Reference (type K) cards have been defined by some preceding Element Definition (type A), External Definition (type H), or EQU control card.

When the definition of a symbol occurs after an External Reference to the symbol, a two-pass run is necessary. However, if the definitions are moved to the front of the deck, a one-pass run can then be made. The move is accomplished by sorting on a 12-punch only in column 2 of the Assembler-produced cards. All these definitions appear on Element Definition (type A) and External Definition (type H) cards which are at the beginning of an Assembler-produced element. Consequently, the cards can be easily stripped off by hand.

For the first pass of a two-pass operation, the Linker object deck precedes the input deck in the input hopper of the card read unit. For the second pass, the input deck only is placed again in the input hopper.

Figures 5–6 and 5–7 illustrate the placement of control cards for a one- and two-pass linking operation.

5.6. LINKING THE LINKER

A new loadable Linker program is produced by combining the several separately assembled components of the Linker by means of a Linker run. The combined Linker occupies memory locations from the upper limits of privileged memory to approximately 1824. Memory addresses greater then 1824 are used by the Linker for ESID tables and the reference table. The Linker can thus handle, when operating in 8K, approximately 200 external references during linking.

Computers with larger memory capacities can handle an even greater number of external references.

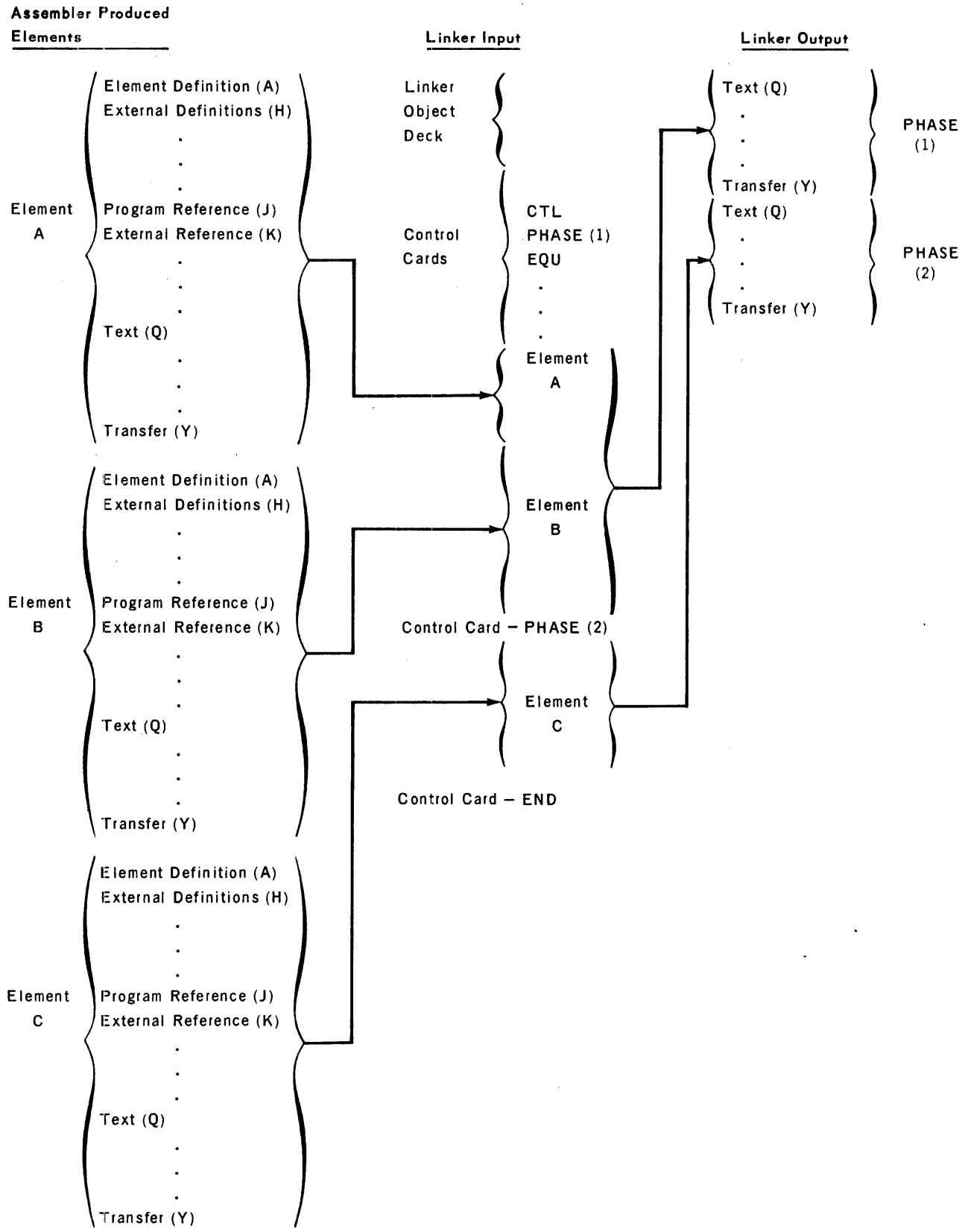


Figure 5–6. Linker Input Deck Sequence for Two-Pass Operation

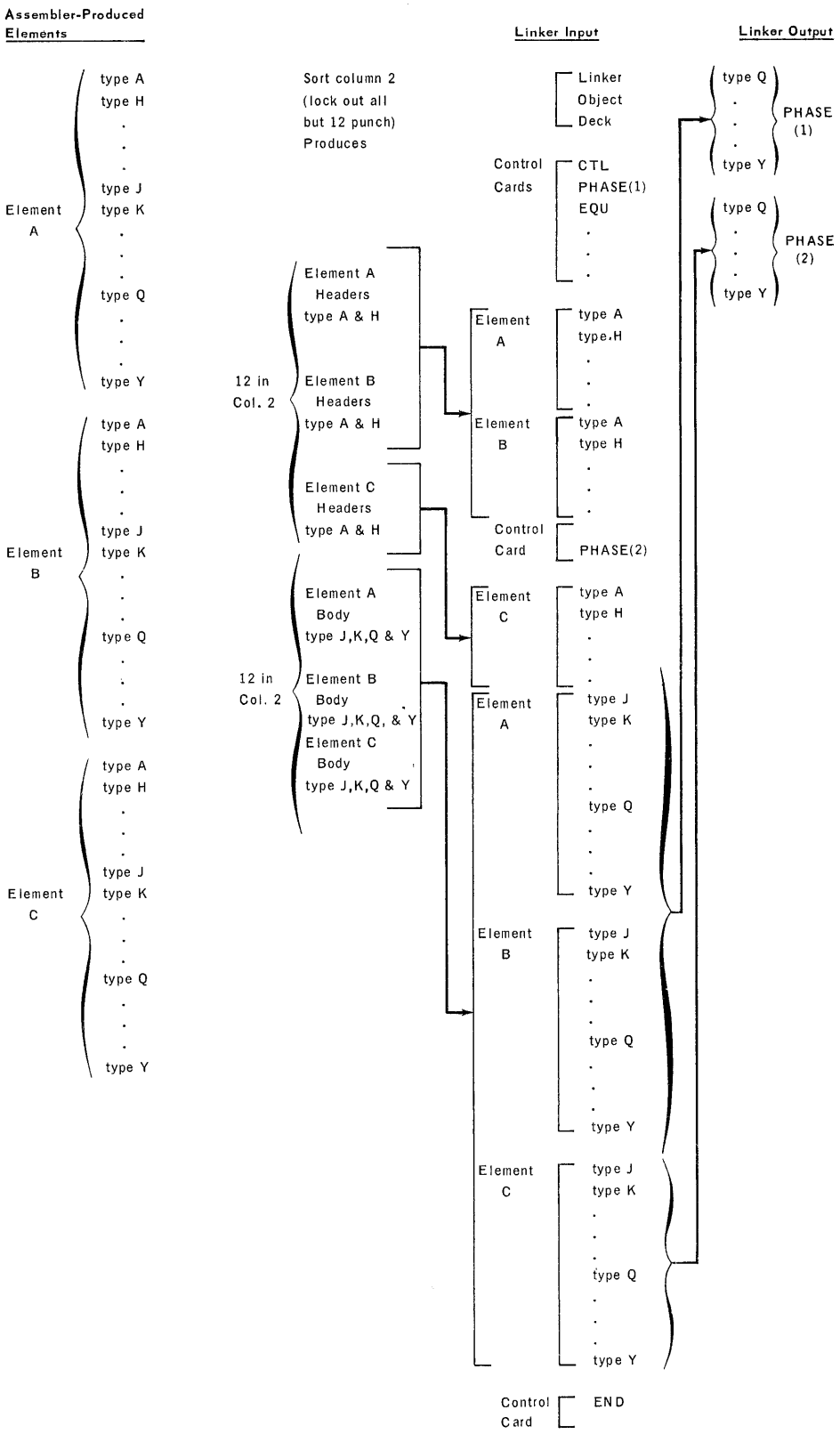


Figure 5–7. Linker Input Deck Sequence for One-Pass Operation

The Linker, as a program, has one phase. The PHASE control card used when the Linker is being linked with the Card Load Routine, LD, is as follows:

Label	Operation	Operand
	PHASE	LNKR,410,A

The PHASE card, when linking with the Card Load Routine (LDCC), is:

Label	Operation	Operand
	PHASE	LNKR,500,A

The EQU cards for the linking operation follow the PHASE card.

The first element in the input deck must be the Card Load Routine. The program name for the routine is LD if the online card reader is used to load the Linker being linked. If the Card Controller is used, the name for the Card Load Routine is LDCC.

The last element in the input deck must be the Linker element which has the program name, LNKR. The elements between the Card Load Routine and LNKR may be in any order. The elements are as follows:

- Exec I, which is named EXEC.

- Card Read Routine. This is generated from a call on either the DTFCR macro for the card reader or the DTFCC macro for the card controller. In either case:

  a. The "filename" must be RDR.

  b. CNTL=YES is not required.

  c. The MODE parameter is specified as MODE=CC.

  For DTFCR, the following are also required:

  a. SENT=NO

  b. IOA1=IOA1.

  For DTECC, specification of the CHAN parameter is also required.

- Input Translation Table. The Linker reads input cards in compressed code. If it identifies a card as a control card, it must translate the card before processing it. To allow the user to punch control cards in any card code he wishes, the Linker uses a user-supplied translation table to effect the translation. The Linker assumes that the result of the translation is EBCDIC. The user-supplied input translation table must be labeled TRN. The Hollerith-to-EBCDIC translation table, labeled TBRD, supplied by the Univac Division, can be used when control cards are punched in Hollerith. In this case, the following EQU card

LABEL	OPERATION	OPERAND
TRN	EQU	0,TBRD

  should be included with the other EQU cards in the input deck.

- Card Punch Routine. This is generated from a call either on the DTFRP macro or on the DTFRW macro for the online serial read/punch or for the row read/punch, respectively. If the online serial read/punch is used, the PUNR=YES parameter specification is made. If the row punch is used, the CHAN parameter must be specified. For both macros:

  a. The "filename" must be RPP.

  b. CNTL=YES is not required.

  c. The MODE parameter is specified as MODE=CC.

  d. The OUAR parameter is specified as OUAR=OUP.

  e. The TYPF=OUTPUT parameter specification is used. Therefore, the EOFA, INAR, ITBL, and ORLP parameters are not specified.

  f. The OTBL parameter is not required because Linker output is punched in compressed code.

- Printer Routine. This routine is generated from a call on the DTFPR macro. For this macro:

  a. The "filename" must be PRNT.

  b. The BKSZ parameter is specified as BKSZ=96.

  c. CNTL=YES is not required.

  d. PROV=YES is used. Consequently, a paper loop containing form overflow and home paper punches, at appropriate points, must be in the printer when the Linker is run.

e. The FONT parameter specified should agree with the print bar used in the printer during Linker execution. The OTBL parameter must be specified if a 48-character bar is used (FONT=48). In this case, an EBCDIC to 48-character print code translation table must also be included in the Linker input deck. If the table TBPR, supplied by the Univac Division, is used, the OTBL=TBPR parameter must be specified when generating the printer routine.

f. PRAD may be specified as one or two, as desired.

The format for the END control card used when the Linker is being linked is as follows:

Label	Operation	Operand
	END	O, BEGN

For convenience, blank cards may be used to separate elements in the input deck since the Linker ignores blanks.

5.7. CARD OUTPUT FROM THE LINKER

The Linker produces two types of output cards:

Type (col. 2)	Name	Caused By
Q	Text card	Input Q cards or REP cards.
Y	Transfer card	See 5.3.4.

5.7.1. Type Q Cards

The Q cards contain the program in loadable absolute form. They are produced when enough contiguous input text is available to fill an output Text card, when text is not contiguous, or when a REP card is processed. An input program could consist of REP cards and no Q cards. This permits the programmer to write short programs in hexadecimal to be linked without assembling.

The format for a Q card is as follows:

Type (Hollerith Q)	Column 2
Card Length	Column 3
Load Address	Column 5–6
Hole Count	Column 7
Absolute Program	Column 11–72 (depending of column 3)

5.7.2. Type Y Cards

The Y cards occur as separators between the loadable phases (overlays) and at the end of the last phase. The Y cards cause the Loader to stop loading and to transfer control to the transfer address in the Y card.

The format for a Y card is as follows:

Type (Hollerith Y)	Column 2
Card Length	Column 3
Hole Count	Column 7
Card Count of Preceding Phase	Column 12–13
Transfer Address	Column 15–16

NOTE: All Linker output cards have the first three characters of the program name (from the first input PHASE card) in column 73–75, a phase identification in columns 76 and 77, a serial number in columns 78–80, and a load key (12–2–9) in column one. The phases are numbered sequentially in hexadecimal. The program name and serial number are punched in Hollerith.

5.8. LINKER MAP

The Linker map is printed during a one-pass Linker run or during the second pass of a two-pass Linker run. The map contains a line for each Linker Control Card and for each of the following Assembler-produced cards:

Card Type	Card Name	Generated in assembly by
A	Element Definition	START Directive
H	External Definition	ENTRY Directive
J	Program Reference	START Directive
K	External Reference	EXTRN Directive
Y	Transfer	END Directive

No line is printed for type Q (Text) cards.

5.8.1. Linker Map Print Lines

For a type A card, the printed line contains (in order from left to right) the load location, in hexadecimal, assigned to the element name, the card type identification code (A), the Assembler-assigned ESID number, the element start address, the element name, the element length, and the output phase (overlay) number.

For a type H card the printed lines contain the value, in hexadecimal, assigned to the externally defined symbol, the card type identification code (H), the Assembler-assigned value of the externally defined symbol, and the name of the externally defined symbol.

For a type J card the printed lines contain the load location, in hexadecimal, assigned to the element name, the card type identification code (J), and the element name. On the line for the first type J card of a phase, the phase number is printed following the element name.

For a type K card the printed lines contain the value assigned to the externally referenced symbol, the card type identification code (K), the Assembler-assigned ESID number, and the name of the externally referenced symbol.

For the first type Y card following a PHASE card and containing a start address, the printed line contains the start address and the card type identification code.

The type Q cards are not printed. For each input element, the type Q cards immediately precede the type Y card.

At the left of each PHASE, EQU, and END control card line is printed the value specified on the card. For the PHASE control card, the value printed is the initial storage address as specified in the card. For the EQU card, the value is that of the symbol defined by the EQU card. For the END card, the value is the specified transfer address.

If a line is flagged with an error message, the error message precedes the card type identification code.

5.8.2. Linker Map Error Messages

TBL END	Too many external references for the combined table area to handle.
SHORT	A PHASE card has missing information. The PHASE card may be partially processed by the Linker. Punching is discontinued.
NO SYMB	An undefined symbol was contained in the operand of a PHASE, EQU, or END card. Partial processing of the erroneous card may occur. Punching is discontinued.
UNEQU	Label field (columns 1–4) on an EQU card was previously defined differently. The value printed on the left is the previously defined value. The new value is ignored. Punching continues.
ESIDX	The ESID number (column 8) on a type A card is not 01. The name is used by the Linker but values are not. Punching is discontinued.
NO DEF	The symbol on a type J or K card was not defined by a type A (Element Definition) or type H (External Definition) card from assembly or by a manually entered EQU to the Linker. The card is partially processed by the Linker. Punching is discontinued.

EXT VAL	The external value defined by an External Definition (type H) card is not equal to the value previously defined for the symbol. The value printed on the left is the previously defined value. The new value is ignored. Punching is discontinued.
CCOUNT	The card count on a Transfer (type Y) card does not agree with card count of the preceding element. Punching continues.
X	An output load address has exceeded the object memory size specified by q of the CTL card.

5.9. LINKER CONSOLE DISPLAYS

DISPLAY	REASON FOR STOP	ACTION
1F03	Invalid card type.	Press START to ignore card.
1F04	After a Y card, no J card was found preceding the next K or Q input card.	Press START to ignore card.
1F05	Input card hole count error.	The card on which the error occurred is the second one from the top of the output stacker. To reread the error card, place it and all cards that follow it (including the card in the wait station of the reader) at the bottom of the deck in the input hopper, manually feed a card, and press the START button.
1F06	An external reference has been made to a label not defined by a K card. (The K card could have been lost or misplaced in the input deck.)	Check the Linker map against the Assembly listing, re-order the card deck appropriately, and start the Linker operation over again.
1F0F	First pass is finished.	Press START to begin last pass.*
1FFF	Last pass is finished.	Press START to begin new Linker run.

* Note that, to start the last pass, the input deck must be removed from the hopper and once more placed in the input hopper. If the Card Controller is being used as the input device, at the end of the first pass the last two cards of the input deck are still in the device and must be run out by pressing the PRI UNLOAD button twice.

APPENDIX A. PREASSEMBLY MACRO PASS

A1. GENERAL DESCRIPTION

The Preassembly Macro Pass of the UNIVAC 9200/9300 Card System is used in conjunction with the Assembler to promote ease and efficiency in preparing programs for execution on the UNIVAC 9200/9300. A schematic of the Preassembly Macro Pass is shown in Figure A–1.

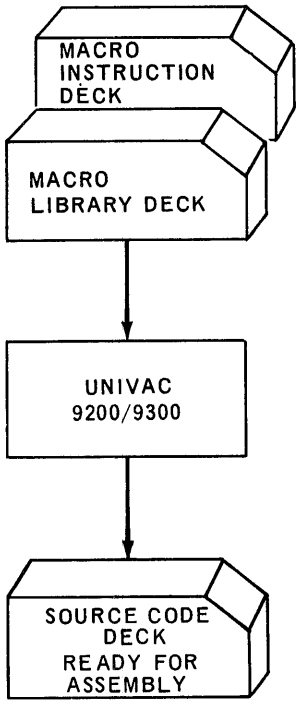


Figure A–1. Schematic of Preassembly Macro Pass Operation

The macro library is a card deck in which the macros in the library are punched in a compressed form to minimize both library passing time and memory storage space. The macro library is read in first and is stored in memory. Then, the card deck of macro instructions is read in. This deck contains the parameters and controls required to generate a source code deck in Assembler format. The output deck represents the selected library routines modified as instructed. The source code deck may be combined with user source code cards and assembled as one element, or it may be assembled as a separate element and linked with other relocatable elements to make up a program.

Because it is a card deck, the library is separable; only those routines called for during the operation of a particular Preassembly Macro Pass need be in the library for storage. The Preassembly Macro Pass ignores the presence of any blank cards in both the macro library and the macro instruction deck.

A2. MACRO INSTRUCTION FORMAT

A macro instruction is similar in form to a source code instruction; it has a label (optional), an operation code, and an operand consisting of one or more expressions separated by commas. The prime difference is that the macro instruction causes the generation of a series of source code instructions representing a number of Assembler operations; whereas a source code instruction causes the Assembler to do one specific operation.

The format for a macro instruction is as follows:

LABEL	OPERATION	OPERAND
label	operation	$P_1,P_2,P_3,\ldots,P_n,N_1=P_{n+1},N_2=P_{n+2},N_3=P_{n+3},\ldots,N_m=P_{n+m}$

The label may be any symbol, but is not necessarily assigned the current value of the location counter. The operation is the name of the macro definition describing the pattern of the code to be included. The operand, P_1 through P_{n+m}, is a sequence of expressions specifying parameters. P_1 through P_n are called positional parameters. P_{n+1} through P_{n+m} are called keyword parameters. A macro instruction may have positional parameters only, keyword parameters only, neither, or both positional and keyword parameters.

A2.1. Parameters

There are two types of parameters; positional and keyword.

- Positional Parameters. All positional parameters must be specified before any keyword parameters may be specified. The order of the expressions in the operand determines the order of the parameters specified. Parameter specifications are separated by commas. When a positional parameter specification is omitted, the comma must be retained to indicate the omission. Thus, if a macro has three positional parameters and the second one is not specified, the operand appears as follows:

$$P_1,,P_3$$

If the third parameter is not specified, and the second is specified, the operand is written:

$$P_1,P_2$$

Thus, no trailing commas need be present.

- Keyword Parameters. The specification of a keyword parameter is as follows:

$$N=P$$

where: N is the name of the parameter (any symbol of four or fewer characters is a legitimate keyword name).

P is the parameter specification (a value or a character string).

Keyword parameter specifications are separated by commas; however, the comma need not be retained if the specification is omitted. There must be a comma between the last positional parameter and the first keyword parameter. The order of the keyword parameter specifications is not significant. For example, if a macro has three keyword parameters, the operand of the macro instruction might be:

$$N_1=P_1,N_2=P_2,N_3=P_3$$

or

$$N_2=P_2,N_1=P_1,N_3=P_3$$

and so on.

A macro may have positional and/or keyword parameters with commas separating the specifications. For example, the operand of a macro instruction with three positional and two keyword parameters might be as follows:

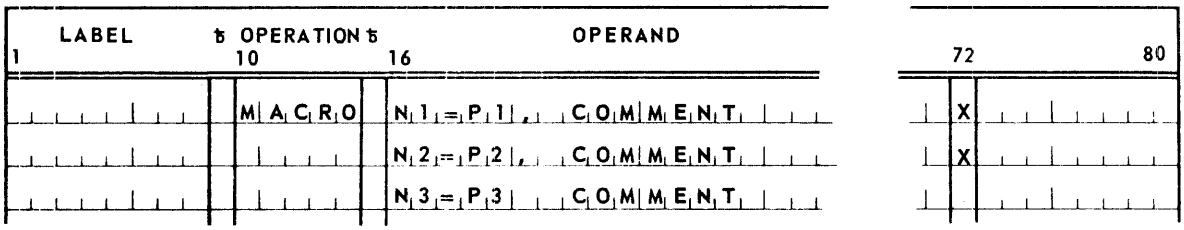
$$P_1,P_2,P_3,N_1=P_4,N_2=P_5$$

The number of parameters which may be specified with one macro instruction depends on how much space is required to store the specifications. One macro instruction may normally specify as many as 50 parameters in its operand. When the operand overflows the space provided in one record, provision is made to continue the operand in the following record by putting a nonblank in column 72. The continuation of the operand begins with column 16. The Macro Pass searches for a continuation record as soon as one of the two following events occurs:

- Information is taken from column 71 of the current record.
- A comma followed by a space is detected in the current record.

Columns 1 through 15 of a continuation record must be blank.

If the information in a record is terminated prior to column 71 by means of a comma followed by a space, comments may be written after the space. For example, a macro instruction with three keyword parameters might be written as follows:



The specification of a parameter may not contain an equal sign or a comma, except when it occurs between apostrophes.

A3. WRITING MACRO DEFINITIONS

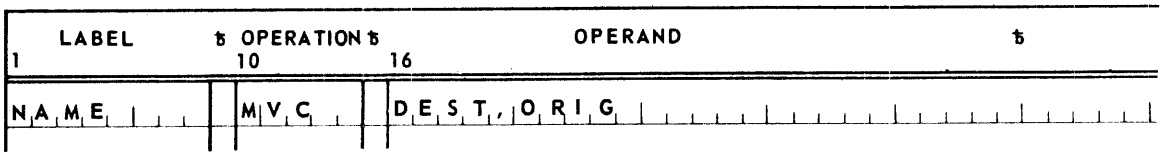
The routines for the macro library are written in standard Assembler source code. They are then passed through a special run (the Compressor) to compress them into the library form expected by the macro pass. To distinguish one macro from another in the library, three directives are used: PROC, NAME, END.

A3.1. PROC Directive

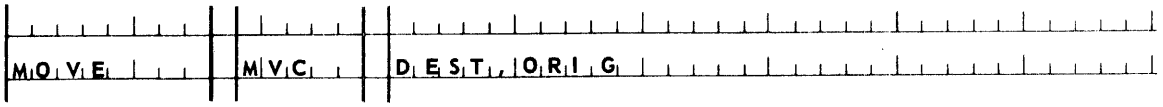
The first source code statement of a macro definition is a PROC directive, which has the following form:

LABEL	OPERATION	OPERAND
(optional)	PROC	(optional)

The label may be any symbol, but it is optional. When used, the label in the macro instruction calling on the macro is substituted for the PROC label, wherever the PROC label appears in the macro. For example, suppose the symbol MOVE were specified for the label of a macro instruction, that the label of the PROC directive of the associated macro was NAME, and that the macro contained the following line of source code:



Then, the source code generated by the macro definition would appear as follows:



If the PROC directive does not have a label, but the macro instruction does, the label of the macro instruction remains undefined.

A3.2. NAME Directive

The second line of a macro definition must be a NAME directive, which has the form:

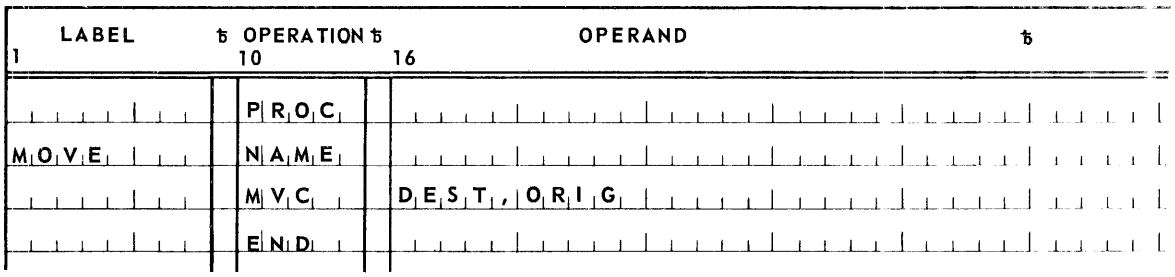
LABEL	OPERATION
label	NAME

This is the call name for the macro and is the name that is specified in the operation field of the macro instruction. The name may have as many as five characters, the first of which must be alphabetic; the others alphanumeric.

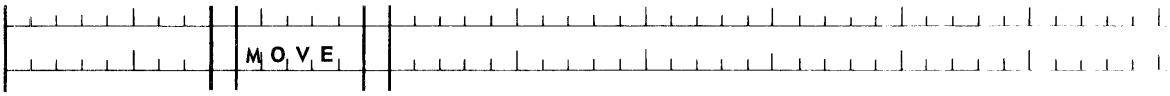
A3.3. END Directive

The end of a macro definition is indicated by an END directive. It has no operand and requires no label.

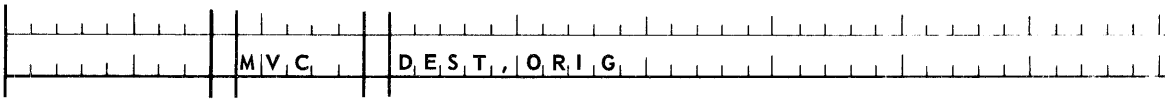
If the following macro is in the library:



then the macro instruction:



is equivalent to the source code instruction:



Note that none of the macro directives (PROC, NAME, END) are produced as output of the macro pass.

A3.4. Comments

Comment lines may be inserted between the last NAME directive and the END directive in a macro definition. Each of these comment cards must have an asterisk in column 1 and must end at or before column 67. (Columns 68–71 are used by the Preassembly Macro Pass for card sequencing.)

Comments are not permitted on a PROC line if the PROC directive does not specify any parameters. However, they may be written after the last parameter specified in the operand and must be separated from the operand by at least one blank.

Comments may be written on Assembler source code lines, but they are not reproduced by the Macro Pass when the source code lines are generated.

A4. INCORPORATING PARAMETERS INTO MACRO CODING

The operand of a PROC directive, when used, has the following form:

$$p,n,N_1,N_2,N_3,\ldots,N_m$$

The first expression (p) in the operand is a symbol used to address the parameters for the macro. The second expression (n) is the number of positional parameters associated with the macro. The series ($N_1,\ldots N_m$) are the names of the keyword parameters. Any symbol of four or fewer characters is a legitimate keyword name. Listing the keyword parameters in this way makes them, in effect, positional parameters to the macro. For example, suppose the PROC directive has the following form:

OPERATION	OPERAND
PROC	p,3,N1,N2,N3

The macro has three positional parameters, P1, P2, and P3. It also has three keyword parameters, N1, N2, and N3. Thus, the keyword parameters become, in effect, positional parameters P4, P5, and P6.

The value specified for a parameter is substituted in the macro coding for an expression of the following form:

$$p(n)$$

where: p is the first expression in the PROC directive operand.

n is the decimal number of the positional parameter. The first has a number of one, the second, two and so forth.

For example, if the following macro is in the library:

LABEL (1)	OPERATION (10)	OPERAND (16)
	PROC	P,0,DEST,LGTH,ORIG
MOVE	NAME	
	MVC	P(1)(P(2)),P(3)
	END	

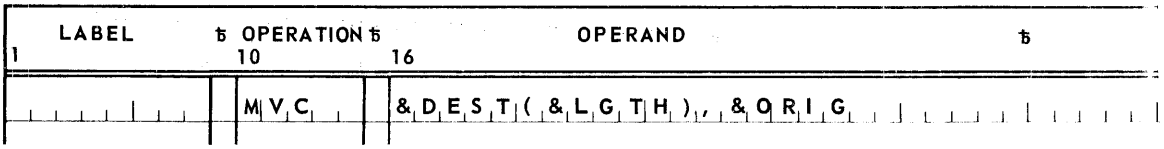
then the macro instruction:

LABEL	OPERATION	OPERAND
	MOVE	DEST=OUT,LGTH=16,ORIG=IN

is equivalent to the source code instruction:

LABEL	OPERATION	OPERAND
	MVC	OUT(16),IN

A keyword parameter may also be addressed by writing its name preceded by an ampersand. Thus, the MVC instruction within the macro definition of the previous example could also have been written:



If a parameter value is unspecified, it will be replaced by the value zero.

When the operand space overflows the space provided in one record, provision is made to continue the operand in the following record by putting a nonblank in column 72. The continuation of the operand begins with column 16. The Macro Pass searches for a continuation record as soon as one of the two following events occur:

- Information is taken from column 71 of the current record.
- A comma followed by a space is detected in the current record.

Columns 1 through 15 of a continuation record must be blank. If the information in a record is terminated prior to column 71 by means of a comma followed by a space, and if column 72 contains a nonblank, then all columns between the comma and column 72 must be blank.

A5. NAME STATEMENT

More than one NAME statement may follow the PROC statement of the macro. However, all the NAME statements in a macro must immediately follow the PROC statement. Each such NAME statement specifies a different name for the same macro.

The object of giving a macro more than one name is to permit reference to different versions of the procedure embodied in the macro. The versions are distinguished within the macro by means of the operands of the NAME statements.

Only one expression may appear in the operand of a NAME statement and may be assigned a value ranging from 0 through $2^{16}-1$. The expression may be a decimal number, a hexadecimal number, or a character expression of the form C'xx'. Any value greater than $2^{16}-1$ will be truncated with no error indication given. This expression is essentially a parameter of the macro; it may be addressed in the macro as:

p(O)

where p is the first expression in the PROC statement operand; consequently, it may be used to distinguish between versions of a macro.

For example, if the following macro is in the library:

```
LABEL      OPERATION      OPERAND
1          10             16
           PROC           P
MV4        NAME           4
MV8        NAME           8
           MVC            DEST(P(O)),ORIG
           END
```

then the macro instruction

```
           MV4
```

would produce the source code

```
           MVC            DEST(4),ORIG
```

while the macro instruction

```
           MV8
```

would produce

```
           MVC            DEST(8),ORIG
```

If a NAME statement has no operand, the parameter p(O) is assigned a value of zero.

If a macro has no parameters and it makes no reference to the operand of any of its NAME statements, then its PROC statement has no operand.

A6. CONDITIONAL MACRO PASS INSTRUCTIONS

The Macro Pass recognizes certain directives which can:

- exclude lines of coding from the output of the Macro Pass,
- include a set of lines in the output of the Macro Pass more than once
- establish and alter values which may be used to determine whether a set of lines shall be included or excluded.

These directives are provided to control the pattern of coding generated, based on the parameters supplied in the macro instruction.

A6.1. DO and ENDO Directives

A DO directive controls the inclusion or exclusion of the lines following it, up to its associated ENDO directive. For example, in the following sequence of coding:

LABEL	OPERATION	OPERAND
	DO	1
		2
	DO	3
		4
		5
	ENDO	6
		7
		8
	ENDO	9

the first ENDO directive is associated with the second DO directive, the second ENDO directive with the first DO directive. In other words, DO and ENDO directives are paired to produce nests. Thus, the first DO directive controls lines 2 through 8, and the second DO directive controls lines 4 and 5. DO's may be nested to a depth of 10.

The operand field of a DO statement contains a single expression. If the value of this expression is greater than zero, it represents the number of times the lines controlled by the DO statement will be included in the output. Otherwise, these lines will not appear. For example, if the following macro is in the library:

LABEL (1)	OPERATION (10)	OPERAND (16)
	PROC	P,0,ACT
MOVE	NAME	
	DO	P(1)
	MVC	DEST,ORIG
	ENDO	
	END	

then the macro instruction

LABEL	OPERATION	OPERAND
	MOVE	ACT=1

would produce the instruction

LABEL	OPERATION	OPERAND
	MVC	DEST,ORIG

in the output of the macro pass; whereas, the macro instruction

LABEL	OPERATION	OPERAND
	MOVE	ACT=0

would not produce the instruction.

Note that the macro instruction

LABEL	OPERATION	OPERAND
	MOVE	

would also cause the suppression of the instruction.

A DO statement may have a symbol in the label field. This symbol may be used only in the statements controlled by the DO. Its value is one the first time these statements are generated, two the second time they are generated, and so on. If the DO is under the control of another DO, and is activated again, the count begins at one again. Only two DO directives within a nest of DO's may be labeled.

A6.2. GOTO and LABEL Directives

The form of the LABEL statement is

LABEL	OPERATION	OPERAND
symbol	LABEL	not used

The symbol in the label field of the LABEL statement is not defined in the usual sense. It may be used only in the operand field of a GOTO statement.

The GOTO directive is used to direct the Macro Pass to another point in the macro definition in its production of source code.

The form of the GOTO statement is

LABEL	OPERATION	OPERAND
not used	GOTO	symbol

The symbol in the operand field must be the label of a LABEL statement.

If the following macro definition is available:

LABEL (1)	OPERATION (10)	OPERAND (16)
	PROC	P,,O,FOUR
MOVE	NAME	
	DO	P(1)
	MVC	DEST(4),ORIG
	GOTO	END
	ENDO	
	MVC	DEST(8),ORIG
END	LABEL	
	END	

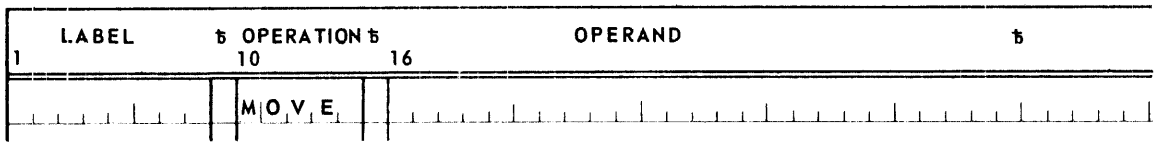
then the macro instruction

LABEL	OPERATION	OPERAND
	MOVE	FOUR=1

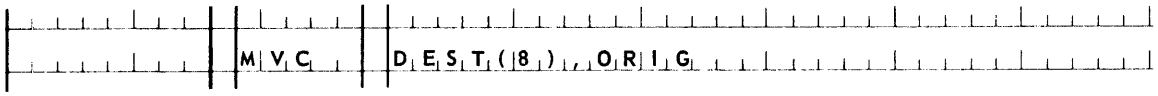
would produce the instruction

LABEL	OPERATION	OPERAND
	MVC	DEST(4),ORIG

while the macro instruction



would produce the instruction



For any GOTO statement, the corresponding LABEL statement must appear in the same macro definition as does the GOTO statement. If a GOTO is within the range of statements under control of a DO statement, but the addressed LABEL statement is not, then execution of the GOTO will terminate the DO whether the DO count, as expressed in the operand field of the DO statement, is exhausted or not.

A6.3. Set Variables

A set variable is a symbol to which a value is assigned during the generation of the code corresponding to macro instructions. Unlike an ordinary symbol, the value assigned to a set variable may be altered during the course of the Macro Pass. A set variable may be either a local or a global variable. A global variable, once declared and given a value by a SET statement, remains defined throughout the Macro Pass and retains the same value until that value is changed by another SET statement for that variable. A local variable is defined only within the macro definition within which it is declared. A value assigned to a local variable within one macro definition does not affect its value within any other macro definition in which a local variable with the same name is declared.

Before a set variable may be set, it must first be declared by a GBL or a LCL directive. The symbol naming a set variable must consist of four characters.

A6.3.1. GBL Directive

The GBL statement has the form

LABEL	OPERATION	OPERAND
not used	GBL	&G%xx

where: $00 \leq xx \leq 49$

The symbol in the operand field of the GBL statement is declared to be the name of a global set variable. The symbol must consist of four characters, not counting the ampersand, and take the form shown above.

A6.3.2. LCL Directive

The LCL statement has the form

LABEL	OPERATION	OPERAND
not used	LCL	&L%xx

where: $00 \leq xx \leq 49$

The symbol in the operand field of the LCL statement is declared to be the name of a local set variable. The symbol must consist of four characters, not counting the ampersand, and take the form shown above.

A6.3.3. SET Directive

The SET directive is used to assign a value to a set variable. The form of the SET statement is

LABEL	OPERATION	OPERAND
symbol	SET	expression

The symbol in the label field is the name of the global or local set variable to which a value is being assigned; the expression in the operand is the value to which the set variable is to be set. The value of the expression may range from 0 through $2^{16}-1$. Until a GBL or LCL variable is set by a SET directive, it has the value zero. Once it has been set to a specific value by a SET directive, the set variable retains that value until it loses its declaration, or until it is set to another value by another SET directive. Declaring a set variable does not affect its value.

A set variable may be addressed by writing its name preceded by an ampersand.

A6.3.4. Relational and Logical Operators

Expressions in the operands field of machine instructions and most Assembler directives contain only the arithmetic operators, add and subtract. The expression in the operand field of a SET or DO statement may contain these arithmetic operators; and in combination with these, it may also contain the following:

- Arithmetic Operators
  - multiply (*)
- Relational Operators
  - greater than (>)
  - equal (=)
  - less than (<)

- Logical Operators
  - logical-and (**)
  - logical-or (++)

The relational operators compare two unsigned binary numbers. The value of a relational expression is 1 if the relation is satisfied; otherwise, it is 0. Thus, if CHAN is the name of a keyword parameter, the expression

&CHAN=5

would have the value 1 if 5 had been specified as a value of the parameter CHAN; otherwise, it would have the value 0.

The logical operators treat a value as a string of 16 bits and produce a 16-bit value. The "logical and " operator corresponds to the NC instruction and the "logical or" corresponds to the OC instruction.

The precedence relation of the various operators in decreasing order is as follows:

1. *
2. + –
3. **
4. ++
5. > = <

Parentheses may be used to override this precedence relation.

A6.3.5. Character Values

A character value is a string of up to 13 characters enclosed in single apostrophes. Apostrophes within the character string must appear as pairs of successive apostrophes. Ampersands within the character string must appear as paris of successive ampersands.

A6.3.6. Use of Character Values

A character value may appear as an operand of a relational operator. The following rules apply:

- A numeric value is greater than a character value.
- A character value is greater than any shorter character value.

A series of characters enclosed by separators is considered a character value. The following are separators:

(* >

) b =

+ ** <

– ++ ,

Such a character value may have a maximum length of 15 (including leading and ending apostrophes when they occur).

A character string in the macro definition which:

1. Matches a set variable name except for the absence of the leading ampersand, *and*
2. Is not enclosed in apostrophes,

will be accepted as the name of that set variable. This type of character string must be enclosed in apostrophes to be treated as a character value.

If the operand field of a SET or DO statement contains an expression which could not otherwise be evaluated, that expression will be treated as a character value even though it is not enclosed in apostrophes.

The following is an example of the use of a local set variable. If the following macro is in the library:

```
LABEL      OPERATION      OPERAND
1          10             16
           PROC           P1,O,ACT
MOVE       NAME
           LCL            &L%00
&L%00      SET            P(1)='YES'
           DO             &L%00
           MVC            DEST,ORIG
           ENDO
           END
```

then the macro instruction

```
           MOVE           ACT=YES
```

would produce the source code instruction

```
           MVC            DEST,ORIG
```

while any other form of the MOVE macro instruction would suppress production of the source code instruction.

The following is an example of the use of a global set variable. Assume the following two macros are in the library:

```
LABEL     OPERATION     OPERAND
1         10            16
          PROC          P1,O1,ACT
GIVE      NAME
          GBL           &G%00
&G%00     SET           P(1)='YES'
          DO            &G%00
          MVC           DEST1,ORIG
          ENDO
          END
          PROC
TAKE      NAME
          DO            &G%00
          MVC           ORIG1,DEST
          ENDO
          END
```

If the only macro instructions in the source statements for a particular assembly are the following:

```
          GIVE          ACT1=YES
          TAKE
```

in the order shown, then the following source code would be produced:

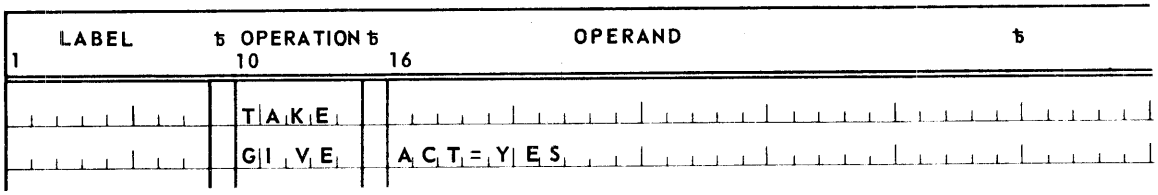
```
          MVC           DEST1,ORIG
          MVC           ORIG1,DEST
```

If the only macro instructions in the source statements for a particular assembly are the following:

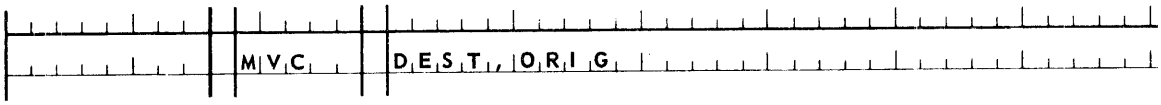
```
          GIVE
          TAKE
```

no source code would be produced.

If the only macro instructions in the source statements are the following:



then the following source code would be produced.



Thus, the value of a global set variable is a function of the order of the macro instructions in the source statements.

A7. CONTINUATION CARDS

Continuation cards are allowed on all cards in a macro definition except comment cards.

The following rules will cause a continuation mark to be searched for:

- A comma followed by a space is detected.
- A comma in column 71 is detected.
- A separator in column 71 is detected (except on a PROC directive card).

Columns 1 through 15 of a continuation card must be blank. Column 16 of a continuation card must contain the next valid character of the operand.

A8. LABELS USED IN UNIVAC PRODUCED MACROS

It should be noted that if the output of a macro pass is to be combined with user source code cards and assembled as one element, any symbol used as a label in a source code instruction produced by the macro pass may not be used as a label in the user's own code. To avoid the necessity of the user checking a list of symbols used in Univac written macros, a special feature has been incorporated into the Assembler to allow all such symbols to incorporate a question mark as their second character.

A9. MACRO INSTRUCTION DECK

Regardless of the order of the macro routines in the library, the macro instruction deck may be in random order with respect to the library, and a particular macro may be referenced as many times as desired. However, the order of the macro instructions does determine the sequence of the source code instructions generated as output of the macro pass operation.

During the macro pass operation, any cards in the macro instruction deck that are not macro instructions referring to macros in the macro library are reproduced unchanged in the output source deck.

The macro pass operation recognizes the end of the macro instruction deck by means of an END card which it reproduces and includes at the end of the output source code deck.

A10. MACRO PASS OUTPUT FORMAT

The Preassembly Macro Pass can produce four types of output cards. The formats for these cards are presented in the following paragraphs.

A10.1. Source Code Card Format

Columns	Contents
1–4	Label
5	Blank
6–10	Operation
11	Blank
12–67	Operand Field (No Comments)
69–71	Card number within the macro
72–73	Blank
74	Asterisk
75–76	Blank
77–80	Columns 77–80 of last parameter card

A10.2. Macro Instruction Card Format

Columns	Contents
1	Asterisk
2–72	Columns 1–71 of the input card
73–80	Columns 73–80 of the input card

A10.3. Comments Card Format

Columns	Contents
1	Asterisk
2	Blank
3–68	Comment
69–80	Same as standard source code format

A maximum of 65 characters of the original comment line will be preserved and all leading and terminal blanks will be dropped.

A10.4. Error Card Format

There are certain rules of writing macro definitions which may have been broken. If they have been broken, a card will be punched containing the following information:

Columns	Contents
1	*
2–4	Blank
5	E
6	Error Code (see below)
7–68	Blank
69–80	Same as standard source code format

The error codes are:

CODE	MEANING
1	Too many right or left parentheses occur in a DO or SET directive.
3	The first character of a DO or SET directive is a separator other than a minus sign, a left parenthesis, or an apostrophe.
9	The information to be turned out as one source code statement will not fit on one card.

A11. MACRO PASS CONSOLE DISPLAYS

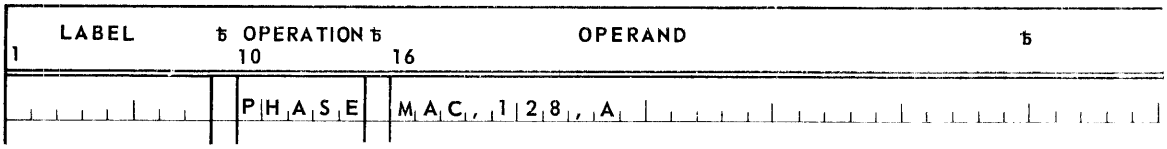
Display	Reason for Stop	Action
01FF	Macro Library is loaded.	Press the START switch to process the macro instruction deck.
1FFF	Current macro instruction deck has been processed (END card has been read).	Press the START switch to process another set of macro instructions.
0111	Card count check failure while loading the Macro Library. After the card read device has been cleared, the card in error will be the third card from the last in the output stacker.	Remove all cards, beginning with the error card, from the read unit. Determine reason for error (shuffled cards, etc.) and place corrected deck in the input stacker. Feed one card, press the START switch. The next card to be read will be examined to see if it is in correct sequence. If it is, normal processing will continue. If it is not, it will be assumed that a new macro is being loaded. The card count expected will be card number 1 and the previous macro will be ignored.

Display	Reason for Stop	Action
0122	Hole count check failure while loading the Macro Library. After the card read device has been cleared, the card in error will be the third card from the last in the output stacker.	Remove all cards, beginning with the error card, from the read unit. Replace in read unit, feed one card and depress the START switch. If the same stop occurs again, the card has been incorrectly punched. Restart with different library.
0177	Not a library card. Column 1 does not contain a 12–3–9 punch and the card is not a blank card. After the card read device has been cleared, the card in error will be the third card from the last in the output stacker.	Remove error card. Place all cards following error card back in input hopper. Feed one card. Depress START switch to continue normal processing of the Macro Library.
0100	An end sentinel card has been detected (/*) but the card preceding is not a 'Z' type card. After the card read device has been cleared, the end sentinel card will be the third card from the last in the output stacker.	Remove all cards, beginning with the end sentinel card, from the reader. Find the 'Z' card and place in reader followed by rest of library deck. Feed one card and and depress START switch. If the card now read is a 'Z' card, normal processing will be resumed. If it is not a 'Z' card, the current macro will be destroyed and it will be assumed that a new macro is being processed.
0133	Current library will not fit in computer memory.	Remove rest of current macro (up to first 'Z' card) deck. Place rest of library back in reader and depress the START switch. Processing will resume with processing of a new macro which will overlay start of the previous one. Another choice is to break down the Macro Library deck into smaller libraries and reload.

A12. LINKING THE MACRO PASS

The Macro Pass is composed of several separately assembled parts which must be combined with a Linker run in order to produce a loadable Macro Pass. The following description of how to link the Macro Pass assumes a two-pass Linker operation. The Macro Pass can operate in an 8K UNIVAC 9200/9300 system.

The Macro Pass is a single-phase program. The PHASE control card to be used in linking the Macro Pass is as follows.



EQU cards for the linking operation should follow the PHASE card. Following the EQU cards should be the relocatable object code decks of the elements making up the Macro Pass.

The first element of the input deck must be the Card Load Routine. If the online card reader is going to be used to load the Macro Pass, the name of the Card Load Routine is LD. If the Card Controller is going to be used, the name of the Card Load Routine is LDCC.

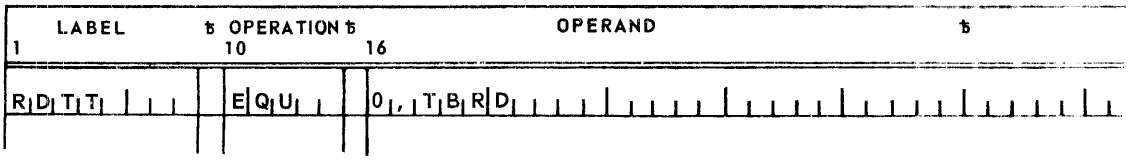
The last element of the input deck must be the Macro Pass element named MP. The following elements may appear in any order between the Card Load Routine and the MP elements.

- A card read routine. This could be either the Systems Programming internal routine for the online serial card reader, named XRDR, or the Systems Programming internal routine for the Card Controller named XRDC. If XRDC is used, then an EQU card of the form

LABEL	OPERATION	OPERAND
RDCN	EQU	n

  should be included with the other EQU cards in the input deck. 'n' is the number of the channel on which the Card Controller is located.

- An input translation table. To allow the user to punch source code in any card code that he wishes, the Macro Pass uses a user-supplied translation table to translate from card to internal code. The Macro Pass assumes that the result of the translation is EBCDIC. The input translation table must be labeled RDTT. If source code cards are punched in Hollerith, then the UNIVAC supplied Hollerith-to-EBCDIC translation table, which is named TBRD, can be used. In this case the following card should be included with the other EQU cards in the input deck:

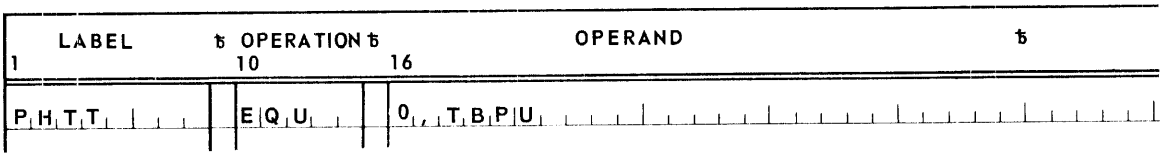


- A card punch routine. This is either the UNIVAC supplied internal routine for the online serial punch, named XPCH, or the UNIVAC supplied internal routine for the row punch, named XPRW. If XPRW is used, then an EQU of the form

LABEL	OPERATION	OPERAND
CHAN	EQU	n

should be included with the other EQU cards in the input deck. 'n' is the number of the channel on which the row punch is located.

- An output translation table. To allow the user to have the output source code of the Macro Pass punched in any card code that he wishes, the Macro Pass uses a user-supplied translation table to translate from internal code to card code. The Macro Pass expects all information to be translated from EBCDIC. The output translation table must be labeled PHTT. If it is desired to have the cards punched in Hollerith, then the UNIVAC supplied EBCDIC-to-Hollerith translation table, named TBPU, can be used. In this case, the following card should be included with the other EQU cards in the input deck:



Blank card separators may be placed between the elements in the input deck, as the Linker ignores blank cards.

The END control card used when the Macro Pass is linked must have the operand field:

0,INIT

A12.1. Operating Instructions

1. Place the Macro Pass deck, the optional CTL card, the compressed library deck including the final /* card, and one blank card in the reader.
2. Load cards.
3. Stop X'01FF'
4. Place macro instructions including the END card in the reader followed by two blank cards.
5. Press the START switch.
6. Stop X'1FFF'. Final stop.
7. The punch output hopper contains the source code cards for assembly.
8. To process another set of macro instructions return to Step 4.

To process a second library, the Macro Pass must be reloaded, since in processing macro instructions, the Macro Pass overlays the routine used to load the library.

A12.2. Control Card

The compressed library deck may be preceded by a control card, which contains CTL in columns 10, 11, and 12, and a decimal number beginning in column 16. The number may take any value between 8191 and 32767. If a control card is present, the decimal number tells the Macro Pass the memory size in which it is to operate. If the control card is absent, or if the value specified lies outside the permissible range, the Macro Pass assumes that it is to operate in a 16K memory.

A13. THE COMPRESSOR

The purpose of the Compressor is to process source coded macro definitions and from these create a binary compressed macro library, which may then be used by the Macro Pass to create a source code deck from macro instructions and the Macro Library.

The Compressor run creates a macro library from macro definitions which may then be used as input to the Macro Pass. One or many macro definitions may be passed through the Compressor at a time.

Each macro definition is terminated by an END card and each library of macro definitions is terminated by an end sentinel card (/* in columns 1 and 2 of the input card).

The Compressor run takes all macro definitions, up to an end sentinel card, and produces both a printed listing and an EBCDIC, binary punched card deck.

The printer output consists of a listing of all input cards, plus an error code wherever the macro definition is incorrect. At the end of each macro definition, a total macro error count is printed and the paper is advanced to a new page.

The card output consists of one or more macro definitions punched in compressed binary format. Each binary card is numbered consecutively within the current macro definition. Each macro within the library causes several binary cards to be punched. These cards begin with a count of one, and contain an identification which is taken from columns 73–77 of the PROC directive of the macro definition. The output macro library is followed by an end sentinel card.

A13.1. Compressed Macro Library Deck Format

All cards in the macro library produced by the Compressor have the following format:

Columns	Contents
1	12–3–9 punch
2	Checksum of columns 3–72 in binary
3	Card type ID; B, C, or Z
4–72	Variable information in binary
73–77	Identification
78–80	Card number in Hollerith (starts with 1 for each macro)

The card number is stored in the numeric portions of columns 78–80. These columns may be overpunched.

End of library is /* in columns 1–2.

A13.1.1. Data Cards

These cards contain the actual representation of the macro in the order it appears in the internal format. In this internal format, each macro definition is preceded by an eight-byte header, containing information used by the Macro Pass to process the macro definition when it is called. When the macro definition is processed by the Compressor, it leaves the first six bytes of this header filled with zeros. In the last card of the macro definition, it supplies the information that the Macro Pass is to insert in these six bytes before processing the macro definition.

The information on a data card is as follows:

Columns	Contents	
3	B	Identifies data card.
4		A number which is one less than the number of characters of macro definition on this card.
5–n		The actual characters of macro definition.
		This information may extend as far as column 72.

A13.1.2. Fixups

Since GOTO references to a label of a LABEL directive may be forward references, all such references to a single label are chained. The operand field of a GOTO receives the address of the last previous reference (relative to the address of the first character of the header of this macro) . The information for fixing these references is contained on special cards after the end of the body of the macro. The format of these special cards is:

Columns	Contents	
3	C	Identifies a fixup card.
4		The number of items of fixup information on this card.
5–72		A variable number of items of fixup information. Each item consists of four bytes of information.
		0,1: relative value of the label.
		2,3: relative address of last reference to the label.

Note: Fixups also occur for other types of information. Some of these may occur in the punched output interspersed with the data cards.

A13.1.3. Header Specification

This is the last card of the representation of a macro. It contains the information required for fixing up the header.

Columns	Contents
3	Z Identifies this card as a header specification.
4	blank
5,6	The total number of bytes occupied by the macro in internal format.
7,8	The relative position of the start of the Name table within the macro.
9,10	The relative position of the start of coding within the macro.

The last card of input to the Compressor must contain /* in columns 1 and 2. The last card of output from the Compressor is such a card.

Several macro libraries may be created from one library by inserting an end sentinel card after any or all 'Z' cards.

One macro library may be created from several by removing all end sentinel cards from each library and combining these decks, followed by an end sentinel card.

A13.2. Error Indications

When the Compressor detects an error in the source code of a macro definition, it flags the erroneous line of source code with a one-character flag. One or more of these flags may occur on a line. The flag is an indication of the error type. Flags and their meaning are as follows:

Ⓐ One of the following has been detected:

- A label of a line other than a macro language directive consists of more than four characters.
- An operation consists of more than five characters.

Ⓑ One of the following rules was broken on the operand field of a DO or SET Line:

- A right parenthesis must be followed by a space, a right parenthesis, or an operator. It must be preceded by a term or a left parenthesis.
- A comma may not appear.
- A left parenthesis must be preceded by an operator, a left parenthesis, or a space. It must be followed by a term, a minus sign, or a left parenthesis.

Ⓒ A character string is greater than 15 characters, including leading and ending apostrophes.

Ⓓ A label which has been defined by a LABEL directive occurs as a label of another line.

Ⓔ One of the following has been detected:

- In parameter references of the type P(n), a right parenthesis is missing, or it is not followed by a space.
- In parameter references of the type P(n), an illegal separator occurs after the left parenthesis.
- The operand of a NAME directive specifies a value of more than $2^{16}-1$.
- On C'. . . ' type constants, the terminal apostrophe is not followed by a separator.
- The explicit length specified in a DC or DS directive is less than one.
- A hexadecimal constant is greater than 16 bytes.
- A location counter reference (*) is not followed by a +, −, space, or comma.

Ⓕ More than ten DO's are nested together.

Ⓖ A comma is followed by a separator in the operand field of a PROC directive.

Ⓘ One of the following has been detected:

- Part, or all of the line has not been processed due to the occurrence of one of the other errors. The other error code will be printed with the I.
- An ENDO occurs without a corresponding preceding DO. The line is ignored.
- The line is not under the control of a PROC directive and will be ignored.

Ⓚ The keyword which is referenced within the macro is not defined in the PROC directive.

Ⓛ More than two DO labels occur within nested DO's.

Ⓜ An ENDO is missing. Each DO must match with an ENDO.

Ⓝ The number of positional parameters defined in the PROC directive is nonnumeric.

Ⓞ One of the following has been detected:

- There is no operation field on this line.
- A NAME directive does not follow a PROC directive or another NAME directive.

Ⓟ The macro is incomplete. An END record does not occur before the next PROC directive or before the end sentinel card.

Ⓣ One of the following has been detected:

- A GOTO directive label consists of more than 4 characters.
- A keyword defined by the PROC directive is more than 4 characters long.
- The name defined on the PROC directive for referencing positional parameters is more than 4 characters long.

Ⓤ A GOTO references an undefined label (LABEL directive is missing).

Ⓧ Some restriction which the Compressor places on the continuation of information from one card to the next has been violated. The restrictions are as follows:

- If an input card has a continuation mark in column 72, the last character from this card must be a separator.
- If a PROC directive card has a continuation mark in column 72, the last character from this card must be a comma.

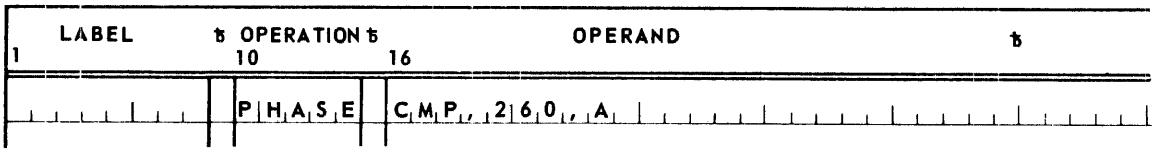
A13.3. Compressor Console Displays

DISPLAY	REASON FOR STOP	ACTION
1111	A GOTO or LABEL directive is being processed and the label table is full.	The Compressor does not have enough memory space to continue processing this macro definition. If the user desires the Compressor to begin the processing of another macro definition, he should remove any remaining cards of the current macro definition from the card read unit, load the next macro definition into the card read unit and depress the START switch. The Compressor will terminate processing of the current macro definition and start processing the next macro definition.
1FFF	The Compressor has either just been loaded or has completed processing a Macro Library.	Depress the START switch to have the Compressor process the next Macro Library.

A13.4. Linking the Compressor

The Compressor is composed of several separately assembled parts which must be combined by a Linker run in order to produce a loadable Compressor. The following description of how to link the Compressor assumes a two-pass Linker operation. The Compressor can operate in an 8K UNIVAC 9200/9300 System.

The Compressor is a single-phase program. The PHASE control card to be used in linking the Compressor is as follows:



EQU cards for the linking operation should follow the PHASE card. These EQU cards define the following labels as indicated:

- BLNK – 64 if a 63/character print bar is to be used when running the Compressor; otherwise, 16.
- FONT – Zero if a 63/character print bar is to be used; otherwise, 128.
- PRTR – Fifteen if a 63/character print bar is to be used; otherwise, zero.
- PRTT – Zero if a 63/character print bar is to be used; otherwise, 0,TBPR.

The EQU cards should be followed by the relocatable object code decks of the elements making up the Compressor. The first element of the input deck must be the Card Load Routine. If the online card reader is going to be used to load the Compressor, the name of the Card Load Routine is LD. If the Card Controller is going to be used, the name of the Card Load Routine is LDCC.

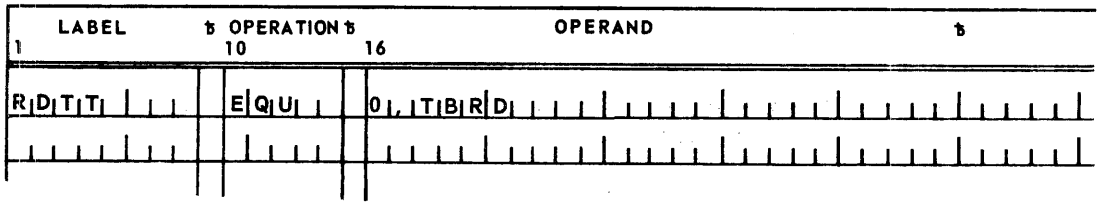
The last element of the input deck must be the Compressor element, named CP. In between the Card Load Routine and CP elements, the following elements may appear in any order:

- A card read routine. This could be either the online serial card reader routine, named XRDR, or the Card Controller routine, named XRDC. If XRDC is used, then an EQU card of the form:

LABEL	OPERATION	OPERAND
RDCN	EQU	n

should be included with the other EQU cards in the input deck, where 'n' is the number of the channel on which the Card Controller is located.

- An input translation table. To allow the user to punch source code in any card code that he wishes, the Compressor uses a user-supplied translation table to translate from card to internal code. The Compressor assumes that the result of the translation is EBCDIC. The input translation table supplied must be labeled RDTT If source code cards are punched in Hollerith, then the UNIVAC supplied Hollerith to EBCDIC translation table, which is named TBRD, can be used. In this case, an EQU card of the form:



should be included with the other EQU cards in the input deck.

- A card punch routine. This is either the online serial punch routine, named XPCH, or the row punch routine, named XPRW. If XPRW is used, then an EQU card of the form:

LABEL	OPERATION	OPERAND
CHAN	EQU	n

should be included with the other EQU cards in the input deck, where 'n' is the number of the channel on which the row punch is located.

- The Systems Programming internal routine for the printer, XPRT.

- If the PRTR EQU cards has been used to specify a value of TBPR, the UNIVAC supplied EBCDIC to 48-character-print-code translation table, which is named TBPR.

Blank card separators may be placed between the elements in the input deck, as the Linker ignores blank cards.

The operand field of the END control card used when the Compressor is linked must be: O,INIT

A13.5. Operating Instructions

1. Place the Compressor deck in the reader.
2. Home paper; clear the reader, punch, and printer.
3. Load the Compressor deck.
4. Stop X'1FFF'.
5. Place the macro definition source cards in the reader. Follow these cards by an end sentinel card and two blank cards.
6. Depress the START switch.
7. Stop X'1FFF'
   Punch output hopper contains the Macro Library.
8. To process another library, return to Step 5.

A13.6. Control Card

When the Compressor is being run from initial load, the macro definition source card deck may be preceded by a control card, which contains CTL in columns 10, 11, and 12, and a decimal number beginning in column 16. The number may take any value between 8191 and 32767. If a control card is present, the decimal number tells the Compressor the memory size in which it is to operate. If the control card is absent, or if the value specified lies outside the permissible range, the Compressor assumes that it is to operate in a 16K memory.

APPENDIX B. INPUT/OUTPUT CONTROL SYSTEM (IOCS)

B1. GENERAL DESCRIPTION

The Input/Output Control System (IOCS) provides the user with tested input/output routines to control the data which are the input or output of programs written in Assembler language. IOCS consists of two parts:

(a) the input/output routines themselves, which are macros and generated as a result of macro calls. The macros used to generate the input/output routines are called *declarative macro instructions.*

(b) the macro instructions used by the worker program to communicate with the input/output routines. These macro instructions are called *imperative macro instructions.*

B2. GENERAL USAGE

The user is provided with a complete set of routines for controlling all input/output operations required by the system. Since not every source program requires every routine or its variable functions, Univac Division of the Sperry Rand Corporation provides a Preassembly Macro Pass program which in effect is a generator capable of adapting each input/output routine to the requirements of the user.

The Preassembly Macro Pass first reads declarative macro instructions made by the user describing the input/output operations required by the application. Based on these instructions the Preassembly Macro Pass selects the required routines from the macro library, develops them for the specific application, and punches them into cards in the Assembler language format. They may then be assembled as part of the source program or assembled separately and linked with the user program at linker time. This function is provided by the UNIVAC 9200/9300 Card Linker program.

The user communicates with the IOCS routines through use of the macro calls (imperative macro instructions) in his problem program. Typical imperative macro instructions are OPEN, CLOSE, GET for an input file, and PUT for an output file. These imperative macro instructions are related to the input/output routine to which they refer by means of a file name. The same file name appears as a parameter in all of the imperative macro instructions referring to one file and also appears as the label of the declarative macro instruction generating the input/output routine for the file.

B3. DEFINITION STATEMENTS (DECLARATIVE MACROS)

The programmer must use definition statements to describe to the Preassembly Macro Pass the characteristics of the particular input/output file to be processed. These statements are then used by the macro pass to specialize the particular input/output routine to meet the requirements of the file and the program.

Each input/output device required by the program must be defined by means of these definitions. *A definition statement is herein defined as consisting of one Header Entry card and a number of Detail Entry cards.* In a definition statement, each header and detail entry card must have a character punched in column 72, except the final detail entry card which must not contain this continuation character in column 72.

B3.1. Header Entry Card

A header entry card is the first card of a definition statement and requires three items of information. The first is the symbolic name of the file assigned by the user and is entered in the label field of the card. The symbol may consist of as many as four characters and must adhere to the Assembler language rules for labels. The second item is written in the operation field and must be one of the following:

1. DTFCR – DEFINE THE FILE FOR THE CARD READER
2. DTFPR – DEFINE THE FILE FOR THE PRINTER
3. DTFRP – DEFINE THE FILE FOR THE READ/PUNCH
4. DTFCC – DEFINE THE FILE FOR THE CARD CONTROLLER
5. DTFRW – DEFINE THE FILE FOR THE ROW READ/PUNCH

The third item is a keyword parameter specification, described in Section B3.2, and is entered in the operand field.

For example, the header entry card for a reader routine with a file named "MSTR" would appear as follows:

LABEL	OPERATION	OPERAND
MSTR	DTFCR	keyword = specification

B3.2. Detail Entry Cards

The detail entry cards are used to define parameters such as mode of processing, buffer area name, and print bar.

Each detail entry card is composed of a keyword immediately followed by an equal (=) sign which is in turn followed by one specification. A comma must immediately follow the specification for each detail entry card in the definition statement except for the final detail entry card. A given detail entry must be used only once in each definition statement. Entries which do not apply to a particular application should be omitted. The summary of detail entry cards listed in Appendix B4, gives the optional as well as the required detail entry cards for a given peripheral device. The format for a detail entry card, with the continuation character in column 72, is as follows:

LABEL	OPERATION	OPERAND	72
symbol	DTFxx	keyword=specification,	x

B3.2.1. Block Size Entry (BKSZ)

This entry must be provided for all printer files. The keyword is BKSZ. Any number from 1 through 132 may be specified, but the number should be no larger than the number of print positions available. The user-defined work area where print images are made available to IOCS must contain the same number of bytes as is specified for BKSZ. IOCS left justifies the print images on the printed line and supplies spaces for printing in the remaining unspecified print positions. The keyword and specification for 132 print positions have the following form in the operand field:

BKSZ=132

B3.2.2. Channel Entry (CHNL)

This entry is used to define the general purpose channel to which the UNIVAC 1001 Card Controller or the Row Read/Punch is connected. The keyword is CHNL; the allowable specification is one of the general purpose channels 5 through 12. The keyword and specification for a channel entry for general purpose channel 5 have the following form:

CHNL=5

B3.2.3. Control Entry (CNTL)

This entry must be provided for all files to which a CNTRL macro instruction is directed in the main program.

The keyword is CNTL. The specification is YES.

CNTL=YES

CNTL is a detail entry card within a definition statement. CNTRL is an imperative macro and its use is described in a later section.

B3.2.4. End-of-File Address Entry (EOFA)

This entry is used to specify the symbolic name of the end-of-file routine provided by the user. The keyword is EOFA and the specification is the symbolic name of the user end-of-file routine. The format for an end-of-file routine labeled END is as follows:

EOFA=END

If the image to be delivered is an end-of-file card, IOCS jumps unconditionally to the user end-of-file routine when a GET macro instruction is issued for an input file.

An end-of-file card contains a slash (/[0-1 punch]) in column one and an asterisk in column two. (In actuality, the card system IOCS routines recognize an end-of-file card by means of the slash in column one alone.) An end-of-file card must be followed by other cards in the input hopper to avoid a hopper empty indication before the end-of-file card is sensed. The following cards may be special if the user has some purpose for them (such as an overlay to be loaded); otherwise, their content is not significant and any cards the user wants may be used (such as blank cards or more end-of-file cards).

For the online card reader, when control is transferred to the user end-of-file address, the end-of-file card image is in the work area and the image of the card immediately following the end-of-file card is in the input area.

The EOFA parameter is not applicable to the UNIVAC 1001 Card Controller. The user must test for his own end-of-file.

B3.2.5. The Function Entry – UNIVAC 1001 Card Controller (FUNC)

This entry specifies the symbolic name of a one-byte user-defined area where the hexadecimal value for the required function is stored before each GET or PUT macro instruction.

The keyword is FUNC. The specification is the label of the one byte user area, and for a function area labeled CCXF, has the following form:

FUNC=CCXF

B3.2.6. Allowable Functions for the UNIVAC 1001 Card Controller

The following table illustrates the allowable hexadecimal values which may be stored into the user-defined one-byte area before each GET or PUT macro instruction is issued. Once set, the area may remain the same or be altered as desired.

HEXADECIMAL VALUE	FUNCTION SPECIFIED
08	Transfer and Read Primary
09	Transfer and Read Secondary
00	Transfer Primary
01	Transfer Secondary
02	Transfer Primary and Secondary
0A	Transfer and Read Primary and Secondary
24	Send Data to 1001 (1001 code only)
14	Receive Data from 1001 (1001 code only)

The GET macro is used with all functions but "Send Data to 1001". With this function a PUT macro is used.

B3.2.6.1. Transfer-and-Read Functions

The previous image read into the UNIVAC 1001 Card Controller is transferred into the UNIVAC 9200/9300 memory and another image is read into the UNIVAC 1001. The function for the first GET executed after opening a Card Controller file should be a transfer-and-read function which, in contrast to the general case, causes the first card in the feed specified to be read and transferred, and the second card to be read.

B3.2.6.2. Send-and-Receive Data Functions

These functions are not available on the standard board. However they are provided for by IOCS in the event the user wishes to wire his own board for a particular application.

No translation is provided for these functions and they must be performed in UNIVAC 1001 mode only.

The user work area must contain one byte more than is required for the data to be sent or received. The extra byte must be the first byte of the area and must contain the number of characters to be transmitted. This first byte must not be in UNIVAC 1001 mode, but must contain a binary number.

Typically, the data sent to the UNIVAC 1001 contains some function character the modified board is to interpret, as well as data to be used in the execution of the function.

For example, assume the board has been modified to interpret the code of a hexadecimal value of 77 as a search primary for a name. The following steps implement this function:

(1) Set function entry area to a send-data function.

(2) Store hexadecimal 77 into the second byte of the work area.

(3) Store name (assume 6 characters) in work area bytes 3 through 8.

(4) Store a binary 7 (6+1 function) into first byte of the work area (the number of characters to be transmitted).

(5) Issue a PUT macro instruction.

When the UNIVAC 9200/9300 program receives the data the UNIVAC 1001 has developed as a result of performing this search, the following steps are taken:

(1) Set the function entry area to a receive-data function.

(2) Store the number of characters to be received in the first byte of the work area.

(3) Issue a GET macro instruction.

The data will be received in byte 2 and the following bytes of the work area. Typically, the data received from the UNIVAC 1001 contains some status character (find/no-find, for example) and the data requested by the preceding send-data function.

The nature of any function or status characters embedded in data to be sent or received and the location of these characters in the data message is a user responsibility. The IOCS system makes no attempt to control the information content of data sent or received.

B3.2.7. Input Area Entry (IOA1)

This entry specifies the name of the input buffer area. In the UNIVAC 9200/9300 Card System, it is used only for the reader file. The keyword is IOA1. The specification is the symbolic name of the input buffer area assigned to the device. This symbolic name must be the symbol used by the programmer in the DS statement defining the area in his main program.

IOA1=CARD

In this instance, CARD is the symbolic name of the input buffer area. The symbolic name assigned by this entry is never referenced directly by the programmer. Images are delivered by the input/output routines into a specified work area.

B3.2.8. Input Area Entry (INAR)

This entry is used to specify the symbolic name of the user-defined input buffer area when the read feature of the read/punch unit is required. The keyword is INAR. The specification is the symbolic name of the area assigned to the read/punch unit as defined by the programmer. The operand for a read/punch buffer area labeled INPC has the following form.

INAR=INPC

B3.2.9. Input Translate Table Entry (ITBL)

This entry specifies the symbolic name of a translate table located in the main gram by which all records of a given input file are to be translated.

The keyword is ITBL and the specification is the symbolic name assigned by the programmer to the table. The operand for a translate table labeled CODE has the following form:

ITBL=CODE

B3.2.10. Mode Detail Entry (MODE)

This entry is used to specify the mode of the input/output file and is required as part of the definition statement for all devices but the printer. The keyword of the entry is MODE. The allowable specifications are:

OPERAND FORM	REMARKS
MODE=BINARY	For cards read and/or punched in column binary mode.
MODE=CC	For cards read and/or punched in compressed code (80-byte I/O area required).
MODE=1001	For cards read in UNIVAC 1001 mode without translation (Card Controller only) (80-byte I/O area required).
MODE=TRANS	For cards to be read and/or punched translated by the table specified by the ITBL or OTBL entry.
MODE=TRANSTC	For Card Controller only, if translation of UNIVAC 1001 code is required through the translation table specified by the ITBL entry.

There are two translation modes which may be defined with the UNIVAC 1001 Card Controller.

- TRANS, implies all cards read into the UNIVAC 9200/9300 from the UNIVAC 1001 are translated *from compressed code* by the translate table specified by the ITBL detail entry card.

- TRANSTC, implies all cards read into the UNIVAC 9200/9300 from the UNIVAC 1001 are translated *from 1001 code* by the translate table specified by the ITBL detail entry card. This mode is used when combined reading (both primary and secondary in one function) is required, since basic UNIVAC 1001 memory capability is exceeded if two images are read in other than 1001 code.

For the online serial card reader operating in translated mode, card images are read into the input area in compressed code, moved to the work area, and translated there. For example, when control is transferred to the user end-of-file address, the image of the end-of-file card is in the work area in translated mode, and the image of the card immediately following the end-of-file card is in the input area in compressed code.

For the Card Controller operating in translated mode, card images are read into the work area in compressed code and are translated in the work area.

B3.2.11. Output Area Entry (OUAR)

The entry specifies the symbolic name of the output buffer area as defined in the main program when the punch function of the punch, read/punch unit is required.

The keyword is OUAR. The specification is the symbolic name assigned by the programmer in the DS statement defining the area. The operand for an output area labeled OUPC has the following form:

OUAR = OUPC

There is no need to define an output buffer area for the printer, since IOCS uses the print buffer area in restricted memory.

B3.2.12. Output Translate Table (OTBL)

The entry specifies the symbolic name of the translation table located in the problem program through which all output images are to be translated.

The keyword is OTBL. The specification is the symbolic name assigned to the table. The operand for a translation table labeled CRDC has the following form:

OTBL = CRDC

B3.2.13. Overlap Entry (ORLP)

This entry specifies that the read/punch unit file is to be processed in an overlap mode and applies only to the read/punch unit when used as both a reader and a punch. The entry is *omitted* when information is to be punched in a card which has been read previously.

The keyword of this entry is ORLP and the specification is YES. The operand has the following form:

ORLP = YES

B3.2.14. Print Bar Entry (FONT)

The entry specifies the print bar the program expects to find in the user configuration. The keyword is FONT and the allowable specifications are 48 or 63. The operand for a 63-character print bar has the following form:

FONT = 63

When using a UNIVAC 9300 System with a 16-character print bar, FONT=48 should be specified. *Numeric printing must be specified in the CNTRL macro instruction (B6.7.4); an alphanumeric print may damage the print bar.*

B3.2.15. Printer Advance Entry (PRAD)

This entry is used in conjunction with printer files and enables the programmer to specify a standard advance of one or two lines.

The keyword is PRAD. The allowable specifications are 1 or 2. The operand for double spacing has the following form:

PRAD = 2

B3.2.16. Punch Error Entry (PUNR)

This entry specifies that automatic error recovery, where possible, is to be provided in the online serial or row punch routine and applies only to these devices. If it is not specified, all punch errors bring the computer to a stop.

The keyword is PUNR. The allowable specification is YES. The operand has the following form:

PUNR = YES

B3.2.17. Printer Overflow Entry (PROV)

This entry must be provided if the user wants any special action as a result of form overflow on the printer. If the printer overflow entry is not provided, printer spacing proceeds as directed by the printer advance detail entry and/or the CNTRL macro specifying skipping or spacing.

The keyword of the entry is PROV. The specification may be either YES or a label. The operand has the following form:

PROV = YES

or

PROV = label

If the specification is YES, an automatic skip to channel 7 (home paper) in the paper tape loop is provided in response to form overflow.

If the specification is a label other than YES, control is transferred unconditionally to the specified label in response to form overflow. The label specified should be the symbolic name assigned to the user overflow routine provided to perform the desired form overflow action.

The user indicates the point at which form overflow is to occur by a channel 1 punch in the paper tape loop. The form overflow punch (channel 1 punch) is recognized when spacing paper, either in response to a CNTRL macro specifying spacing before printing or in response to a PUT macro after printing a line. (The form overflow punch is not recognized during a printer skip operation.)

Response to recognition of a form overflow punch may be illustrated by the following sequence of operations:

```
(1)          PUT     FILA
      (or    CNTRL   FILA,SP,m,n   m≠0)
             .
             . Process
             .

(2)          PUT     FILA
      (or    CNTRL   FILA
             .
             . Process
             .

(3)          PUT     FILA
      (or    CNTRL   FILA
        RET  .
             . Process
             .
```

If the form overflow punch is recognized during the spacing associated with (1), then after (3) is executed, the form overflow action specified is taken. If the action is to transfer control to a user subroutine, then control goes to that subroutine rather than to the label RET. The address of the label RET is in general register 14 when control is transferred to the form overflow subroutine.

B3.2.18. Type of File Entry (TYPF)

This entry indicates whether the file is an input, output, or a combined file. It is applicable only to the UNIVAC 9200/9300 Read/Punch Unit.

The keyword of the entry is TYPF. The allowable specifications are given below.

OPERAND FIELD	COMMENTS
TYPF = INPUT	Reading only
= OUTPUT	Punching only
= COMBND	Reading and punching

B4. SUMMARY OF DETAIL ENTRY CARDS

OPERANDS FIELD		APPLIES TO					
KEYWORD	ALLOWABLE SPECIFICATION	READER	PRINTER	SERIAL READ PUNCH	ROW READ/ PUNCH	CARD CON- TROLLER	REMARKS
BKSZ	1–132		X				Required for online printer
CHNL	5 thru 12				X	X	Required for UNIVAC 1001
CNTL	YES		X	X	X	X	Required if CNTRL macro is used
EOFA	Symbolic name of user end-of-file routine	X		X	X		Applies to input files only
FUNC	Symbolic name of user defined 1-byte area where function is stored					X	Required by Card Controller
IOA1	Symbolic name of user defined input buffer area	X					If binary image requested, 160-byte area required
INAR	Symbolic name of user defined input buffer area			X	X		Required if reading in the read/punch file
ITBL	Symbolic name of user defined input translate table	X		X	X	X	Required if translation of input file desired
MODE	See Appendix B,3.2.10	X		X	X	X	
OUAR	Symbolic name of user defined output buffer area			X	X		Required for punch files and read/punch files
OTBL	Symbolic name of user defined output translate table		X	X	X		Required if translation of output file desired
ORLP	YES			X	X		
FONT	48 or 63		X				Specifies 48- or 63-character print font
PROV	YES or symbolic name of user form overflow routine		X				Required if form overflow action is to be taken
PRAD	1 or 2		X				Specifies standard print advance
PUNR	YES			X	X		Automatic error recovery desired
TYPF	INPUT			X	X		Reading only
	OUTPUT			X	X		Punching only
	COMBND			X	X		Reading & punching

B5. DEFINITION STATEMENT EXAMPLES

The following are examples of definition statements.

B5.1. Online Serial Punch File Example Definition

LABEL (1)	OPERATION (10)	OPERAND (16)	COMMENTS	72	80
MSTR	DTFRP	CNTL=YES,		X	
		MODE=TRANS,		X	
		OTBL=MTCC,		X	
		OUAR=PUNC,		X	
		TYPF=OUTPUT			

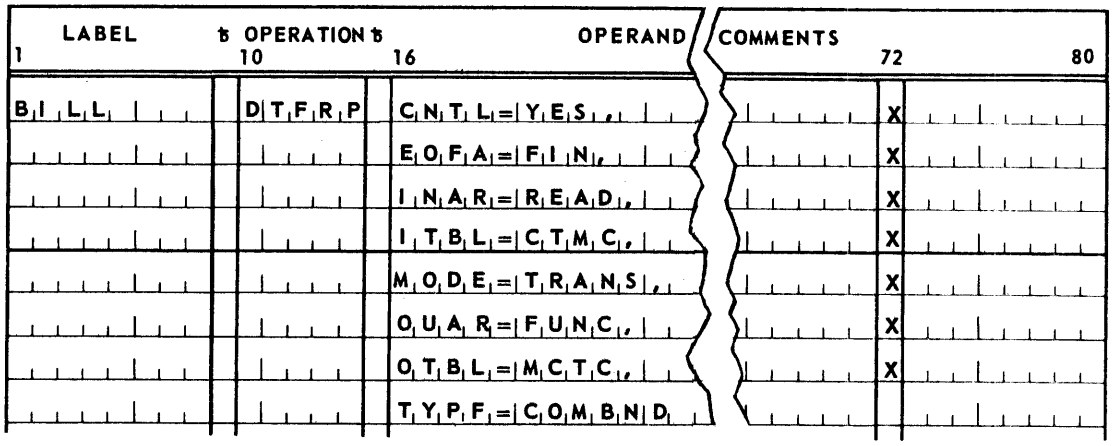
B5.2. Reader File Example Definition

LABEL (1)	OPERATION (10)	OPERAND (16)	COMMENTS	72	80
INPU	DTFCR	EOFA=END,		X	
		IOA1=CARD,		X	
		ITBL=CTMC,		X	
		MODE=TRANS			

B5.3. Printer File Example Definition

LABEL (1)	OPERATION (10)	OPERAND (16)	COMMENTS	72	80
LIST	DTFPR	CNTL=YES,		X	
		FONT=48,		X	
		PRAD=2,		X	
		PROV=YES		X	
		OTBL=PB48,		X	
		BKSZ=132			

B5.4. Online Serial Read and Punch File Example



B5.5. Card Controller File Example

B6. IOCS MACRO INSTRUCTIONS (IMPERATIVE MACROS)

This section describes the format, function, and use of the IOCS macro instructions used to communicate with the input/output routines and to control their operations. These symbolic instructions are used in the problem program to provide the communication necessary with the IOCS routines previously defined by means of the definition statements to the Preassembly Macro Pass. The handling of records into and out of I/O areas is performed by IOCS exclusively. Each file is processed in the manner dictated by the definition statement.

Source programs using IOCS may not contain any Assembler I/O instructions.

The format of the macro instruction follows the rules of the Assembler coding format. The macro verb is the operation, and the operand field may contain up to four parameters as required by the particular macro. All macros may have a label. The imperative macro instructions are not handled by the Preassembly Macro Pass but are processed by the Assembler itself.

B6.1. GET Macro Instruction

The GET macro makes the next record available in the user-defined work area or transfers control to the end-of-file address entry upon recognizing an end-of-file card in an input file.

The GET macro has the following form

LABEL	OPERATION	OPERAND
	GET	filename,workarea

where "filename" is the symbolic name defined in the label field of a DTF(XX) header entry card.

"workarea" is the symbolic name of the user-defined storage where the record is made available for output.

B6.2. PUT Macro Instruction

The PUT macro transfers a record from the work area for printing, punching, or sending to the UNIVAC 1001 and immediately frees the work area for problem program use.

The PUT macro has the following form

OPERATION	OPERAND
PUT	filename,workarea

where "filename" is the symbolic name defined in the label field of a DTF(XX) header entry card.

"workarea" is the symbolic name of the user-defined storage where the record is made available for output.

B6.3. Work Area Considerations

The imperative macro instructions, GET and PUT, require as a second parameter the symbolic name of a work area for transferring records from and to input/output buffer areas. Input/output areas (those assigned by IOA1, INAR, and OUAR detail entry cards) may not be used as work areas as they are used by IOCS to maintain standby reserve areas.

The programmer must therefore provide, through the use of DS statements, work areas where records are processed. These work areas may be common to more than one file as efficiency demands, but must be as large as the largest record to be processed therein.

B6.4. Programming Considerations – Read/Punch Combined File

When the Overlap detail entry card is used for a read/punch combined file, the following rule applies:

A PUT macro instruction causes punching into the card which follows the one made available by the last GET macro instruction, because the card made available by the last GET macro is already past the punch station when the PUT macro is given.

When the Overlap detail entry card is omitted for a read/punch combined file, a PUT macro instruction causes punching into the card made available by the last GET macro instruction. It should be noted that in this nonoverlap mode, the read/punch unit is cycled on PUT's only; therefore, successive GET's must be separated by at least one PUT.

B6.5. OPEN Macro Instruction

This macro instruction initializes the file and must be issued before any other macro instruction pertaining to the same file.

The OPEN macro has the following form:

OPERATION	OPERAND
OPEN	filename

where "filename" is the symbolic name defined by the user in the label field of the DTF(XX) header entry card.

After a file has been OPENed, it should be CLOSEd before reOPENing the same file. In such a case, the input/output routine is set back to an initial state. As as example:

1. For an input file, the card image delivered in response to the first GET executed after a second OPEN macro is the image immediately in front of the read station at the time the second OPEN macro is given.

2. For an output file, the first item transmitted after the second OPEN macro is the item delivered by the first PUT executed after the reOPEN.

B6.6. CLOSE Macro Instruction

This macro instruction insures the proper closing of all files. The CLOSE macro has the following form:

OPERATION	OPERAND
CLOSE	filename

where "filename" is the symbolic name defined in the label field of the DTF(XX) header entry card.

B6.7. CNTRL Macro Instruction

The CNTRL macro is used by the programmer for printer spacing, printer skipping, stacker selection, numeric printing, and specifying the number of columns of card punching.

B6.7.1. Printer Spacing

The CNTRL macro for printer spacing has the following form:

OPERATION	OPERAND
CNTRL	filename,SP,m,n

where "filename" is the symbolic name of the file defined in the label field of the DTFPR header entry card.

SP specifies spacing.

m is the number of lines to space the form before printing (m=0, 1 or 2).

n is the number of lines to space the form after printing (n=0,1 or 2).

The programmer may omit "m" or "n". If no CNTRL macro instruction specifying delayed spacing is given before the next PUT macro for the printer file, the printer advances the standard amount as specified in the PRAD detail entry card of the definition statement.

If more than one CNTRL macro specifying paper movement after printing is given between PUT macros to the printer file, only the last CNTRL macro is effective.

B6.7.2. Printer Skipping

The CNTRL macro for printer skipping has the following form:

OPERATION	OPERAND
CNTRL	filename,SK,m,n

where "filename" is the symbolic name of the file defined in the label field of the DTFPR header entry card.

SK specifies skipping

m is the number of the tape channel the carriage is skipped to before printing (m=1,2....7).

n is the number of the tape channel the carriage is skipped to after printing (=1,2....7).

The programmer may omit "m" or "n". Between PUT macros, only the last CNTRL macro specifying paper movement after printing is effective.

Due to timing conditions, throughput is maintained at the best possible level if delayed spacing and skipping are used where possible.

B6.7.3. Stacker Select

The CNTRL macro for selecting other than the normal stacker on the serial punch, read/punch, or for selecting any stacker on the card controller has the following form:

OPERATION	OPERAND
CNTRL	filename,SS,n

where "filename" is the symbolic name of the file defined in the label field of the header entry card.

SS specifies stacker select.

n is the stacker number where the card is to be selected on the card controller only. Allowable values are specified in the following table.

	FEED							
	PRIMARY				SECONDARY			
Stacker	1	2	3	C	1	2	3	C
Specification (n)	1	2	3	4	5	6	7	8

NOTE: If this CNTRL macro is *not given* for the UNIVAC 1001, primary feeds are selected to P1 and secondary feeds to S1.

The CNTRL macro for Card Controller stacker selection operates under the following rules. The card made available by a transfer-and-read from a particular feed is selected on the next transfer-and-read from that feed. The card made available by a transfer only from a particular feed is selected on the second following transfer-and-read from that feed. If the user issues a CNTRL macro after receiving a particular card image, the CNTRL macro governs the stacker selection for that particular card, regardless of the sequence of operations following that CNTRL macro. If no stacker is selected from a card in the manner described here, the card will be put in the normal stacker, which is P1 for the primary feed and S1 for the secondary feed.

The CNTRL macro instruction for selecting a stacker on the read/punch should be executed before the PUT for the card to be selected, or in the case of reading only, after the GET for the card to be selected. If the CNTRL macro instruction is to be used for selecting a stacker when both reading and punching, and if stacker selection is done on the basis of information in the card read, the read/punch routine must be operated in the no overlap mode.

B6.7.4. Numeric Printing

The CNTRL macro instruction for numeric printing enables the programmer to maintain maximum printing speeds. Once set, it remains set until and unless another numeric print CNTRL macro is given specifying that alphanumeric printing is requested.

The CNTRL macro instruction for numeric printing has the following form:

OPERATION	OPERAND
CNTRL	filename,NP,m

where "filename" is the symbolic name of the file as defined in the label field of the DTFPR header entry card.

NP specifies a change in the mode of printing.

m is mode of printing requested.

m = 0, alphanumeric printing required.

m = 1, numeric printing required.

NOTE: Alphanumeric printing is assumed by IOCS if no CNTRL macro is given.

The CNTRL macro instruction should be used only in conjunction with a 48- or 16-character print bar.

B6.7.5. Specifying Columns to be Punched

The CNTRL macro instruction enables the programmer to vary or alter the number of columns punched in the punch file. This function enables the punch to run at maximum speed for the particular application. Once set by the macro, the number of columns punched remains the same unless or until another such macro is given. If no CNTRL macro is given, IOCS assumes a full card is required.

The CNTRL macro for punching has the following form

OPERATION	OPERAND
CNTRL	filename, NC,n

where "filename" is the symbolic name of the file defined in the label field of the DTFRP header entry card.

NC identifies a number-of-columns specification.

n is the number of columns to be punched (an even number 2,4,6...80).

B6.8. Summary of UNIVAC 9200/9300 Card System IOCS Imperative Macros

			DEVICE ADDRESSED				
LABEL	OPERATION	OPERANDS	READER	PRINTER	SERIAL READ/ PUNCH	ROW READ/ PUNCH	CARD CONTROLLER
OPTIONAL	OPEN	filename	X	X	X	X	X
	GET	filename, workarea	X		X	X	X
	PUT	filename, workarea		X	X	X	X
	CLOSE	filename	X	X	X	X	X
	CNTRL	filename, SP, m, n		X			
	CNTRL	filename, SK, m, n		X			
	CNTRL	filename, NP, m		X			
	CNTRL	filename, NC, n			X	X	
	CNTRL	filename, SS, n			X	X	X

B 7. PROGRAMMING CONVENTIONS – PROGRAM REGISTERS

A user routine may be required in the main source program that is accessed by IOCS when certain checking features are required (for example, printer overflow). IOCS automatically stores the program re-entry address in register 14 when the branch to the user routine occurs. The user routine is therefore required to provide the necessary return linkage to the main source program. If the user routine uses register 14, it must, therefore, preserve and restore register 14 before terminating. This must also be done if any macro instruction is executed by the user routine, since all macros use program registers 14 and 15. If register 14 is not preserved, the re-entry address is lost. Register 15 may also be used by the user routine and it need not be preserved. However, its contents are altered by the execution of any macro instruction.

B 8. GENERAL PROCEDURE SUMMARY FOR USING IOCS

The programmer defines his input/output control routines and their associated files through the use of definition statements presented to the Preassembly Macro Pass program. The generated I/O routines are then either assembled as part of the main source program or assembled separately and linked with the main program at load time. If the I/O routines are assembled independently, the user must supply appropriate USING directives; no USING directives are generated by IOCS. During the execution of the main program, input/output functions are accomplished through the imperative macro calls.

B 9. STORAGE REQUIREMENTS

The IOCS routines require the use of EXEC I. The following two charts show the storage requirements for the IOCS routines.

DEVICE	MODE OF OPERATION	STORAGE REQUIREMENT
EXEC I	–	180
Reader	–	150
Printer	–	330
Serial Read/Punch	–	–
Read	–	190
Punch	–	260
Read/Punch	No overlap	330
Read/Punch	Overlap	340
Card Controller	–	720
Row Read/Punch	–	–
Read	–	360
Punch	–	400
Read/Punch	No overlap	530
Read/Punch	Overlap	560

MACRO INSTRUCTION	STORAGE REQUIREMENT
OPEN	4
GET	6
PUT	6
CNTRL	8
CLOSE	4

B10. APPROXIMATE TIMES FOR IOCS ROUTINE EXECUTION

The following chart shows the approximate execution times (in milliseconds) required for the various IOCS macro instructions on the UNIVAC 9200. The UNIVAC 9300 execution times are one half the UNIVAC 9200 execution times.

DEVICE	MODE OF OPERATION	TIME REQUIRED (IN MILLISECONDS)	
		GET	PUT
Reader*	–	5	–
Printer	–	–	4
Serial Read/Punch	–	–	–
Read*	–	5	–
Punch*	–	–	5
Read/Punch*	No overlap	5	5
Read/Punch*	Overlap	5	5
Card Controller*	--	5	–
Row Read/Punch	--	–	–
Read*	–	7	–
Punch*	--	–	7
Read/Punch*	No overlap	5	7
Read/Punch*	Overlap	7	8

**Translation time included.*

B11. CARD READER DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
filename	DTFCR	EOFA = location of the user end-of-file routine,	X
		IOA1 = location of the input buffer area,	X
		ITBL = location of the input translate table,	X
		MODE = BINARY or MODE = CC or MODE = TRANS	

ITBL is required if MODE = TRANS.

EOFA, IOA1 and MODE must always be present.

The input buffer area and word area are each 80 bytes long, if MODE = CC or TRANS, and 160 bytes long if MODE = BINARY.

At the time control is unconditionally transferred to the end-of-file routine, both the end-of-file card and the following card are in the output stacker.

B11.1. Preparing the Card Reader

1. Empty the read wait station.
2. Place the reader online.
3. Place the input deck in the hopper.
4. Depress the reader CLEAR switch.
5. Feed one card into the wait station.

B11.2. Error Indications

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
RDR ABN (Program Loop)	Unit placed offline while in the process of reading a card.	Recoverable	Put online to continue.
6100 RDR ABN	Offline	Recoverable	Put online. Press START to continue.
6140 RDR ABN	Not ready, hopper empty, stacker full.	Recoverable	Correct condition. Clear reader abnormal. Press START to continue.
6140 RDR ABN	Misfeed – card has not properly entered the wait station.	Recoverable	One card, the card in wait station, must be placed at bottom of input deck. Feed one card. Clear reader abnormal. Press START to continue.
6180	Card jam or Photo-cell check error – card did not progress thru read station properly.	Recoverable	Two cards, the last card to go to stacker and card in wait station, must be placed at bottom of input deck. Feed one card. Clear reader abnormal. Press START to continue.
6104	Interrupt pending	Not an error	No action is required by operator. This display can occur only in conjunction with some other display.

NOTE:

- The first two characters (61) displayed identify the stop as a Reader stop.
- The next two characters represent the status byte.
- The status byte may reflect a combination of bit settings; therefore, the operator may have to take corrective action for more than one error condition before attempting to continue.

B12. PRINTER DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
filename	DTFPR	BKSZ = n,	X
		CNTL = YES,	X
		OTBL = location of the output translate table,	X
		FONT = 48,	X
		or	
		FONT = 63,	X
		PROV = YES,	X
		or	
		PROV = location of the form overflow routine,	X
		PRAD = 1	
		or	
		PRAD = 2	

where: n = the number of bytes to be moved from the work area to the printer buffer area for printing (1–132). If BKSZ is omitted, this is assumed to be 132 bytes.

CNTL is required only if the CNTRL macro instruction for spacing or skipping is used by the problem program.

OTBL is required only if MODE = TRANS.

PROV is required only if form overflow is not to be ignored.

If FONT is omitted, the 63-character print bar is assumed.

If PRAD is omitted, normal spacing is set to 1.

B12.1. Preparing the Printer

1. Install the proper paper loop for the program to be run.

IOCS assumes 001 = form overflow position.

111 = home paper position.

If the paper loop is absent, the paper will space one position and the CNTRL SK option in the IOCS routine should not be accessed.

2. Install the proper paper for the program to be run, and adjust it to the top of the page (matching loop).

3. Place the printer online.

4. Depress the printer CLEAR switch.

B12.2. Error Indications

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
Program Loop	Offline	Recoverable	Put online to continue.
6301	Paper low	Recoverable	Correct condition. Press START to continue.
6302	Form overflow 001 sensed at paper loop station, while single or double spacing.	Not an error	No action required by operator. This display can occur only in conjunction with some other display.
6304	Interrupt pending	Not an error	No action required by operator. This display can occur only in conjunction with some other display.
6308 PRNT ABN	Wrong bar setting in XIOF.	Recoverable	Insert correct bar. Set bar switch appropriately. Press START.
6320	Memory overload	Recoverable	Press START to continue.
6340 PRNT ABN	Paper runaway	Recoverable when paper position can be reestablished	Check paper loop. Replace with proper loop. Clear printer abnormal. Press START to continue. If form position has been lost, go to RESTART.
6380	Abnormal or not ready	Recoverable	Correct abnormal condition. Clear printer abnormal. Press START to continue.

NOTE:

- The first two characters (63) displayed identify the stop as a Printer stop.
- The next two characters represent the status byte.
- The status byte may reflect a combination of bit settings; therefore, the operator may have to take corrective action for more than one error condition before attempting to continue.

B12.3. Paper Low

On encountering a paper low condition, the Printer IOCS routine continues to permit printing until paper is homed. At this point, a paper low display is made. To assure proper paper positioning, a page from the new stock should be placed directly over the page in the printer on which the computer stopped.

B13. SERIAL PUNCH DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
Filename	DTFRP	CNTL=YES,	X
		MODE=BINARY,	X
		or	
		MODE=CC	X
		or	
		MODE=TRANS,	X
		OUAR=location of the output buffer area,	X
		OTBL=location of the output translate table,	X
		PUNR=YES,	X
		TYPF=OUTPUT	

CNTL is required only if the CNTRL macro instruction for stacker selection or number of columns to be punched is used by the problem program.

OTBL is required if MODE=TRANS.

PUNR is required only if automatic error recovery (try five times) is desired for punch check errors.

If TYPF is not specified, output will be assumed.

MODE and OUAR must always be present.

B14. SERIAL READ DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
filename	DTFRP	CNTL=YES	X
		EOFA=location of the user end-of-file routine,	X
		INAR=location of the input buffer area,	X
		ITBL=location of the input translate table,	X
		MODE=BINARY	X
		or	
		MODE=CC,	X
		or	
		MODE=TRANS,	X
		TYPF=INPUT	

CNTL is required only if the CNTRL macro instruction for stacker selection is used by the problem program.

ITBL is required if MODE=TRANS.

EOFA, INAR, MODE and TYPF must always be present.

B15. SERIAL READ/PUNCH DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
filename	DTFRP	CNTL=YES,	X
		EOFA=location of the user end-of-file routine,	X
		INAR=location of the input buffer area,	X
		ITBL=location of the input translate table,	X
		MODE=BINARY,	X
		or	
		MODE=CC,	X
		or	
		MODE=TRANS,	X
		OUAR=location of the output buffer area,	X
		OTBL=location of the output translate table,	X
		ORLP=YES,	X
		TYPF=COMBND	

CNTL is required only if the CNTRL macro instruction for stacker selection or number of columns to be punched is used by the problem program.

ITBL and OTBL is required if MODE=TRANS.

ORPL is required when *not* punching into the same card made available by the last GET macro instruction.

EOFA, INAR, MODE, OUAR and TYPF must always be present.

B15.1. Buffer and Work Area Size

The input buffer area and work area are each 80 bytes long if MODE=CC or TRANS, and are 160 bytes long if MODE=BINARY.

The output buffer area and work area lengths are each equal to the number of columns to be punched, if MODE=CC or TRANS, and are each equal to two times the number of columns to be punched, if MODE=BINARY.

B15.2. End-of-File

If the Serial Read/Punch is being used for reading only, or if combined reading and punching is being done with overlap at the time control is transferred to the end-of-file routine, the end-of-file card is in the output stacker. If combined reading and punching is being done in a no overlap mode, the end-of-file card is in the punch wait station.

B15.3. Preparing the Serial Read/Punch

1. Empty the wait stations.
2. Place the Serial Read/Punch online.
3. Place the deck in the hopper.
4. Depress the Serial Read/Punch CLEAR switch.
5. If Reading alone or Reading and Punching — Feed 1 card into wait station.
6. If Punching alone — Feed 2 cards into wait station.

B15.4. Error Indications

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
PNCH ABN (Program Loop)	Unit placed offline while in the process of punching or reading a card.	Recoverable	Put online to continue.
6200 PNCH ABN	Offline	Recoverable	Put online. Clear punch abnormal. Press START to continue.
6202 PNCH ABN	Hopper empty or stacker full.	Recoverable	Correct condition. Clear punch abnormal. Press START to continue.
6204	Interrupt pending	Not an error	No action required by operator. This display can occur only in conjunction with some other display.
6208 PNCH ABN	Photocell check error indicates improper recognition of card presently at punch wait station.	Recoverable only if using read alone	Two cards – card at punch wait and card at read wait station must be placed at bottom of input deck. Feed one card. Clear punch abnormal. Press START to continue.
6220 PNCH ABN	Punch check error	Recoverable only if using punch alone	Error card in error stacker. Clear abnormal condition. Press START to continue.
6280 PNCH ABN	Interlock – casework is open.	Recoverable	Correct condition. Clear punch abnormal. Press START to continue.
6280 PNCH ABN	Misfeed	Recoverable	Clear punch abnormal. Fill unit stations. Press START to continue.
6280 PNCH ABN	Stacker jam, punch entry, exit check.	Recoverable only for punch alone	Error card is last card to go to error stacker. Clear abnormal condition. Press START to continue.

NOTE:

- The first two characters (62) displayed identify the stop as a Serial Read/Punch stop.
- The next two characters represent the status byte.

B16. UNIVAC 1001 CARD CONTROLLER DEFINITION STATEMENTS

LABEL	OPERATION	OPERAND	72
filename	DTFCC	CNTL=YES,	X
		FUNC=location of the 1001 function,	X
		MODE=BINARY, or	X
		MODE=CC, or	X
		MODE=1001, or	X
		MODE=TRANS, or	X
		MODE=TRANSTC,	X
		ITBL=location of the input translate table,	X
		CHNL=5 thru 12	

CNTL is required only if the CNTRL macro instruction for stacker selection is used by the problem program.

ITBL is required only if MODE=TRANS or TRANSTC.

MODE=1001, must be used with UNIVAC 1001 functions '14' and '24'.

MODE=BINARY, CC, or TRANS must *not* be used with UNIVAC 1001 functions '02' or '0A'.

FUNC, MODE, and CHNL must always be present with header entry card DTFCC.

B16.1. Work Area Size

The work area size is a function of the Card Controller function and the mode in which the file is to be read. The work area size is shown in bytes on the following chart:

FUNCTION	MODE		
	CC or TRANS	1001 or TRANSTC	BINARY
Transfer Primary	80	80	161
Transfer and Read Primary	80	80	161
Transfer Secondary	80	80	161
Transfer and Read Secondary	80	80	161
Transfer Both	--	160	---
Transfer and Read Both	--	160	---

B16.2. Preparing the Card Controller

(1) Ensure that the unit is online and that the power is on. (The online switch for the Card Controller is housed in the lower left area at the back of the Printer Processor Cabinet.)

(2) Ensure that the ON-LINE Standard Interface Panel (plugboard program) is mounted.

(3) Set the MODE switch to CONT; set the alternate switches (ALT 1 to ALT 4) off.

(4) Empty the ready and wait stations.

Press: CLEAR
UNLOAD PRI (3 times)
UNLOAD SEC (3 times)

(5) Place the card decks in the input hopper face down, nine edge first.

(6) Rotate the display mask switch until Step 2 appears in the indicators. Place the first card in the ready station by pressing:

LOAD PRI
LOAD SEC
CLEAR
START
RUN

(7) Ensure that Step 2 light is on.

B16.3. Error Indications

When the Card Controller IOCS routine detects an error, it displays the function on which the error occurred, except for the 650F display. Possible displays are as follows:

9200/9300 DISPLAY	1001 FUNCTION	9200/9300 MODE
6500	Transfer Primary	1001 or TRANSTC
6501	Transfer Secondary	1001 or TRANSTC
6502	Transfer Both	1001 or TRANSTC
6504	Transfer Primary	Binary, CC, or TRANS
6505	Transfer Secondary	Binary, CC or TRANS
6508	Transfer Primary and Read	1001 or TRANSTC
6509	Transfer Secondary and Read	1001 or TRANSTC
650A	Transfer Both and Read	1001 or TRANSTC
650C	Transfer Primary and Read	BINARY, CC, or TRANS
650D	Transfer Secondary and Read	BINARY, CC, or TRANS
6514	Receive Data	–
6524	Send Data	–
650F	Not Applicable	Not Applicable

The 650F display indicates either a device address parity error has occurred or the Card Controller is offline. Check the special adapter offline switch located in the lower left area at the back of the Printer Processor Cabinet. Press the START switch on the UNIVAC 9200/9300 to reissue the XIOF. If the problem persists, there may be a hardware malfunction.

For any Card Controller function display other than the 650F, if the cause of the error cannot be determined from the indicator lights on the Card Controller and from the function displayed on the UNIVAC 9200/9300 console, the IOCS routine can be requested to display the Card Controller sense byte in an attempt to determine the cause of the error. The request for this second display (STOP 2) is made by keying a hexadecimal 01 into location 4 and then pressing the START switch.

B16.3.1. STOP 1 (65xx)

The first two characters (65) of the display identify the stop as a Card Controller stop. The next two characters (xx) represent the Card Controller function being executed.

The operator should note the stop and check the 9200/9300 panel lights. The Card Controller indicator lights then determine the specific type of error.

Table B–1 lists the possible Card Controller displays at STOP 1. Indication for the display is given either in the control panel lights or in the display mask indicator lights.

CARD CONTROLLER DISPLAY	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
PRI or SEC MISF	Hopper empty or empty ready station due to damaged card in input hopper.	Recoverable	Correct condition (replace damaged card if necessary). Press PRI or SEC LOad, RUN on the Card Controller. Press START on 9200/9300 control panel. (XIOF will be reissued.)
STKR FULL	Stacker associated	Recoverable	If STEP=02,03,04: Correct condition. Press CLEAR,START,RUN on the Card Controller. Press START on 9200/9300 panel. If STEP≠02,03,04: Correct condition. Press RUN on the Card Controller. Press START on 9200/9300 panel. (XIOF will be reissued.)
P or SW JAM	Card jam at wait 1 or wait 2 station	Recoverable only if stacker selection for jammed cards is known by operator.	Remove the jammed cards by opening the back frame. Repair damaged cards. Manually place them in the appropriate stackers. Follow instructions for STKR FULL.
		Otherwise, non-recoverable	Go to RESTART.
STKR JAM	Card jam after wait 2 station.	Recoverable only if stacker selection for jammed cards is known by operator.	Remove the jammed cards by opening the back frame. Repair damaged cards. Manually place them in the the appropriate stacker. Follow instructions for STKR FULL.
		Otherwise, non recoverable	Go to RESTART.
P or S RD JAM	Card jam at Read	Recoverable	Card at wait 1 must be reread. Remove cards from ready station and wait 1 station by opening back frame. Place the 2 cards in proper input hopper. Press PRI or SEC LOAD, RUN on 1001. Press S TART on 9200/9300 control panel (XIOF will be reissued).
PRGM HALT	Possible program error.	Nonrecoverable	Take control counter reading and a memory dump. Go to RESTART to try again.
STOP switch lighted	Switch depressed	Recoverable	Follow instructions for STKR FULL.
INTERLOCK (OFF switch lighted)	Door or frame open	Recoverable	Follow instructions for STKR FULL.
POWER OFF (OFF switch lighted)	Power dropped	Recoverable in some cases	Check the wall plug and circuit breaker at the rear of the Card Controller. Press the ON switch and momentarily hold it on. The ON indicator should light and the OFF indicator should be extinguished. If power dropped at the very beginning of a program, continue the program. (XIOF will be reissued.) If power dropped while a program was running, go to RESTART.

Table B–1. UNIVAC 1001 Card Controller IOCS Initial Error Indications

B16.3.2. STOP 2 (65yy)

When the cause of an error cannot be determined from the STOP 1 display, the Card Controller sense byte can be displayed by keying a hexadecimal 01 into location 4 and then pressing the START switch. The first two characters (65) of the display identify the stop as a Card Controller stop. The next two characters (yy) represent the Card Controller sense byte.

The sense byte may reflect a combination of bit settings. Therefore, corrective action may have to be taken for more than one error condition before an attempt can be made to continue.

Table B–2 lists the sense byte displays for STOP 2. It also gives the reason for the stop and the action required to correct the error.

STOP 2	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
6580	Unspecified command or bad parity in command sent to 1001 control.	Recoverable in some cases	Press START to reissue order. If it fails again, take control panel reading and memory dump. Suspect program error or possible hardware malfunction.
6540	Operator intervention required.	Recoverable in some cases	Check INTERLOCK/POWER OFF switch plus mask position 6 in order to determine action required.
6520	Parity error on BUS OUT check.	Recoverable in some cases	Press START to reissue order. If error persists, suspect possible hardware malfunction. Go to RESTART.
6510	Data parity error causing a selective reset to 1001 control or interface error or address error or device address parity error on BUS IN.	Nonrecoverable	Go to RESTART.
6508 or 6504	Control logic flip-flop set (primarily intended to be used in conjunction with failure-finding programs).	Nonrecoverable	Should never happen. Take control counter reading and memory dump.
6502	Inhibit status	Not an error	No action required by operator. This display can occur only in conjunction with some other display.

Table B–2. UNIVAC 1001 Card Controller IOCS Requested Error Indications

B17. ROW READ/PUNCH DEFINITION STATEMENTS

The following sections describe the Row Read/Punch definition statements for punch only, read only, and read/punch operation.

B17.1. Punch Only

LABEL	OPERATION	OPERAND	72
filename	DTFRW	CHNL=5 thru 12,	X
		CNTL=YES,	X
		MODE=BINARY,	X
		or	
		MODE=CC,	X
		or	
		MODE=TRANS,	X
		OUAR=location of the output buffer area,	X
		OTBL=location of the output translate table,	X
		TYPF=OUTPUT	

CNTL is required if the CNTRL macro instruction is used by the problem program.

OTBL is required only if MODE=TRANS.

If TYPF is not specified, output will be assumed.

CHNL, MODE and OUAR must always be present.

B17.2. Read Only

LABEL	OPERATION	OPERAND	72
filename	DTFRW	CHNL=5 thru 12,	X
		CNTL=YES,	X
		EOFA=location of the user end-of-file routine,	X
		INAR=location of the input buffer area,	X
		ITBL=location of the input translate table,	X
		MODE=BINARY	X
		or	
		MODE=CC,	X
		or	
		MODE=TRANS,	X
		TYPF=INPUT	

CNTL is required if the CNTRL macro instruction is used by the problem program.

ITBL is required if MODE=TRANS.

CHNL, EOFA, INAR, MODE and TYPF must always be present.

B17.3. Read and Punch

LABEL	OPERATION	OPERAND	72
filename	DTFRW	CHNL=5 thru 12	X
		CNTL=YES,	X
		EOFA=location of the user end-of-file routine,	X
		INAR=location of the input buffer area,	X
		ITBL=location of the input translate table,	X
		MODE=BINARY,	X
		or	
		MODE=CC,	X
		or	
		MODE=TRANS,	X
		OUAR=location of the output buffer area,	X
		OTBL=location of output translate table,	X
		ORLP=YES,	X
		TYPF=COMBND	

CNTL is required if the CNTRL macro instruction is used by the problem program.

ITBL and OTBL are required if MODE=TRANS.

ORLP is required when *not* punching into the same card made available by the last GET macro instruction.

CHNL, EOFA, INAR, MODE, OUAR and TYPF must always be present.

B17.4. Buffer and Work Area Size

The input buffer area and work area are each 80 bytes long if MODE=CC or TRANS, and are each 160 bytes long if MODE=BINARY.

The output buffer area and work area lengths are each equal to the number of columns to be punched if MODE=CC or TRANS; and are each equal to two times the number of columns to be punched, if MODE=BINARY.

B17.5. End-of-File

POSITION OF EOF CARD WHEN GET TRANSFERS TO EOFA

	READ		PUNCH		
READ ALONE	C	B	A	EOF	
READ & PUNCH OVERLAP	C	B	A	EOF	
READ & PUNCH NONOVERLAP	B	A	EOF		

EOF CARD is in work area (translated if required).

Following card, A_2 is in the input buffer area INAR.

Next card, B, is in the Row Punch read buffer verified for Read alone and for Read & Punch overlap.

B17.6. Preparing the Row Read/Punch

(1) Place the unit offline.

(2) Make certain that the AC/DC light is on and the INTL/READY light is on.

(3) Preload the punch track with blank cards by pressing manual FEED followed by CLEAR until a blank card is fed into the output stacker. (Jam indications will occur after each of the first three feeds.)

(4) Place the input deck into the input hopper, face down, 9-edge leading.

(5) If reading only or reading and punching, feed one card.

(6) If punching only, feed three cards.

(7) Press CLEAR on the row read/punch.

(8) Place the unit online. At this time:

- The AC/DC light *must be on.*
- The INTL/READY light *must be on.*
- *All other lights must be off.*

NOTE: In order to clear error lights on the unit:
(1) Correct the error condition at the unit.
(2) Press:
(a) OFFLINE
(b) CLEAR
(c) OFFLINE

B17.7. Error Indications

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
Looping on A5, 47 instruction	Unit nonoperational (in OPEN coding).	Recoverable.	Correct the condition to continue.
640F	Unit nonoperational.	Recoverable.	Correct the condition. Press START to continue.
6401	Hole count error.	Recoverable for punch alone.	The error card and following card are diverted to the error stacker. Press START to repunch last two cards and continue. (NOTE: if PUNR=YES, this stop will not occur as IOCS will provide automatic recovery.)
6402 INTL light on and no error lights on	Parity error on data transfer during execution of a LOAD or UNLOAD XIOF.	Recoverable.	All cards in wait stations are assumed to be good. Press START to continue. (XIOF will be reissued.) If the card at the post punch station is bad or if the error persists, suspect a hardware malfunction. Restart.
6402 error light on	HOPPER empty. CHIP box full. STKR full. STKR jam.	Recoverable.	Correct the condition. Press OFF LINE, CLEAR, OFF LINE. Press START to continue. (XIOF will be reissued.)

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
6402 INTL light off	One of following conditions: ■ Punch assembly and upper card feed raised or improperly seated. ■ Read station brushes not in place. ■ Protective covers not in place.	Recoverable.	Correct the condition. The INTL, light must be ON. Press OFF LINE, CLEAR, OFF LINE. Press START to continue. (XIOF will be reissued.)
6402 A JAM or B JAM Light on	Misfeed or card jam.	Recoverable in some cases for read only or punch only.	READ ONLY: Remove the cards from the input hopper. Place the unit offline. Open the punch frame. Remove *all* cards from the track. Repair the damaged card. Place the cards in the input hopper. Replace the rest of the deck in the input hopper. Press CLEAR on the unit. Feed three cards. Close the punch frame. Press CLEAR, OFF LINE. Press START to continue. (XIOF will be reissued.) If nonrecoverable, restart. PUNCH ONLY: The card at the post punch station is assumed to be in error. Place the unit offline. Open the punch frame. Remove *all* cards from the track and discard. Press CLEAR on the unit. Feed three cards. Close the punch frame. Press CLEAR, OFF LINE. Press START to continue. (XIOF will be reissued.) If nonrecoverable, restart.

STOP	REASON FOR STOP	RESULTING CONDITION	OPERATOR ACTION
6404	Device end.	Not an error.	No action is required by the operator. These displays can only occur in conjunction with some other error.
6408	Channel end.	Not an error.	
6410	Busy.	Not an error.	
6440	Status Modifier.	Not an error.	
Power OFF	Power dropped during execution of program.	Nonrecoverable.	Restart.

B18. IOCS GENERATION

The Preassembly Macro Pass for a card reader, serial punch, 8K UNIVAC 9200/9300 configuration allows approximately 2700 bytes for macro library storage. IOCS macro library routines require approximately the following number of bytes of storage:

ROUTINE	NUMBER OF BYTES
Reader (DTFCR)	600
Printer (DTFPR)	1200
Serial Read/Punch (DTFRP)	1800
Card Controller (DTFCC)	2200
Row Punch (DTFRW)	2400

B19. ADDITIONAL KEYWORD PARAMETER SPECIFICATIONS

Certain additional keyword parameter specifications have been built into the DTFCR, DTFRP, and DTFRW macro definitions to provide functions designed primarily for Univac Systems Programming use. These parameters are available and can be used by the user if he so desires. These parameters are as follows:

- SENT=X 'value' – The IOCS routines test for standard end-of-file sentinels (slash in column 1 of the input cards). The above parameter may be used if something other than the standard sentinel is to be tested. If MODE=CC or TRANS, the IOCS routine tests word zero in the input buffer against X'34'. If MODE=BINARY, the IOCS routine tests for X'OC'. Only one byte is tested; therefore, the *value* specification must be only one byte in length.

- SENT=NO – This parameter may be used if no EOF sentinel test is to be made by the IOCS routine. Coding for the EOF test will be eliminated during the macro pass if SENT=NO is present.

- INCR=m – This parameter may be used if the user expects the EOF sentinel in a column other than column 1. The specification m is a variable, numeric quantity indicating column n–1.

APPENDIX C. CARD LOAD ROUTINE

C 1. GENERAL

The Card Load Routine for the online card reader and the 80-column UNIVAC 1001 Card Controller consists of the following sections of coding:

a. Bootstrap coding to read the Load Routine into memory. Before transferring control to the Load Routine, the bootstrap coding sets the EBCDIC mode and enters the processor state.

b. Coding to clear a selected portion of memory to a selected character. This coding is executed before the Load Routine itself is read into memory. If the area specified to be cleared includes the Load Routine and its read area, they are not cleared.

c. Coding to load a program in Assembler output format into the internal storage of the UNIVAC 9200/9300. The Load Routine performs a hole count check of each card used. Upon encountering a Transfer Card (a card with Y in column 2) signifying termination of loading, the Loader compares the number of External Reference (type K), Text (type Y), and blank cards read with the number contained in columns 12 and 13 of the Y card. If the numbers agree, the Load Routine loads register 13 with the address at which to begin program execution and transfers control to that address. This is the address contained in columns 15 and 16 of the Y card. If these columns are blank, the transfer address used is that contained in columns 15 and 16 of the program reference card (type J).

   If the card count check fails, the Load Routine halts. At this point pressing the START switch causes the Load Routine to begin execution of the program just loaded. During the loading of the EXEC I portion of a program, the Load Routine sets up the I/O PSC for EXEC I.

The name of the Load Routine for the online card reader is LD; for the Card Controller, LDCC.

C 2. PARAMETERS FOR THE LOAD ROUTINE

The Load Routine can be maintained as an object code deck ready to be linked. Certain labels exist as external references. Defining these labels supplies the variable information required by the Load Routine. These labels may be defined by means of EQU control cards at the time the Load Routine is linked to a program. The external labels and their meaning are shown below. See 5.6 for further use of these labels.

LABEL	MEANING
L?AR	Start of the read area for the Load Routine.
L?PG	Start of the coding of the Load Routine.
L?LO	First memory location to be cleared.
L?HI	Last memory location to be cleared.
L?CH	Character with which to fill the area to be cleared.
L?AM	The value assigned determines whether alterations are to be stored in memory location 4 or the memory location specified in the address switches. If the value assigned is four, alterations are stored in location 4; if zero, in the location specified by the memory address switches.

Note that all labels used by the Load Routine begin with the characters "L?".

The read area for the Load Routine is 80 bytes long and does not have to be contiguous with the Load Routine, but each must begin on a halfword boundary. The coding for the Load Routine is approximately 275 bytes long in the case of routine LD and 360 bytes long in the case of routine LDCC.

The Load Routine is coded relative to the labels L?AR and L?PG. Thus, once these two labels are defined by EQU cards, the location at which the Load Routine and its read area reside in memory is fixed. In setting up the Linker input deck the user must be certain that the Linker does not allocate to any other part of the phase in which the Load Routine appears the memory that has been allocated to the Load Routine by the L?AR and L?PG EQU cards.

When the Load Routine is linked to a program, it must appear as the first element in the input deck for the first phase of the program being linked. To produce a self-contained loader that will load the object program that follows it, add an END card with an operand of L?PG to the Load Routine deck and link the loader separately.

For routine LDCC, L?AR must be set equal to 160 or more.

C3. LOADING ADDITIONAL PROGRAMS

If the Load Routine is in memory, it may be used to load another program by branching to the initial location of the Load Routine (represented by the tag L?PG). The program to be loaded must not overlay the Load Routine or its read area. The first card loaded will be the card at the wait station at the time L?PG is branched to.

A terminating program may also initiate the loading of a successor program if the successor contains a load routine of its own. The program must read the first card (bootstrap card) of the successor into a location in memory, set the address of that location into register 15, and transfer to that address plus 26. The address chosen for the bootstrap card must not overlap either the load routine of the successor program or the read area of the Load Routine. The bootstrap card must be read in in compressed code and must not be translated. This facility is possible only if the online card reader is being used to load programs.

C4. LOAD ROUTINE STOPS

DISPLAY	MEANING	ACTION
4300	Card Count error	Press START to begin execution of program just loaded.
61ss (LD) 65CE (LDCC)	Hole count error: sum of the bytes read from columns 8–72 does not agree with the hole count byte column 7 (ss has no meaning).	Press START to ignore the card and continue loading.
61ss (LD) (Reader abnormal light on)	Card Read Error (ss is the status byte)	Press READER, CLEAR. Refeed the error card (if any). Press START to continue reading.
6504	Card Controller error	Nonrecoverable. Less than 10 cards have been read. Start over.
650C	Card Controller error while trying to execute a TR & RD PRI or TR PRI function	Recoverable in some cases. Cause of error may be determined by the display lights on the Card Controller. Check the masks and follow the error recovery procedure as outlined under STOP 1 of the Card Controller IOCS operating instructions; however, the sense byte display (STOP 2) cannot be made available in the LOADER.

The 61ss displays are applicable to the LD Load Routine only.

In the case of a card read error, the error is at the top of the output stacker and is present unless the error is misfed, not ready, offline, hopper empty, or stacker full.

C5. DESCRIPTION OF OPERATION

The Card Load Routine consists of four parts: Boot, Reader, Clearing, and Loader.

C5.1. Bootstrap Section

The Bootstrap section is the first card in the Load Routine and is read into privileged memory (0–79) by a console operation. The Bootstrap section is a card reader routine that reads the next card in the input hopper into L?AR and exits to the first instruction (at L?AR+10) on the card read. The first instruction at L?AR+10 stores the remainder of the card into L?PG. The instruction that follows transfers control back to the Bootstrap section. This cycle of reading and storing of cards continues until the Reader and Clearing sections are stored in L?PG. The last card of the Clearing section read into L?AR sets all alteration switches as determined by the value in L?AM, sets further operations under processor PSC, and transfers control to the Clearing section coding in the L?PG area.

C5.2. Clearing Section

The Clearing section clears memory specified to the specified character except for the privileged area and the L?PG area. Control is then transferred to the Reader section in the L?PG area to read in the Loader section.

C5.3. Reader Section

The Reader section reads the next card in the input hopper into L?AR and exits to the first instruction on the card which stores the remainder of the card into L?PG above the Reader section. The second instruction on the card transfers control back to the Reader section in L?PG to read the next card. The cycle continues until all the cards making up the Loader Section are read and transferred into L?PG. The last card on the Loader section read into L?AR changes the transfer-control address in the Reader section from L?AR+10 to the Loader section. Control is then transferred to the Loader section.

C5.4. Loader Section

This section remains in the L?PG area with the Reader section during the loading process. The Reader section brings in the program cards. The Loader section checks the hole count and the card count, loads the program from the Text (Q) cards, and on detection of a type Y (END) card, determines the start address, places the start address in register 13, and transfers control to it. The Loader section loads the I/O PSC with the first four bytes of data (columns 11–14) of any type Q (Text) card with a load address of 16.

C6. PROGRAMMING CONSIDERATIONS

In situations where memory is at a premium, it is possible to use the problem program I/O areas to store the Loader and then overlay the Loader with input or output data. This can be accomplished by the following example:

- Coding

LABEL	OPERATION	OPERANDS	COMMENTS
	CTL	2,8191,8191	
	PHASE	UNI,256,A	Program UNI will start at location 256 absolute.
L?AR	EQU	160	80 byte read-in area begins at location 160.
L?PG	EQU	240	Loader will be loaded starting at location 240.
L?HI	EQU	8191	
L?LØ	EQU	160	Locations 160–8191 will be cleared to zero before loading loader.
L?CH	EQU	0	
L?AM	EQU	4	All alterations will be stored in location 4.

- LD or LDCC object code deck
- Problem program starting with DS statements for I/O areas and working areas.
- Other elements
- END card

In the above example, the problem program must have the DS statements for I/O areas and working areas as the first lines of coding. The total memory requirements of all DS statements must be greater than 275 bytes when using the card reader loader, or 360 bytes when using the 1001 Card Controller loader. If the total of the DS statements is less than this, the starting address in the PHASE card must be adjusted to make up the difference so that the loader will not be overlayed before loading has been completed.

Text cards are not punched for DS statements; therefore, nothing is loaded into these memory locations at load time. Thus, if this approach is used, the problem program must clear the output areas before using them, as they may contain parts of the Loader routine. It should be noted that the loader is then destroyed and must be loaded again for any subsequent programs.

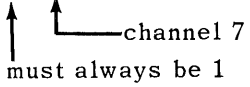
The example above also can be used for creating a freestanding loader, except that no problem program or other elements will follow the loader deck. Still required is a PHASE card, in which any fictitious parameters may be used.

C7. LOADING FROM CARD READER.

1. Place program deck in hopper.
2. Press READER CLEAR, READER FEED.
3. Set DATA ENTRY switches to 0000 0001.
4. Press general CLEAR.
5. Set LOAD.
6. Press START.
7. Reset LOAD.
8. Press START.

C8. 1001 LOADER LOADING PROCEDURE

1. Remove all cards from the PRI hopper of the Card Controller by pressing UNLOAD PRI three times.
2. Place program deck in PRI hopper.
3. Set the ALT 1 switch on.
4. Press the CLEAR, LOAD PRI, CLEAR, START, RUN, switches on the Card Controller.
5. Set the ALT 1 switch off.
6. Enter the device address in the DATA ENTRY switches of the 9200/9300. (Example: The Card Controller is on channel 7; therefore, enter B8 in DATA ENTRY switches. B8 = 10111000)



7. Press the PROC CLEAR, CHANNEL CLEAR switches on the 9200/9300.
8. Press the LOAD switch on the 9200/9300.
9. Press the START switch on the 9200/9300.
10. Reset the LOAD switch on the 9200/9300.
11. Press the START switch on the 9200/9300.

APPENDIX D. EXEC I

D1. GENERAL

EXEC I is designed for UNIVAC 9200/9300 Card System only and takes the form of a relocatable element which must be included in the worker program at Linker time. The primary functions of EXEC I are to monitor interrupts, handle messages to and from the operator, and provide restart communication.

D2. MACRO INSTRUCTIONS

EXEC I provides the following macro instructions:

D2.1. Message Macro (MSG)

The message macro has the format given below:

OPERATION	OPERAND
MSG	Message,REPLY

The REPLY parameter is optional. Message may be any acceptable two-byte hexadecimal expression of the form X'nnnn'.

This macro generates the following code:

OPERATION	OPERAND
SRC	0,8
DC	Y(message)
DC	CL1'x'
DC	X'0'

where message is the two-byte hexadecimal display which appears in the HPR instruction. It takes the form of an assembler language expression.

x = A, if the parameter REPLY appears; x = a blank (EBCDIC code 01000000), if it does not.

The one-byte reply, keyed in by the operator into location 4, appears in the last byte of the calling sequence.

EXEC I responds to this macro by doing a BAL, using register 15, to its own display subroutine. It moves the message from the calling sequence of the SRC instruction to the calling sequence of the BAL instruction before executing the BAL instruction. The display subroutine sets location 4 to binary zero and displays the message by means of an HPR instruction. When the START switch is depressed, the display subroutine returns control through register 15. EXEC I then moves the contents of location 4 to the reply byte of the calling sequence and returns control to the problem program.

For example, if the user codes the following macro instruction,

```
DSPL    MSG    X'FFF',REPLY
```

the Assembler treats this macro instruction the same as the following source code:

```
DSPL    SRC    0,8
        DC     Y(X'FFF')
        DC     CL1'A'
        DC     X'0'
```

When the object code produced from this source code is executed, the computer stops with a display of 000111111111111. The operator may then answer this display using the DATA ENTRY and ALTER switches. When the START switch is subsequently depressed, control is returned to the user's coding at the instruction located at DSPL + 8. The byte inserted into the computer by the operator through the DATA ENTRY and ALTER switches is in location DSPL + 7. If the operator did not introduce any data through the DATA ENTRY and ALTER switches, then on return of control to the problem program, the byte in location DSPL + 7 contains binary zeros.

The MSG macro instruction is not handled by the Preassembly Macro Pass, but is processed by the Assembler itself.

D2.2. Restart Macro

The restart macro has the following format:

OPERATION	OPERAND
RSTRT	Restart-name

The restart-name is the label of a user-coded routine which is designed to handle a restart operation.

This macro generates the following code:

OPERATION	OPERAND
SRC	0,0
DC	Y(Restart-name)

In response to this macro instruction, EXEC I stores the address of the restart-name. Restart is accomplished by a general clear followed by depression of the START switch. This causes the instruction in memory locations 22 through 25 to be executed in I/O mode. EXEC I has a branch unconditional instruction in this location that allows it to force all alterations to be stored in memory location 4, set the processor PSC to the restart-name and then go to RE-ENTRY. At RE-ENTRY, EXEC I sets the device address byte to zero, resets (without destroying the SRC field) the I/O PSC in preparation for the next interrupt, and returns to processor state.

At the restart-name location the user must provide a restart routine. This routine must re-establish variable information in the program and set initial conditions for all input/output routines. (To aid the user in accomplishing this goal, the execution of the OPEN macro resets the initial conditions for all IOCS routines.) The user must establish conventions to reposition card decks and printer paper.

The RSTRT macro instruction is not handled by the Preassembly Macro Pass, but is processed by the Assembler itself.

D3. I/O CONTROL ROUTINE MESSAGES

All IOCS routines operating in I/O mode may display messages through direct access to the display subroutine. After execution, if a reply is expected, the control routine itself must examine the contents of location 4.

The following instructions are required to execute a display:

OPERATION	OPERAND
BAL	15,E?DS
DC	XL2'message'

where E?DS is the label for the first byte of the display routine.

message is a two-byte hexadecimal expression.

APPENDIX E. TRANSLATION TABLES

E1. GENERAL

The UNIVAC 9200/9300 I/O software enables any desired internal 8-bit code to result from the reading of most punch configurations, to be punched into nearly any desired punch configuration, and to cause printing of any character of a 63- or 48-character font. This is accomplished by the optional use of automatic translation in the I/O routines. Input/output data is translated according to specified translation tables. The programmer may use his own translation tables or he may use any of the standard translation tables.

Standard translation tables are available in source code form and in object code form. The source code form can be assembled with another program or can be assembled by itself to provide a linkable element. The object code form is a relocatable element with its name defined by the External Definition (type H) card. Each table occupies 256 bytes of object memory.

There are three standard translation tables; Table Read, Table Punch, and Table Print. Each table is used by the appropriate, associated I/O routine.

Table Read, TBRD, converts the compressed image read from Hollerith-coded input cards to internal EBCDIC code.

Table Punch, TBPU, converts the internal EBCDIC code to a compressed image that produces output cards punched in Hollerith code.

Table Print, TBPR, converts the internal EBCDIC code to a code which causes the 48-character print bar to print the same character as would be printed if the 63-character print bar were used. Characters other than 0–9, A–Z, and .+&$*–/,%'#@ are treated as blanks.

UNIVAC
FEDERAL SYSTEMS DIVISION
UP 4092 – Rev 2 MAY 1968