TRIM III

ASSEMBLER

USER'S INSTRUCTIONS

SECTION III-C. TRIM III ASSEMBLY SYSTEM

1. INTRODUCTION

The TRIM III assembly system provides programming assistance through the use of its symbolic shorthand. As illustrated in Figure III-C-1, this assembly system converts a source program written with symbolic addressing into an object program with absolute or relocatable addressing. TRIM III produces the assembled object program on punched paper tape, punched cards, or magnetic tape.

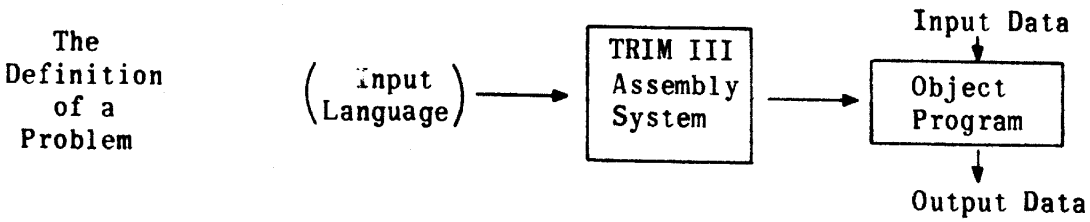


Figure III-C-1. TRIM III Solution of a Problem

TRIM III has an easy-to-use but effective library retrieval capability. The library of subroutines is stored on the assembler magnetic tape. The user simply calls by name those subroutines he wishes to include with his assembled program. TRIM III honors the calls by automatically adding them to the end of the source program during assembly. A companion program to TRIM III called the library builder routine provides easy library building, insertion, replacement, deletion, and listing capabilities.

TRIM III possesses source language level correction capability in combination with an assembly run. Although this feature is primarily designed for use with paper tape input, it may be used with any combination of input modes.

2. DESCRIPTION

TRIM III is basically designed for a minimum equipment configuration of a computer with at least 16,384 words of core memory, a magnetic tape system with two or more tape transports, and an I/O console consisting of a punched tape reader, tape punch, keyboard, and console typewriter. Optional equipment is an on-line card processor system with card reader, card punch, and high-speed printer.

The TRIM III assembler is stored on magnetic tape in functional segments. During an assembly run the segments are read into computer memory and executed in the proper sequence by the assembler controlling routine. See Figures III-C-2 and III-C-3. TRIM III is a single external pass assembler. It accepts a source program, converts it to TRIM code, and stores it on magnetic tape for subsequent processing. If the user has included calls for library subroutines in his source program, TRIM III selects them from the library and adds them to the end of the source program before proceeding with assembly. TRIM III also has source language correction capability in conjunction with an assembly run.

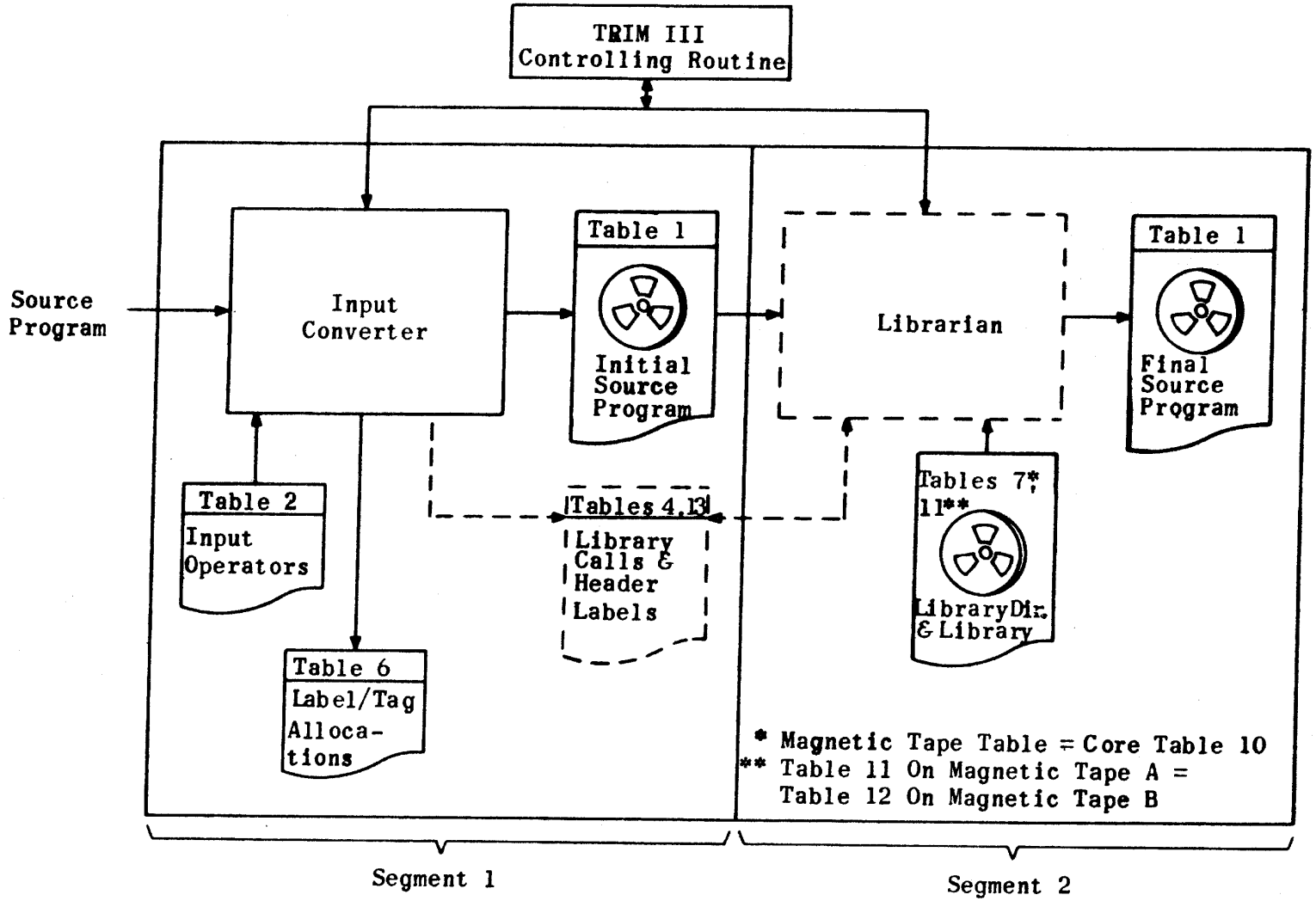
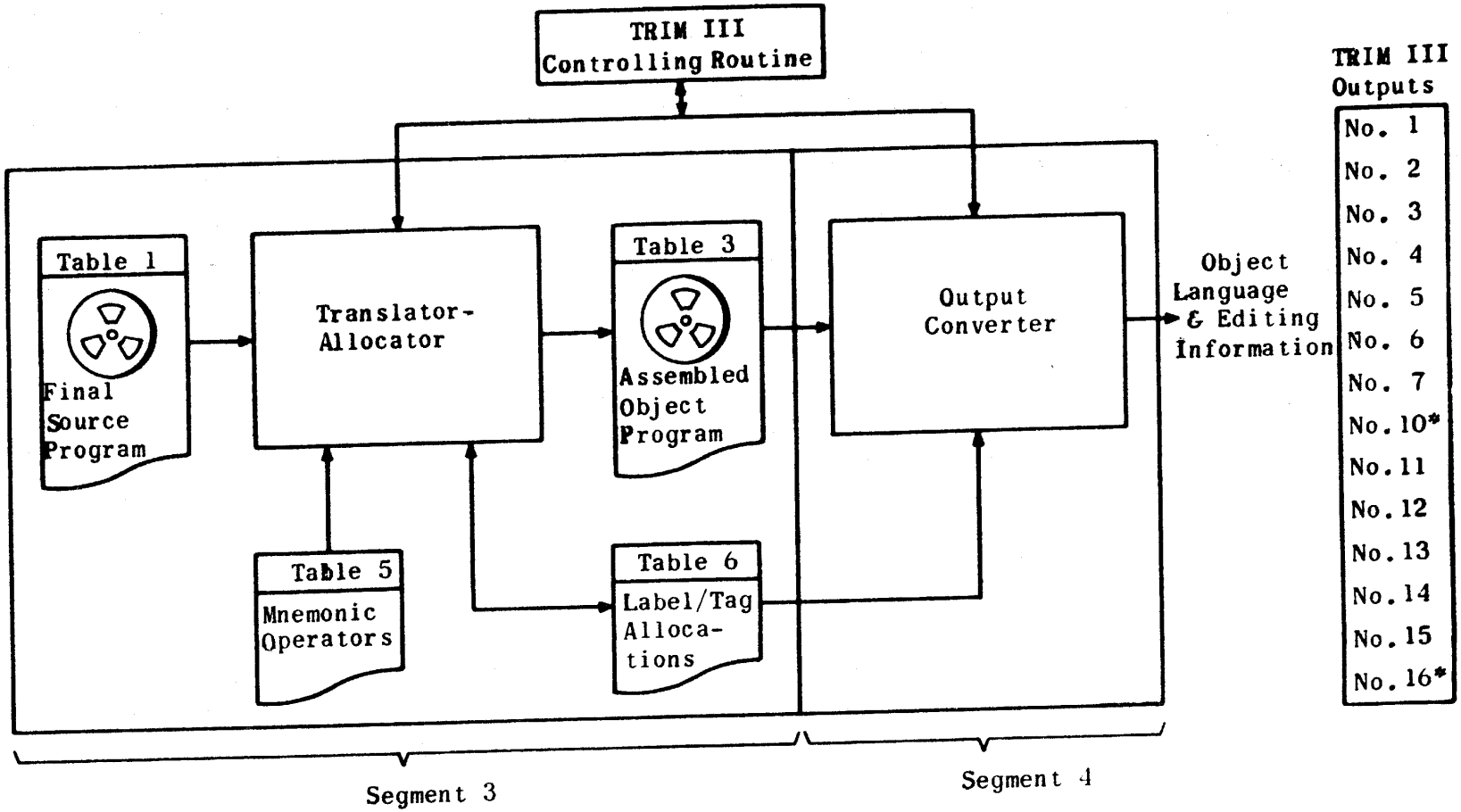


Figure III C-2. TRIM III Segments 1 and 2

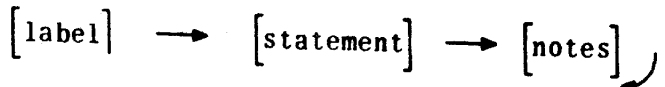


* Output No. 10 is Table 3; Output No. 16 is Table 15

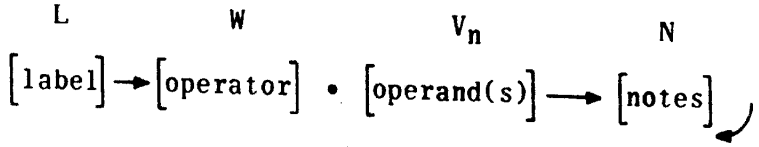
Figure III-C-3. TRIM III Segments 3 and 4

2.1 SOURCE LANGUAGE

A TRIM program as prepared by the programmer is composed of a list of operations which perform the step-by-step processing of a problem. An operation has the following general format:



The general format may be further subdivided into:



2.1.1 LABEL

The label identifies this particular statement. A label is not required for every statement. In an absolute-addressed program every word is assigned an absolute address during the coding process. The assembling process of the TRIM III system equates the label to the machine address assigned to the instruction generated by the statement. Only those statements which are referred to by other statements require a label or symbolic address. Where more than one instruction is generated by a statement, the label refers to the address of the first instruction generated. The term label is used rather than address since it more accurately describes the function of the symbolic address. A label may never be incremented or decremented. The instructions or words generated from unlabeled statements following one another on the source program tape are ultimately assigned to consecutive memory addresses. Each label of an assembly run must be unique.

A label may consist of not more than six alphanumeric characters; it never begins with the letter O or a number, and never consists of the leters LOK alone. The first instruction of each program or subroutine must have a label.

An operand which refers to another operation label is called a tag. The tag must be identical with the label it refers to except that it may be followed by a $\pm$ octal or decimal integer to facilitate reference to unlabeled operations. Whenever a decimal integer is used, it must be followed by the letter D. A tag coincides with the u or k portion of the instruction word. Tags have the same notation restrictions as labels except they may be incremented. Any number of tags may refer to a given label.

If the programmer wishes to reference unlabeled instructions in his program in another manner, he may do so in terms of a specific instruction by means of the LOK tag plus or minus an integer. LOK always refers to the instruction in which it appears. For example, if the instruction JP•LOK-3 appears at address 04503, the resulting generation is a jump to address 04500. Thus the instruction falling at address 04500 need not have been labeled. No valid program label may consist only of the letters LOK. Reasonable care should be exercised in the use of the LOK tag since corrections to the original program may affect the LOK references.

2.1.2 STATEMENT

The statement of an operation is made up of an operator and operand(s). The statement defines the operation.

2.1.2.1 OPERATOR

The operator may be a symbolic shorthand or octal notation which identifies the basic function to be performed. The operator must always be present. It may cause the assembler to generate one machine instruction or a group of machine instructions. The operator coincides with the function code f, and/or sub-function code m, of the instruction word.

2.1.2.2 OPERAND(S)

One or a series of operands associated with the basic operator are referred to as V_0, V_1 ... V_n. These may take several forms depending upon the basic operator. They define, modify, or complete the function.

The operand(s) coincides with the u or k portion of an instruction word and may be either a constant in octal notation or a symbolic alphanumeric notation referring to a constant (either an absolute address or an item of data).

2.1.3 NOTES

Descriptive notes may follow the statement; they are for the programmer's use and in no way affect the instructions generated from the statement. Notes must be restricted in length such that the entire source statement does not exceed one line or one card.

2.1.4 SYMBOLS

The program uses a uniform set of symbols as separators in all coding. These symbols are depicted in Table III-C-1 below.

TABLE III-C-1. TRIM III CODING SYMBOLS

Symbol		Coding Significance
→	(tab)	Major separator delimiting the statement. Must always precede the statement operator. Must precede notes; omitted if notes are not given.
↵	(CR)	Specifies the end of an operation. Must precede end-of-tape double period.
,	(comma)	Separates certain subsets of statement components.
•	(point)	Separates statement components.

TABLE III-C-1. TRIM III CODING SYMBOLS (CONT.)

Symbol		Coding Significance
+		Specifies an integer increment to follow.
-		Specifies an integer decrement to follow.
Δ	(delta)	Specifies space.
|	(vertical line)	Special control character.
..	(double period)	Specifies end-of-tape read-in. Must terminate every input tape.

2.2 HEADER AND DECLARATIVE OPERATIONS

TRIM III recognizes four types of header operations:

L	W	V_0	V_1	N
POKER →	CONTR •	JONES •	10 DEC1964 →	
POKER →	ALLOC •	JONES •	10 DEC1964 →	
POKER →	PROG •	JONES •	10 DEC1964 →	
POKER →	CORREC •	JONES •	10 DEC1964 →	

2.2.1 CONTROL HEADER (CONTR)

The CONTR header operation is a convenience for the user. It enables him to group all of his assembler declarative operations following one CONTR header. A label and identifying operands may be used with the CONTR header, but TRIM III does not require them.

L	W	V_0	V_1
[label] →	[CONTR] •	[name] •	[date] →

Operations which may follow a CONTR header are ALLOC, DEBUG, OUTPUT, REMARK, DECKID and CALL. CALL operations may also follow a PROG header. Figure III-C-4 shows typical coding for a CONTR header and the declarative operations used with it.

TITLE _____ **UNIVAC CODING FORM** PROGRAMMER _____

PAGE _____ of _____ PLT. _____ EXT _____ MS _____

DATE _____

LABEL	OPERATOR		OPERANDS AND NOTES
	HEADER TYPE		
POKER	→ CONTR	•	JONES • 16NOVEMBER1963
POKER	→ ALLOC	•	JONES • 16NOVEMBER
POKER	→ 05000	•	
CHIP	→ 05500	•	
DEBUG	→ 13000	•	
TYPT	→ 12700	•	
	→ OUTPUT	•	1•6•2•5•6•11
	→ DEBUG	•	
	→ CALI	•	SINE TODEC TYPT
	→ REMARK	•	CONTR TAPE FOR DATAX REVISION 3
	→ DECKID	•	SINE
..	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	
	→	•	

Figure III-C-4. Sample CONTR Header And Delcarative Operations

2.2.2 ALLOCATION HEADER (ALLOC)

The ALLOC header follows a CONTR operation and informs TRIM III that the operations which follow constitute assignments of absolute values to labels and/or tags. Any number of ALLOC tapes or cards may be loaded. An allocation tape must always be preceded by a carriage return (see paragraph 3). When the allocations are on a separate tape, the tape must terminate with a carriage return and two periods. ALLOC operations have the following format:

```
   L          W          V0         V1
[label] → [ALLOC]  •  [name]  •  [date]↵
[label]-[assigned value]↵
[label] [assigned value]↵
[label] [assigned value
            etc.       ]↵
```

1) L - The label of the ALLOC header operation itself is optional. However, each assignment operation following must have a label.

2) W - The operator of this header operation is always ALLOC, and must be present. For the subsequent assignment operation, W must be an absolute numeric value expressed either in octal or decimal. When expressed decimally, the number must be followed by the letter D, for example:

```
CAT   → 0100↵
DOG   → 512D↵
CHIPS → 12↵
CHOPS → 10D↵
```

3) V - The V operands of this header operation take the form name and date as illustrated. These operands are omitted for subsequent assignment operations.

2.2.3 PROGRAM HEADER (PROG)

The PROG header informs TRIM III that the operations to follow are program operations as distinguished from control operations. The PROG header must precede the first statement of a program. The PROG header operation on paper tape must always be preceded by a carriage return (see paragraph 3). A program header has the following format:

```
   L          W          V0         V1
[Program] → [PROG]  •  [name]  •  [date]
[ name  ]
```

1) L - The label of the PROG header operation is optional; however, when present it is considered to be the name of the program.

2) W - The operator of this header operation is always PROG and must be present.

3) V - The V operands of this header operation normally take the form name and date as illustrated. The operands are optional and completely flexible in number and length within the maximum line length.

2.2.4 CORRECTION HEADER (CORREC)

The CORREC header informs TRIM III that the operations following are source language corrections to be integrated into the source language program under assembly. A maximum of 192 correction operations is permitted for any one assembly run. Three types of correction operations are provided by TRIM III:

1) Insertions or additions.
2) Replacements.
3) Deletions.

Although the correction feature is primarily intended for use with paper tape input mode, it may be used with any combination of input modes, the only restriction being that all corrections must be read in prior to read-in of the source program.

The format of correction operations is identical to that required by the TRIM corrector (refer to the TRIM corrector description contained in this manual). Figure III-C-5 shows a sample of correction coding which may be used with the CORREC leader in the TRIM III assembler.

Corrections are always made on the basis of the sequential line identifier associated with each source program statement. This sequential identifier appears on TRIM III outputs 2, 12, and 14. If assembly consists of multiple source programs, it must be remembered that the sequential identifiers are cumulative and correction is based upon these cumulative identifiers in any given assembly. If two or more correction operations bear the same integral and fractional identifier, the last one read will supersede the preceding one with the same identifier, permitting a programmer to correct a correction. Only the last such correction will count towards the 192 maximum.

2.2.5 DEBUG DECLARATIVE

TRIM III accepts the declarative operation:

L	W	N
[label] →	DEBUG →	

TITLE MANL (CORRECTIONS)
PAGE 1 of 1

UNIVAC
TAPE CORRECTION FORM

PROGRAMMER
PLT. EXT MS
DATE

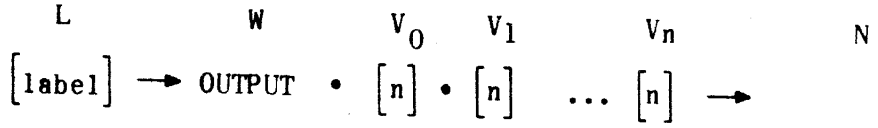
LABEL	OPERATOR	OPERANDS AND NOTES
MANL	HEADER TYPE CORREC	• W. C. Roos • 8Nov 1964
112 • 05		
MANL17	MOVE	• 10 • MANL8 • MANL99 → INSERT CORRECTION
47 • 0		
	ENTALK	• 501 → REPLACE CORRECTION
6 • 05		
	BUFIN	• CHANL • MAD • 100D • MANL
6 • 05		
	BUFIN	• CHAN • MAD • 100 • MANL80 → CORRECTS A CORRECTION
201 •		
	DELETE	• 18D → DELETES THIS AND NEXT 17
17 •		
	DELETE	• → DELETES THIS ONE ONLY
315 • 05		
MANL99	RESERV	• 8D ADDITION TO END OF PROGRAM
315 • 10		
MANLAU	0	•
315 • 15		
MANLAL	0	•
316 •		
MANLB	0	•
.. •		
		•
•		
		•

Figure III-C-5. Sample Correction Coding

The DEBUG operator informs TRIM III that generation is to be performed for debugging operations contained in the source program. If the DEBUG operator is absent, no generation will occur for such debugging operations. The DEBUG operation when used must be loaded prior to the first PROG header. It normally appears on the CONTR tape.

2.2.6 OUTPUT DECLARATIVE

TRIM III accepts the declarative operation:

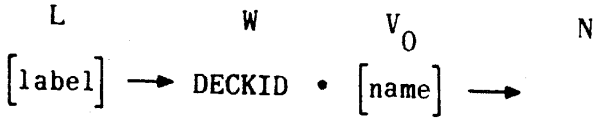


The OUTPUT operation permits the user to specify the assembler outputs he desires. The outputs are specified by number in the V_0 through V_n position. Up to eight outputs may be requested by the OUTPUT operation. Requests in excess of eight will be ignored and multiple OUTPUT statements are not permitted. An example of a legal OUTPUT operation is given below.

→ OUTPUT • 1 • 15 • 6 • 2 • 5 →

2.2.7 DECKID DECLARATIVE

TRIM III accepts the declarative operation:



The DECKID operation permits the user to specify card identification on printer or source card outputs he may select from TRIM III. From one to four alphanumeric characters may be specified in the V_0 position. These characters together with a 4-digit sequential octal number beginning with 0001, are added to each TRIM III statement that is also assigned a sequential line identifier. This card information will appear on the side-by-side printer listing output of the program (output 12) and the punched card output in source language (output 15). The new card identification and numbering preempts that which might be present if the input source program is on cards. Any number of DECKID statements may be inserted anywhere in the source program; however, each DECKID operation affects only those statements following that DECKID statement, and the card numbering will always begin with 0001.

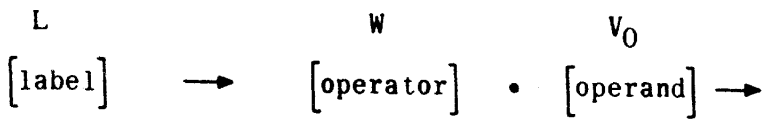
2.2.8 ENDATA DECLARATIVE

The ENDATA operation is used with card input to TRIM III. It informs the assembler of the end of a card deck. It does not mean the end of all input. The ENDATA operation does not cause any object language generation. It may have a label and notes. One blank card must follow each ENDATA card.

2.3 MONO-OPERATIONS

Mono or one-to-one operations consist of the mnemonic function codes in the instruction repertoire and symbolic addresses, absolute machine codes, or constants. Mono-operation statements may be in one of the following formats:

2.3.1 FORMAT A



L - The label is optional.

W - The operator is the f or fm portion of the operation statement and is the mnemonic representation of the desired function code of the computer instruction repertoire.

V_0- Represents the u or k portion of the statement and may be a tag, a tag ± an integer, or an integer only. Integers may be in octal or decimal representation. When decimal representation is used, the integer must be followed by the letter D. Incrementing or decrementing of integers is not permitted. If V_0 is absent, TRIM III generates zeros for the operand without error indications.

Examples:

→ ENTAL•CAT →	
→ STRADR•CAT+1 →	
→ CMAL•CAT-8D →	
→ ENTBK•28D →	
→ ENTALK•7776 →	Minus 1 to AL
→ STOP•DOG →	DOG defined by an ALLOC opn
→ SKPOIN•7 →	
→ CPAU →	Results in 506200
→ JP•LOK-10 →	LOK signifies this address
→ OUT•6 →	Output transfer channel 6
→ 0•CHEESE+1 →	Buffer terminal address
→ 0•CHEESE →	Buffer initial address
→ ENTAL →	Results in 120000
→ JP• →	Results in 340000

2.3.2 FORMAT B

TRIM III also accepts programs coded with absolute function codes and absolute or symbolic addressing. Normal instructions are represented by a 2-digit function code followed by a point separator and the desired u or mk operand. However, absolute instructions may also be represented by 6 consecutive digits without a point separator.

Examples:

→ 12•3505 →
→ 63•CAT+6 →
→ 50•1305 →
→ 50•6200 →
→ 506200 →

2.3.3 FORMAT C

Constants may be represented in a number of ways:

→7 →	Results in 000007
→7•0 →	Results in 700000
→77 →	Results in 000077
→77•0 →	Results in 770000
→777 →	Results in 000777
→77070 →	Results in 077070
→777•7 →	Illegal*
→777• →	Results in 000777*
→123456 →	Results in 123456

Two special mono-operations are available for the programmer's use; STOP and SKP. If either of these operators is used without a k operand, TRIM III will automatically generate an unconditional instruction of 50 56 40 or 50 50 40 respectively.

2.4 POLY-OPERATIONS

Frequently groups of instructions which perform a specific function appear iteratively in a program. A single poly-operation generates a unique sequence of instructions designed to perform some such specified function. This is the one-to-many relationship between instructions herein termed poly-coding; the parent instruction is termed a poly-operation. TRIM III provides for several poly-operations. In some cases TRIM III generates only a single instruction or,

*Whenever there is an expressed value following the point separator, only 1 or 2 digits are permitted in the operator position.

as in the case of REMARK and CALL operation, no instructions. It is permissible when coding a routine to intermix mono- and poly-operations in any desired order.

The CLEAR and MOVE poly-operations use the currently active B register and the MOVE poly-operation also uses AU in the generated coding. If the programmer does not wish the data in these registers to be destroyed, he must store and restore the data around a MOVE or CLEAR operation. The MOVE and CLEAR operations store and restore the programmer's special register setting. Since poly-operations generate more than one machine instruction, the tag LOK ± an integer must not be used for poly-operation coding.

2.4.1 RESERVE OPERATION (RESERV)

L	W	V_0
[label] ⟶	RESERV •	[Number of words] ⟶

The RESERV operation causes the desired number of sequential words to be reserved within a program. The operation generates the number of zero words* specified by the V_0 operand.

1) L - The label for this operation is optional.

2) W - RESERV must always be present.

3) V_0 - Specifies by an octal or decimal integer the number of zero words to be generated. V_0 may never equal zero.

Examples:

Assume CAT = 1000 and DOG = 2000

CAT ⟶ RESERV • 12 ⟶ Generates zeros at addresses 1000-1011

DOG ⟶ RESERV • 10D ⟶ Generates zeros at addresses 2000-2011

2.4.2 CLEAR OPERATION

L	W	V_0	V_1
[label] ⟶	CLEAR •	[Number of words] •	[starting address] ⟶

The CLEAR operation clears to zero those memory addresses specified in the operation.

1) L - The label for this operation is optional.

2) W - CLEAR must always be present.

*TRIM III outputs 2, 12, and 14, used primarily for hard-copy debugging and documentation, reflect only the first generated zero word of each RESERV operation. All other object language outputs contain the requested number of zero words.

3) V_0- Specifies by an octal or decimal integer the number of consecutive memory locations to clear. V_0 may never exceed 4000 octal or 2048D. A V_0 of zero is not permitted.

4) V_1- Specifies the first address of the area to be cleared. The address may be expressed as an absolute octal number or as a symbolic tag plus or minus an octal or decimal integer; that is, CAT-12D or CAT-11. All the words to be cleared must be wholly contained within one memory bank.

Examples of coding for CLEAR operations are given below.

[label] → CLEAR • 18D • FLIP+12D →

[label] → CLEAR • 22 • FLIP+14 →

[label] → CLEAR • 100D • FLAP-5 →

[label] → CLEAR • 4000 • 130000 →

Examples of CLEAR operations and the absolute coding generated by the assembler are given below.

Assume EXAM1 = 1000, EXAM2 = 1006, and CAT = 10123

Input Operation	Generated Coding
EXAM1 → CLEAR • 70 • 7000 →	36 0067 75 1005 50 7300 41 7000 73 1003 50 7300
EXAM2 → CLEAR • 21D • CAT →	36 0024 75 1013 50 7311 41 0123 73 1011 50 7300

A symbolic representation of the instructions generated is given below.

→ENTBK •[No. of locations -1]→Set B for No. of locations
→STRSR • LOK+4 →Store current SR
→ENTSR • [Bank No. of clear area] →Set SR to clear area
→CLB • [First location] →Clear word at first location + B
→BJP • LOK-1 → Decrement B and repeat loop
→ENTSR • 0 → Return to current bank when B is zero

2.4.3 MOVE OPERATION

L	W	V_0	V_1	V_2
[label] ⟶	MOVE •	[number of words] •	[from address] •	[to address] ⟶

1) L - The label for this operation is optional.

2) W - MOVE must always be present.

3) V_0- Specifies by an octal or decimal integer the number of sequential words to be moved. V_0 may never exceed 4000 octal or 2048D. A V_0 of zero is not permitted.

4) V_1- Specifies the first address of a block of data to be moved. It may be expressed as an absolute address in octal or as a symbolic tag plus or minus an octal or decimal integer. All the words to be moved must be wholly contained within one bank.

5) V_2- Specifies the first address to which the block of data is to be moved. It is expressed the same as the V_1 operand. All the destination addresses into which data are to be moved must be wholly contained within one bank.

Examples of coding for MOVE operations are given below.

[label] ⟶ MOVE • 78D • CAT • DOG-7 ⟶

[label] ⟶ MOVE • 10 • HORSE+10 • COW+8D ⟶

[label] ⟶ MOVE • 4000 • CAT • PIG ⟶

[label] ⟶ MOVE • 100D • 0 • 10000 ⟶

[label] ⟶ MOVE • 100D • 130000 • CAT ⟶

Examples of move operations and the absolute coding generated by the assembler are given below.

Assume EXAM3 = 1014, EXAM4 = 1024, and CAT = 1056

Input Operation	Generated Coding
EXAM3 ⟶ MOVE • 10 • CAT • 7000 ⟶	36 0007 75 1023 50 7300 11 1056 50 7300 47 7000 73 1016 50 7300

Input Operation	Generated Coding
EXAM4 → MOVE • 100 • 12000 • CAT-100 →	36 0077 75 1033 50 7311 11 2000 50 7310 47 0756 73 1026 50 7300

A symbolic representation of the instructions generated is given below.

→ENTBK • [No. of locations-1] →Set B for No. of words
→STRSR • LOK+6 → Store current SR
→ENTSR • [Bank No. of from address]→ Set SR to origin bank
→ENTAUB • [from address] → Get word at from address + B
→ENTSR • [Bank No. of to address] → Set SR to destination bank
→STRAUB • [To address] →Store word at to address + B
→BJP • LOK-4 → Decrement B and repeat loop
→ENTSR • 0 → Return to current bank when B is zero

2.4.4 I/O OPERATIONS

L	W	V_0	V_1	V_2	V_3
[label]	→ EXFCT •	[channel number]	• [AD, MAD, BK, MBK CAD, CMAD, CBK, CMBK]	• [number of buffer words]	• [buffer starting address] →
[label]	→ BUFIN •	[channel number]	• [AD, MAD, BK, MBK CAD, CMAD, CBK, CMBK]	• [number of buffer words]	• [buffer starting address] →
[label]	→ BUFOUT •	[channel number]	• [AD, MAD, BK, MBK CAD, CMAD, CBK, CMBK]	• [number of buffer words]	• [buffer starting address] →

1) L - Label for these operations is optional.

2) W - The operator must always be present.

3) V_0- Specifies the channel number expressed as an integer or a symbolic tag.

4) V_1- Specifies the buffer mode and must be present:

   a) AD - Advance without monitor.
   b) MAD - Advance with monitor.

c) BK - Back without monitor.
d) MBK - Back with monitor.
e) CAD - Advance without monitor - continuous data mode.
f) CMAD - Advance with monitor - continuous data mode.
g) CBK - Back without monitor - continuous data mode.
h) CMBK - Back with monitor - continuous data mode.

} For use with 1219 Input/Output Buffering Mode only

5) V_2 - Specifies as an octal or decimal integer the number of buffer words involved. Maximum of five digits.

6) V_3 - Specifies the address in memory at which buffering is to begin. V_3 may be expressed absolutely or as a symbolic tag plus or minus an octal or decimal integer.

Examples:

Assume CHAN = 07 and CAT = 10000

[label] → EXFCT • CHAN • AD • 1 • 130000 → Generates 501307
130000
130000

[label] → BUFIN • 6 • MAD • 10 • CAT → Generates 501106
010007
210000

[label] → BUFOUT • 3 • CBK • 10 • CAT+7 → Generates 501203
410000
410007

[label] → BUFOUT • 10D • MBK • 100D • CAT+99D → Generates 501212
010000
610143

2.4.5 LIBRARY CALL OPERATION

L	W	V_0	V_1	V_n
[label] →	CALL •	[n] •	[n] ···	[n] →

The CALL operation permits the programmer to specify by name (label of the PROG header) the subroutines he wishes the assembler to retrieve from the library of subroutines. A single CALL operation may name up to eight such subroutines. If the user requires more than eight subroutines, he may specify them with additional CALL operations. Subroutines retrieved from the library are automatically added to the end of the source program and assembled with it. The user has complete control of their address allocation if he wishes via ALLOC operations.

Whenever a CALL operation follows the CONTR header, TRIM III will honor the calls, but the CALL operation itself will not appear on a side-by-side output listing. Only those operations following a PROG header appear on such listings. If a subroutine retrieved from the library contains CALL operations, these calls will also be retrieved and added to the end of the composite program until the last CALL operation has been honored. A request for output No. 7 causes all library CALL operations to be ignored.

The CALL operation causes no object program generation.

Examples:

→ CALL•TYPT•FLP•SINE•TYPC →

→ CALL•PCHC →

2.4.6 REMARK OPERATION

L	W	V_0

[label] → REMARK • [desired statement] →

The REMARK operation causes no object program generation. It is simply an aid to the programmer in expanding normal program notes.

The REMARK statement may not exceed one line or one card in length.

2.4.7 DATA OPERATION

L	W	V_0

[label] → DATA • [integer, binary point specification] →

The DATA operation allows the programmer to specify a positive or negative data integer and its binary point position. The bits are numbered from right to left 0-17D. The binary point specification must be separated from its associated integer by a comma. The absence of a minus sign implies a positive integer. The label is optional.

Examples:

[label] → DATA • 24D, 9D → Generates 030000

or

[label] → DATA • 30, 11 → Generates 030000

The binary representation is:

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	**0**	**0**	0	0	0	0	0	0	0	0	0

[label] → DATA • 123,4 → Generates 002460

The binary representation is:

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	0	1	0	0	1	1	0	0	0	0

2.4.8 PUNCH CONTENTS OPERATION (PCHC)

L	W	V_0

[label] → PCHC • [information to be punched and/or typewriter commands] →

The PCHC operation results in generated coding which, when run on the computer with the PCHC* subroutine, causes the octal contents of A, AU, AL, B or any memory location to be punched on the high-speed paper tape punch. The words to be punched may be interspersed with the following typewriter control symbols to provide subsequent listing in the desired format.

Operand	Performance
• |CR| •	carriage return, line feed
• Δ • or • |SP| •	space

The vertical bars indicate the information enclosed is a special symbol directing the typewriter. Each CR and SP must begin and end with the vertical bar. Controls are separated from other operands by point separators.

1) L - The label is optional.

2) W - The operator PCHC must be present.

3) V_0- Specifies the operands in the order in which they are to be punched. Except for the typewriter commands, all operands imply their contents are to be punched. Such operands may be A, AU, AL, active B, a tag or a tag ± an absolute value, or an absolute address.

Examples:

CAT → PCHC • A • Δ • 7070 • Δ • DOG-11D • |CR| →

→ PCHC • DOG • Δ • B • AL →

→ PCHC • |CR| • AU • |SP| • AL →

→ PCHC • DOG+10 • |SP| • CAT →

*See paragraph 3.4 6).

2.4.9 PUNCH TEXT OPERATION (PCHT)

L	W	V_0
[label]	→ PCHT •	[text and/or typewriter commands] →

The PCHT operation results in generated coding which, when run on the computer with the PCHT* subroutine, causes the text and/or typewriter commands in the V_0 operand position tc be punched by the high-speed paper tape punch. The text may be interspersed with the following typewriter control symbols as desired; each CR and SP must be set off between two vertical bars.

Operand	Performance
|CR|	carriage return, line feed
Δ or |SP|	space

1) L - The label is optional.

2) W - The operator PCHT must be present.

3) V_0- Is the text to be punched interspersed with typewriter commands desired by the programmer. If the text is too long for one PCHT operation, the programmer can write successive operations.

Examples:

CAT → PCHT • PROFIT Δ AND Δ LOSS Δ FOR →

→ PCHT • JULY Δ 10, Δ 1967 |CR| →

NOTE: Point separators are not required within V_0; they will be punched if present.

2.4.10 TYPE CONTENTS OPERATION (TYPC)

L	W	V_0
[label]	→ TYPC •	[information to be typed and/or typewriter commands] →

The TYPC operation results in generated coding which, when run on the computer with the TYPC* subroutine, causes the octal contents of A, AU, AL, current B, or any memory location to be typed on the typewriter. The words to be typed may be interspersed with the following typewriter commands.

*See paragraph 3.4 6).

Operand	Performance
• |CR| •	carriage return, line feed
• Δ • or • |SP| •	space (may be used for formatting)

The vertical bars indicate the information enclosed is a special symbol directing the typewriter. Each, CR or SP must begin and end with a vertical bar.

1) L - The label is optional.

2) W - The operator TYPC must be present.

3) V_0- Specifies the operands in the order in which they are to be typed. Except for the typewriter commands, all operands imply their contents are to be typed. Such operands may be A, AU, AL, active B, a tag or a tag ± an absolute value, or an absolute address.

Examples:

CAT ⟶ TYPC • A • Δ • Δ • 7070 • Δ • Δ • DOG-11D • |CR| ⟶

⟶ TYPC • AU • Δ • AL • |SP| • B • HORSE ⟶

2.4.11 TYPE TEXT OPERATION (TYPT)

L	W	V_0
[label] ⟶	TYPT •	[text and/or typewriter commands] ⟶

The TYPT operation results in generated coding which, when run on the computer with the TYPT* subroutine, causes the text and/or commands in the V_0 operand position to be typed by the typewriter. The text may be interspersed with the following typewriter commands:

Operand	Performance
|CR|	carriage return, line feed
Δ or |SP|	space (may be used for formatting)

1) L - The label is optional.

2) W - The operator TYPT must be present.

3) V_0- Is the text to be typed interspersed with typewriter commands. If the text is too long for one TYPT operation, the programmer may use successive operations to complete the text.

*See paragraph 3.4 6).

Examples:

CAT ⟶ TYPT • PROFIT |SP| AND Δ LOSS Δ FOR ⟶

⟶ TYPT • JULY Δ 10, Δ 1967 |CR| ⟶

NOTE: Point separators are not required within V_O; they will be typed if present.

2.4.12 DOUBLE SET OPERATION

L	L
[label] ⟶	DBLSET ⟶

The DBLSET operation insures that the Y of a double add or subtract instruction is located at an even address. The DBLSET operation is normally followed by a Y constant. TRIM III examines the address to which the following constant (or instruction) would normally be assigned. If the address is odd, a word of zeros is first generated to insure that the constant (or instruction) will be assigned to an even address. If the address is even, no generation results.

2.4.13 SETSR OPERATION

L	W	V_O
[label] ⟶	SETSR •	[alphanumeric tag] ⟶

The SETSR operation enables the programmer to place responsibility for setting k of an ENTSR instruction upon TRIM III. Based upon an ALLOC operation or the assembled address of the referenced tag, TRIM III generates an ENTSR instruction (5073 k) with the proper k value for each SETSR operation.

1) L - Label is optional.

2) W - SETSR must be present.

3) V_O- Must be an alphanumeric tag corresponding to a program label or an allocated value. The tag may not be incremented or decremented.

Examples:

Assume CAT is a label at 36421 and DOG is a label at 170460 and COW is allocated to 010000, then:

⟶ SETSR • CAT ⟶ Generates 507313

⟶ SETSR • DOG ⟶ Generates 507337

⟶ SETSR • COW ⟶ Generates 507310

2.5 DEBUGGING OPERATIONS

TRIM III provides two debugging operations for punching a paper tape output of either the contents of registers AU, AL, and current B, or the contents of specified sequential memory locations. TRIM III recognizes these operations only if a DEBUG declarative operation is read prior to the first PROG header operation. When recognized, these operations generate a set of three or five instructions in the object program which, when run on the computer with the DEBUG* subroutine, produce the desired dump. Each set of instructions is assigned a sequential identifying number which appears with each punched output, thereby enabling programmer recognition of repeated times through given coding paths. The debugging operations take the following form.

L	W					
[label]	→ DUMPR →					
L	W	V_0		V_1		N
[label]	→ DUMPM •	[number of words to dump]	•	[address of first word to dump]	→	

1) L - Label is optional.

2) W - DUMPR or DUMPM must always be present.

3) V_0- Applicable to the DUMPM operation only. Specifies the total number of memory locations to be dumped. The number may be expressed in octal or in decimal followed by the letter D.

4) V_1- Applicable to the DUMPM operation only. Expresses the address of the first word to be dumped. It may be expressed as an integer or a tag plus or minus an integer.

Examples of coding for DUMPR or DUMPM operations are given below.

[label] → DUMPR →

[label] → DUMPM • 12 • 10000 →

[label] → DUMPM • 10D • 10000 →

[label] → DUMPM • 10D • CAT+28D →

[label] → DUMPM • 12 • CAT-15 →

[label] → DUMPM • 64D • CAT →

*See paragraph 3.4 6).

Examples of the DUMPR and DUMPM operations and the coding generated by the assembler are given below.

Examples:

Assume EXAM5 = 1000, EXAM6 = 1050, and DEBUG = 30000.

Input Operation	Generated Coding
EXAM5 ⟶ DUMPR ⟶	301001 030000 000001
EXAM6 ⟶ DUMPM • 5 • 10000 ⟶	301051 030000 400002 000005 010000

A symbolic representation of the instructions generated is given below. The first three instructions apply to both DUMPR and DUMPM. The last two instructions apply to DUMPM only.

⟶ IRJP • DEBUG ⟶ Indirect return jump to DEBUG
⟶ 0 • DEBUG ⟶ Address of DEBUG
⟶ X • [Y] X = 0 for DUMPR, 4 for DUMPM
Y = No. of DUMPR or DUMPM operation in this program
⟶ [No. of words] ⟶ No. of words to be dumped
⟶ [First address] ⟶ Address of first word to be dumped

Both DUMPR and DUMPM operations preserve existing values in AU, AL, and the current B register.

TRIM III also provides two additional debugging operations for programmer use: DSTOP and DTYPT.

L	W
[label] ⟶	DSTOP ⟶

The DSTOP operation permits the programmer to intersperse strategic debugging stops within his program. If the DEBUG operator is read in the assembly prior to the PROG header, the DSTOP will generate an unconditional stop (505640); otherwise, TRIM III will ignore the operation.

L	W	V_0
[label] ⟶	DTYPT •	[text and/or typewriter commands] ⟶

The DTYPT operation performs the same function as the TYPT operation. If the DEBUG operator is read in the assembly prior to the PROG header, TRIM III will perform the generation; otherwise, the operation will be ignored.

2.6 TRIM III OUTPUTS

TRIM III provides 13 different outputs of the assembled and/or source program. The user selects his outputs in accordance with his needs and the available peripheral devices.

The available outputs are listed under the output device on which they are produced.

Monitoring typewriter:

No. 1 - Program summary consisting of the number of memory locations used and inclusive addresses.

Paper tape punch: Except for outputs 6 and 11, all paper tapes are loadable via the utility packages. Outputs 6 and 11 may be used as input to TRIM III.

No. 2 - Absolute assembled program, sequential line identifier, source program, and assembly error alarms when applicable. This is a side-by-side listing in source code preceded by a program summary consisting of the number of memory locations used and inclusive addresses.

No. 3 - Absolute assembled program in source code, consisting of a carriage return, 88, carriage return, addresses and instructions, a carriage return, double period and checksum.

No. 4 - Absolute assembled program in bioctal format, consisting of a 76 code, inclusive area addresses followed by the instructions only, and a checksum.

No. 5 - Relocatable assembled program in bioctal format starts with a 75 code followed by the assembled program relative to base 00000, and terminates with a checksum. The output tape may be loaded starting at any desired memory location.

No. 6 - Allocation output in source code consisting of an ALLOC header, followed by all program tags and labels and addresses in allocation format.

No. 11- The source program only, produced in source code.

High-speed printer:

No. 12- Absolute assembled program, sequential line identifier, deck and card number if applicable, source program, and assembly error alarms when applicable. This is a side-by-side listing suitable for hard-copy editing and documentation.

No. 14 - This is the same as output No. 12 except that there is no card information and page size is assumed to be 11 inches wide by 8½ inches long.

Card processor:

No. 13 - Relocatable assembled program on Hollerith-coded 80-column cards. The first card contains only the base load address. Subsequent cards contain up to 8 computer words, a cumulative checksum, and a card sequence number.

No. 15 - Source program only, on Hollerith-coded 80-column cards. Each card contains one TRIM III statement as well as any card deck identification and sequence number.

Magnetic tape unit:

Relocatable object program. Assembled object program in assemble table 3 format.

No. 10 - During assembly TRIM III automatically produces this output on the magnetic scratch tape. The tape data can be loaded into the computer memory absolutely or relocated to any specified base address by the utility packages.

No. 16 - Source program on magnetic tape. This output does not include declarative operations such as ALLOC, OUTPUT, or DECKID. Output No. 15 may be used as input to TRIM III.

Miscellaneous:

No. 7 - Output No. 7 is not itself an output, but does affect all other requested outputs, since it causes TRIM III to ignore all library CALL operations of the input program.

3. PROGRAMMING PROCEDURES

3.1 PAPER TAPE INPUT FORMAT

Two versions of TRIM III are available; one version accepts a source program paper tape prepared in field data code, the other version accepts a source program paper tape prepared in ASCII code (refer to Appendix A, Tables A-2 and A-3).

Each source tape must begin with a carriage return and terminate with a carriage return and two periods.

3.1.1 KEYBOARD CORRECTION METHODS

Typing-error correction procedures have been incorporated in both versions of the TRIM I, TRIM II, and TRIM III assemblers, and the TRIM corrector for deleting immediate keyboard errors that might be made in the preparation of input tapes for these programs on the UNIVAC 1232 and 1532 I/O consoles. These procedures are described under TRIM I, paragraph 8.

3.2 80-COLUMN CARD INPUT FORMAT

For those installations whose peripheral equipment configuration includes an on-line card reader, TRIM III accepts source programs prepared in Hollerith code on standard 80-column cards as well as source programs prepared on paper tape. The two input types may be intermixed.

Basically the coding format is similar for either card or paper tape unit. Interpretation of coding separator symbols for card input is given in Table III-C-2.

TABLE III-C-2. CODING SYMBOLS FOR CARD INPUT

Symbol	Key	Rows Punched
→ (start statement)	SKIP	(none)
→ (start notes)	≡ ≡ ≡	4,8 4,8 4,8
↵ (carriage return)	REL	(none)
| (vertical line)	$ *	11, 3, 8
• (point separator)	$ *	11, 4, 8
. (period)	. (	12, 3, 8
, (comma)	, ;	0, 3, 8
+ (plus)	+ P	12
- (minus)	SKIP	11

The straight coding arrow is interpreted according to its format position; it represents a SKIP key at the beginning of a statement and three dashes at the end of a statement. The point separator is represented by the * key in all card input.

Card control: The ENDATA operation card followed by one blank card denotes the end of a card input deck.

Coding format: The examples on the top of page III-C-30 illustrate the basic coding format for card input: (Also see Figure III-C-6).

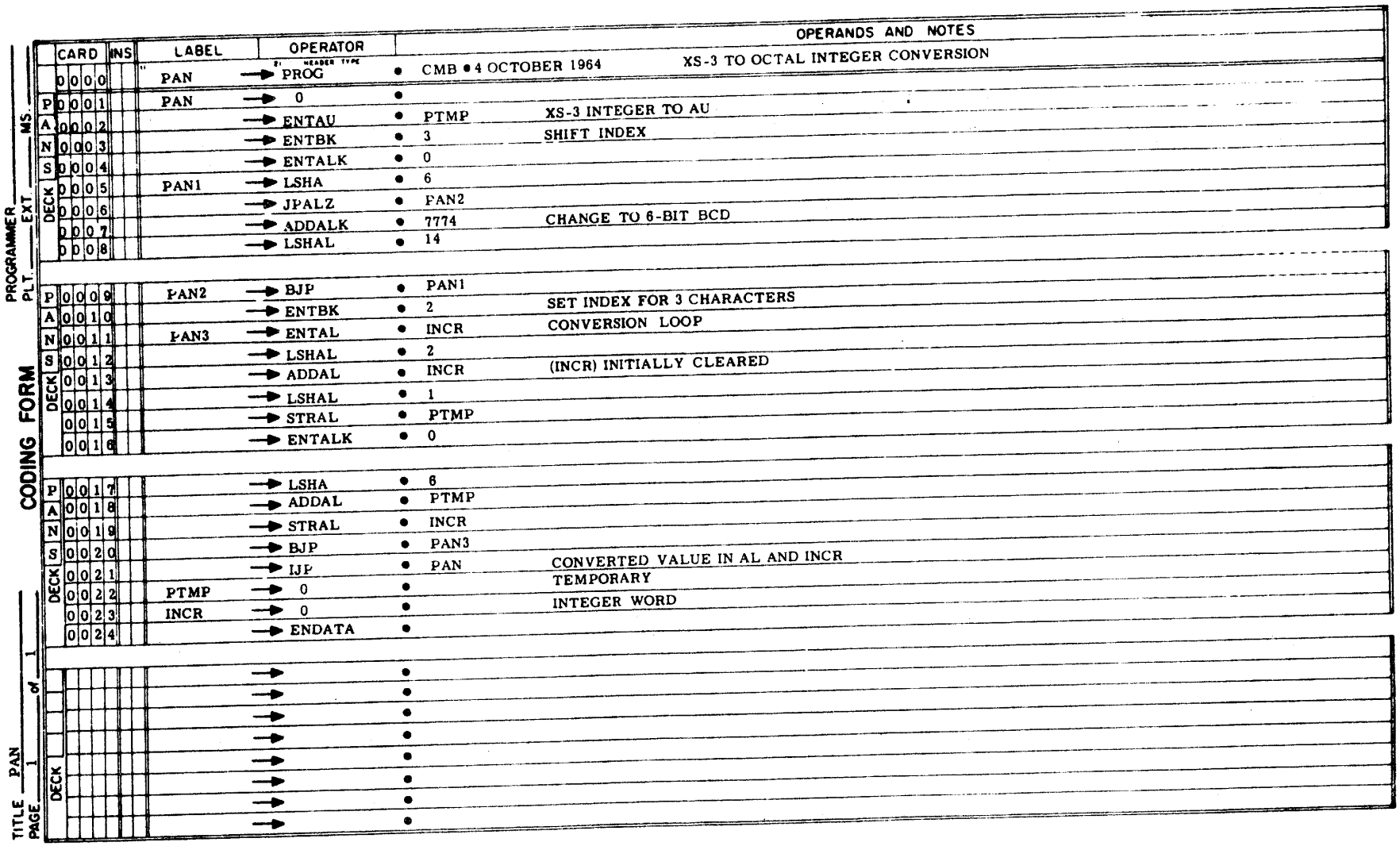
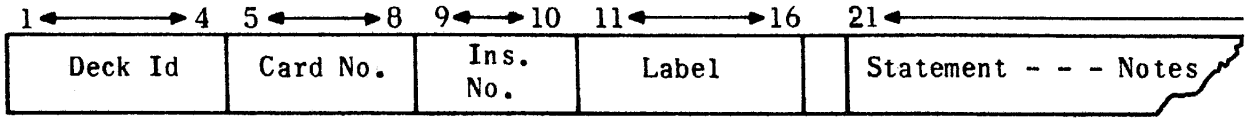


Figure III-C-6. Typical Coded Programmer Card Input

```
DECK      CARD      INS
 ID       NO.       NO.      L          W          V          N

B017      0005                CAT4 ──►ENTAUB  •  DOG5─►MASK FOR SEARCH
B017      0005      05              ──►CMSK   •  CAT10
B017      0006                      ──►JPNOT• LOK-3 ──► LOOK AGAIN
```

Card format: Card format uses card columns 1-4 for deck identifier, 5-8 for card number, 9-10 for card insert number, 11-16 for the label, and 21-80 for statement and notes.



Three dashes (---), punched code 4,8, always follow the statement whether or not there are notes. The REL (release) key terminates the card. TRIM III makes no provision for the statement and notes to overflow one card. Any attempt to continue notes on a second card results in improper generation for that card.

The column-skip feature on the key punch provides a convenient means to bypass unused columns reserved for the label. The keypunch operator begins a label with column 11 and skips any unused columns between the end of the label and column 21. If no label is present, the operator depresses the SKIP key and the card is automatically positioned at column 21. The statement always begins at column 21. Figure III-C-7 shows a typical input operation in punched card format.

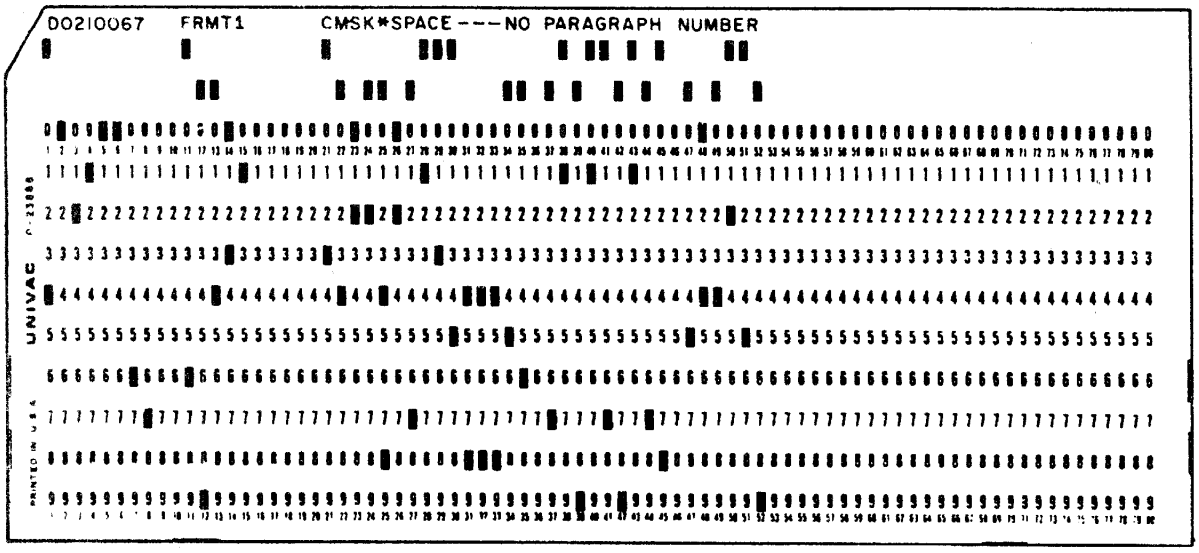


Figure III-C-7. Typical Punched Card Input Operation

3.3 SOURCE PROGRAM CORRECTIONS

When so directed by a CORREC header operation, TRIM III will perform source program corrections in conjunction with an assembly run. Outputs from the assembly will include the requested corrections. The following rules govern the use of the correction capability:

1) A maximum of 192 corrections per assembly is allowed.

2) Corrections must follow the CORREC header.

3) All corrections must be loaded prior to the loading of the source program to be corrected.

4) Corrections need not be in any special order. TRIM III will sort the correction items prior to merging them with the source program.

5) If two or more correction operations bear the same integral and fractional value, the last such operation overrides. This permits programmers to correct an erroneous correction.

6) Corrections are based on the assembler-assigned sequential line identifier for each source statement (see paragraph 2.2.4).

7) The integral and fractional portions of the correction identifier must be expressed in octal notation only. The integer is limited to a maximum of 6 digits; the fraction (which indicates insertion or addition) is limited to 3 digits. The fraction is a straight binary magnitude, (for example, the correction identifiers 12•2, 12•20, and 12•200 all have the same value).

8) Corrections may be prepared on punched cards or punched paper tape.

3.3.1 PAPER TAPE CORRECTION FORMAT

The format of correction operations prepared on paper tape is identical to that required by the TRIM corrector (refer to input formats in the TRIM corrector description contained in this manual) with the following exceptions:

1) Corrections must follow a CORREC header.

2) A maximum of six integral digits is permitted.

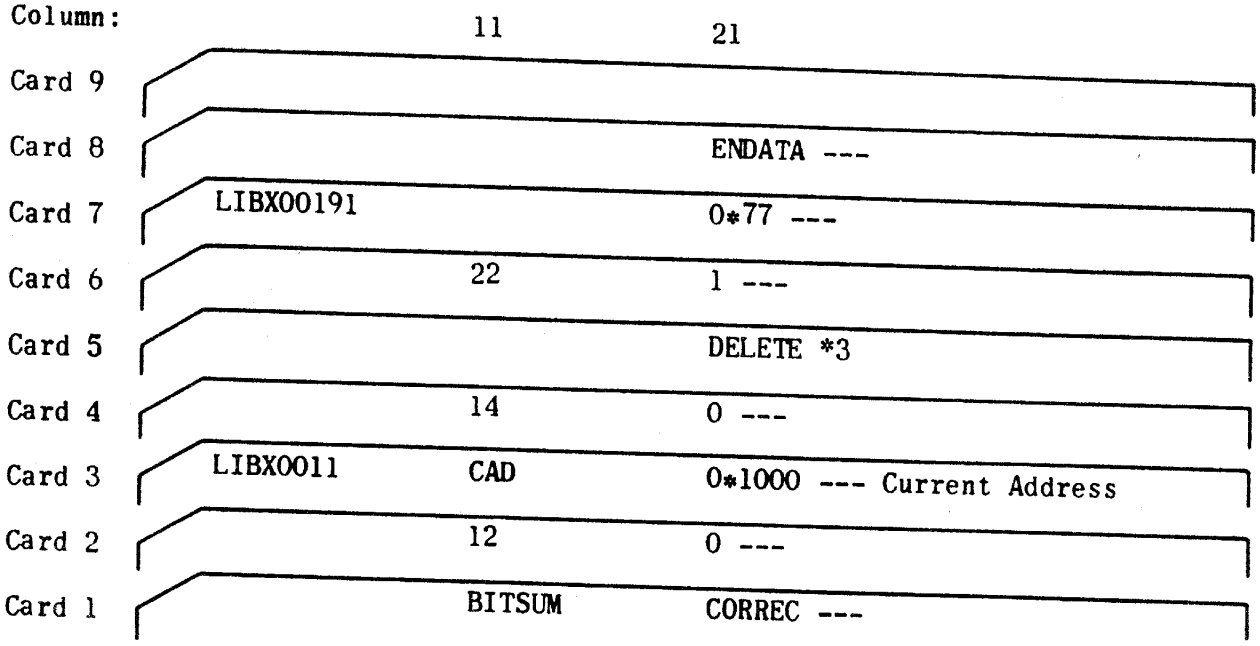
3.3.2 CARD CORRECTION FORMAT

Special formatting rules apply to correction operations prepared on 80-column punched cards. Except for the CORREC header each correction action requires two cards. The first card contains the integer and fraction of the sequential identifier while the second card contains the correction operation itself. If the correction operation is an insertion or replacement, the first ten columns of this second card may also contain card identifications which are included in the outputs 12 and 15 of the assembled corrected program, unless a DECKID operator is used.

On the first card, the integral portion of the sequential identifier must begin in column 11. A point separator (asterisk) must not be used. The fractional portion, if any, must begin in column 21 and must be terminated with the conventional three dashes. If there is no fraction, one zero code followed by the three dashes must still be punched beginning at column 21.

An ENDATA card followed by a blank card must always follow the last correction card even if other cards are to follow in the assembly.

The following example of card correction format makes changes to the program illustrated in Figure III-C-8.



3.4 GROUND RULES

Regardless of the input format, there are certain conventions which the programmer must bear in mind when coding for TRIM III.

1) No label may exceed six characters. The label must not begin with a number, the letter O, nor may it consist only of letters LOK. The label may never contain a +, -, comma, or point separator code.

2) The maximum size program which TRIM III can assemble is limited only by the number of memory locations above address 13000_8 used for label/tag storage (3 words per label or tag).

3) Each break in addressing sequence constitutes a program area. A total of 64 such areas is permitted.

4) TRIM I operators SETADR and EQUALS are ignored by TRIM III. The ALLOC operation replaces these two functions.

OUTPUT 12

```
MEM. STRG. USED  11061
00240 THRU 00404
01000 THRU 07743
20000 THRU 21747

LOK     INSTR      LIID DECK CARD  LO
                   0    LIBX 0001  BITSUM   PROG*JRS*6NOV64      FLEX
00240   34 0364    1    LIBX 0002  BITSUM   JP*SEOK              LIBBLD BOTTSTRAP LOAD
00241   76 0355    2    LIBX 0003  UPAKX    RJP*ERP
00242   40 0247    3    LIBX 0004  UPAK     CL*CHECK             CLEAR ACCUM CKSUM
00243   42 0254    4    LIBX 0005           STRB*BNASTY          SAVE B
00244   76 0345    5    LIBX 0006  UPAK1    RJP*RF               READ FRAME
00245   61 0244    6    LIBX 0007           JPALZ*UPAK1          IGNORE LEADER
00246   34 0270    7    LIBX 0008           JP*BILD

00247   00 0000    10   LIBX 0009  CHECK    0*                   ACCUM CKSUM
00250   00 0000    11   LIBX 0010  SUM      0*                   ZERO CONST
00251   00 0000    12   LIBX 0011  CAD      0*                   CURRENT ADDR
00252   00 0000    13   LIBX 0012  FAD      0*                   FINAL ADDR
00253   00 0076    14   LIBX 0013  BIO      76*                  BIOCTAL CODE
00254   00 0000    15   LIBX 0014  BNASTY   0*                   BKEEPER
00255   00 0000    16   LIBX 0015  BUWD     0*                   DATA I/O BUFFER
00256   00 0000    17   LIBX 0016           0*

00257   00 0000    20   LIBX 0017           0*
00260   00 0000    21   LIBX 0018           0*
00261   00 0000    22   LIBX 0019           0*
00262   00 0000    23   LIBX 0020           0*
00263   00 0000    24   LIBX 0021           0*
00264   00 0000    25   LIBX 0022           0*
00265   00 0000    26   LIBX 0023           0*
00266   00 0000    27   LIBX 0024           0*

00267   00 0000    30   LIBX 0025           0*
00270   40 0247    31   LIBX 0026  BILD     CL*CHECK
00271   10 0250    32   LIBX 0027           ENTAU*SUM            CL AU
00272   76 0335    33   LIBX 0028           RJP*BILD7            6 DIGITS OF ADDRS
00273   50 4717    34   LIBX 0029           LSHA*17
00274   46 0251    35   LIBX 0030           STRAU*CAD
00275   10 0250    36   LIBX 0031           ENTAU*SUM
00276   50 4703    37   LIBX 0032           LSHA*3

00277   70 0001    40   LIBX 0033           ENTALK*1
00300   74 0336    41   LIBX 0034           STRADR*BILD7+1
00301   76 0335    42   LIBX 0035           RJP*BILD7            GET REST OF ADDRS
00302   44 0252    43   LIBX 0036           STRAL*FAD
00303   70 0002    44   LIBX 0037           ENTALK*2
00304   74 0336    45   LIBX 0038           STRADR*BILD7+1
00305   76 0335    46   LIBX 0039  BILD2    RJP*BILD7
00306   32 0251    47   LIBX 0040           ENTB*CAD
```

Figure III-C-8. TRIM III Output 12 from Card Input

5) When specifying a decimal integer, the letter D occupies one digit position; therefore, the maximum decimal integer that can be expressed is 99999D.

6) Assembler support subroutines TYPT, TYPC, PCHT, PCHC, and the debugging package, DEBUG, are included in the TRIM III library of subroutines. The programmer uses a CALL operation to retrieve them from the library. The programmer may allocate these subroutines through normal ALLOC operations. If he does not allocate them, TRIM III will assign them sequential addresses immediately following the principal program. If these subroutines are not assembled with the principal program and the programmer has not provided for their allocation, TRIM III will arbitrarily assign all references to them to the following fixed addresses:

TYPT	17000
TYPC	17160
PCHT	16400
PCHC	16560
DEBUG	17470

Each of these five subroutines uses the tag CHAN for all I/O instructions. It is the programmer's responsibility to provide an ALLOC operation equating CHAN to the appropriate I/O channel.

7) If a program contains ADDA, ADDAB, SUBA, or SUBAB instructions, regardless of whether or not a DBLSET operation was used, the following restrictions shall apply to loading a TRIM III output No. 5, 10, or 13 into computer memory:

a) If the program was assembled starting at an even address, it must be loaded starting at an even address.

b) If the program was assembled starting at an odd address, it must be loaded starting at an odd address.

8) TRIM III informs the user of a duplicate label via a typeout on the on-line typewriter. The typeout includes the sequential line identifier, the warning, DUP LBL, and the label name. Except for the warning typeout TRIM III will normally ignore duplicate labels equating all references to the address of the first such label. However, if the user has allocated a label which is in fact a duplicate, that allocation is lost with unpredictable results in address assignment.

9) Any program to be assembled by TRIM III must be assembled for only one 32K segment of memory, either 000000 through 077777 or 100000 through 177777. This means that the assembled program must reside in one 32K segment or the other, but not both. This does not preclude inter-segment references which would be implemented exactly as for inter-bank references. The TRIM assemblers do not provide any alarm indications for this condition.

4. TRIM III LOADING AND OPERATING PROCEDURES

4.1 BASIC INFORMATION

TRIM III is a magnetic-tape-stored assembly system which accepts source programs written with absolute or mnemonic function codes and symbolic addressing and produces assembled output programs suitable for loading into the computer and/or hard copy editing and documentation. TRIM III has been designed to fit the channel and equipment configuration of the center in which it is used. The assembler provides the user with a simple means for selecting three optional modes of input, and each mode is represented by a number code.

Input Mode	Number Code
Cards	000001
Paper Tape	000002
Magnetic Tape	000003

Each TRIM III has one normal mode built into it (the one prevailing at the center where it is used). If only the normal mode is required, the user need not concern himself with the modes at all. However, if different input modes are to be used (for example, card and paper tape in combination) TRIM III users must familiarize themselves with the input mode codes and their use.

Prior to loading and operating TRIM III, the computer and the I/O equipment (magnetic tape unit, I/O console or card processor) must be placed in the operational state with all switches in the normal operating position.

4.2 LOADING TRIM III

1) For installations possessing a magnetic tape wired bootstrap:

   a) Mount the TRIM III assembler tape on magnetic tape cabinet 1, transport 1, and set the corresponding write enable button on the magnetic tape control panel.

   b) At the computer control panel, press MASTER CLEAR, LOAD, and START. The TRIM III executive will be loaded into memory and the computer will stop with P = 01404.

2) For installations possessing a paper tape wired bootstrap:

   a) Mount the TRIM III assembler tape on magnetic tape cabinet 1, transport 1, and set the corresponding write enable button on the magnetic tape control panel.

   b) Mount the TRIM III paper tape loader in the paper tape reader.

   c) At the computer control panel, press MASTER CLEAR, LOAD, and START. The TRIM III executive will be loaded into memory and the computer will stop with P = 01404.

4.3 INITIALIZING TRIM III

When the computer stops with P = 01404, it is necessary to identify the magnetic tape configuration to be used for assembly. This identification need be made only once for all subsequent assemblies unless it is necessary to change the configuration. TRIM III expects tape information in the format 00XXCT where XX is the channel number (bits 6 through 11), C is the tape cabinet number (bits 3 through 5), and T is the transport number (bits 0 through 2). Thus, 001512 represents channel 15, cabinet 1, transport 2, and 000112 represents channel 1, cabinet 1, transport 2.

The procedures required to initialize TRIM III are as follows:

1) Set the AU register to the number (00XXCT) which identifies the location of the assembler tape.

2) Set the AL register to the number (00XXCT) which identifies the transport to be used as the magnetic scratch tape.

3) Start the computer. The computer stops with AU and AL cleared.

4) If only two transports are to be used in the assembly, start the computer. When the computer stops with P set to 01400, TRIM III is ready for use (refer to paragraph 4.4).

5) If more than two transports are to be used in the assembly, perform the procedures below.

   a) If source input is to be from magnetic tape, set the AU register to the number (00XXCT) which identifies the transport to be used for source input.

   b) If a magnetic tape output other than output No. 10 is to be requested, set the AL register to the number (00XXCT) which identifies the transport on which the output is to be produced. Output No. 10 is always produced on the scratch tape.

   c) Start the computer. When the computer stops with P set the 01400, TRIM III is ready for use (refer to paragraph 4.4).

If, at any time during assembly, the user wishes to change the magnetic tape configuration, he may do so by setting P to 01404 and repeating the initialization procedures.

4.4 USING TRIM III

1) For paper tape input, mount the source tape in paper tape reader.

2) For card input from the UNIVAC 1004 Card Processor, initialize the UNIVAC 1004 Card Processor as follows:

   a) Mount the source card deck in card reader input hopper.

   b) Press CLEAR, START, FEED, and RUN.

3) For magnetic tape input, mount the input tape as follows:

   a) If two tape transports are being used, mount the input tape on the scratch tape transport.

   b) If three or four transports are being used mount the input tape on the transport designated for the input tape. Since this transport is used only for magnetic tape input, no warning typeout occurs.

4) Master clear.

5) Set P = 01400.

6) Set PROGRAM SKIP key 1.

7) For error typeout suppression, set PROGRAM SKIP key 3.

8) If source input is other than the normal mode, set AL to the proper code. (For normal mode, AL remains equal to zero.)

9) Start the computer.

10) TRIM III prepares to read input. If further operator action is necessary to continue assembly of the input currently being read, the computer stops after an operator instruction typeout occurs on the on-line typewriter. Refer to paragraph 4.5 to determine the operator action required.

11) When the current read-in is complete, the computer stops with AL equal to zero. At this time the operator must perform one of the following:

   a) If additional input is required, mount the source input on the input device as directed in 1), 2), or 3) above and restart the procedure at step 8) above.

   b) If no additional input is required, release PROGRAM SKIP key 1 and start the computer. TRIM III begins assembling the program. If further operator action is necessary, the computer stops after an operator instruction typeout (refer to paragraph 4.5).

12) If for any reason the operator wishes to abort an output during processing, he must perform the following procedures:

   a) Stop the computer.

   b) Master clear the computer.

   c) Set the P register to 01401.

   d) Start the computer. TRIM III begins processing the next requested output. If all outputs have been processed, the typeout SELECT OUTPUTS IN A will occur.

NOTE: Since TRIM III includes correction features, it is possible to correct a source program and assemble it for the desired new outputs in the same assembly run. Although this feature is intended primarily for the paper tape input mode, it may be used with any combination of input modes. Correction tapes or cards must be read in before the source program(s). If the assembly consists of multiple source programs, it must be remembered that the sequential line identifiers are cumulative and the corrections are based upon these identifiers in any given assembly. The input of the corrections and source program proceeds as in a normal assembly run. After the corrections have been read in, the source program(s) must follow with PROGRAM SKIP key 1 still set. Any outputs selected will contain the requested corrections including source program outputs 15 (cards), 11 (paper tape), and 16 (magnetic tape).

4.5 OPERATOR INSTRUCTION TYPEOUTS

TRIM III contains limited error detection capability. The majority of programmer errors are handled internally. However, it is desirable when practicable to permit the user to take corrective action during the assembly process in order to achieve an accurate assembly. The headings of the subparagraphs which follow are instruction typeouts that may occur during assembly. Information given in each subparagraph directs operator actions required by the typeout. When PROGRAM SKIP key 3 is set, certain typeouts are suppressed.

4.5.1 SET KEY 1

This typeout will occur if the user has not set PROGRAM SKIP key 1 at the start of the assembly process. To correct, set PROGRAM SKIP key 1 and start again. PROGRAM SKIP key 3 has no effect on this error typeout.

4.5.2 IDENT. MTUS IN A

This typeout occurs when no magnetic tape configuration identification has been made prior to the first assembly. To correct this condition, perform the following steps:

1) Identify magnetic tape units exactly as outlined in paragraph 4.3 of this section.

2) Start the computer. PROGRAM SKIP key 3 has no effect on this error typeout.

4.5.3 IDENTIFY TAPE JOB IN AL

This typeout occurs when the magnetic tape input to be read is in the format produced by the CART (card-to-tape) program. Since this format may contain more than one source program on the same tape, the operator must identify the program to be read from tape by performing the following actions:

1) Set the AL register to the number which identifies the position of the program on the tape (1 for the first program, 2 for the second program, and so forth).

2) Start the computer. TRIM III directs the tape unit to pass tape until the selected program is reached and then begins reading the input program. If two or more programs on the same tape are to be assembled together, these procedures are repeated for each program.

4.5.4 REMOVE INPUT TAPE TO SAVE

This typeout occurs when input has been read from the magnetic tape transport identified as the scratch transport. The output No. 10 will also be written on this transport during the assembly process. Therefore, if the input tape is to be saved, the operator must change the tape on the scratch transport before the output No. 10 is produced. This typeout occurs each time a magnetic tape read-in is completed; therefore, if more than one input program is read from the same magnetic tape, the operator must remove the tape only after the last input has been read. After the last magnetic tape input has been read and the tape has been removed, the operator must mount a scratch tape on the scratch tape transport and proceed with the assembly at step 11) of paragraph 4.4.

4.5.5 SELECT OUTPUTS IN A

This typeout may occur twice during an assembly. If it occurs before any outputs have been produced, it indicates that the programmer has neglected to select outputs via a programmed output operation. To correct this condition, perform the following steps:

1) Set AU_{5-0} and AL_{5-0} to desired output numbers.

2) Start the computer.

   a) The computer stores the outputs and stops. Repeat steps a) and b) until all desired outputs (not more than eight) have been selected. When the procedure is repeated with either AU or AL equal to zero, TRIM III assumes all selections have been made and proceeds. PROGRAM SKIP key 3 has no effect on this typeout.

If this typeout occurs after at least one output has been produced, it indicates that the assembly is complete. If another program is to be assembled at this time, the operator may elect to stack the output No. 10 for the next program behind the output No. 10 for the previously assembled program on the scratch tape. To select this option, perform the following steps:

1) Set PROGRAM SKIP key 2.

2) Start the computer. After TRIM III performs an index table adjustment, the computer stops with P set to 01400.

3) Release PROGRAM SKIP key 2, and begin assembly of the next program.

4.5.6 IF NECESSARY CHANGE SCRATCH TAPES FOR THIS OUTPUT

This typeout occurs when a magnetic tape output, other than output No. 10, has been requested but no tape transport has been identified for magnetic tape output. The typeout indicates that TRIM III is ready to write the magnetic tape output on the scratch tape and destroy the output No. 10 in the process.

If the operator does not wish to save the output No. 10, he may start the computer to continue assembly. If the operator does wish to save the output No. 10, he must change the tape on the scratch tape transport before starting the computer to continue assembly.

4.5.7 MTU ERROR CTXX IMPR. COND.

This typeout indicates an error condition on the XX tape unit. The user must correct the condition before restarting the assembly. PROGRAM SKIP key 3 has no effect on this error.

4.5.8 SET BASE ADDR. IN AL

This typeout indicates that the programmer has neglected to allocate the first program label. To correct, set AL_{15-0} to the desired base address and start. If PROGRAM SKIP key 3 is set, TRIM III arbitrarily allocates the program to address 01200, and no typeout occurs.

4.5.9 NNNNN DUP. LBL XXXXXX

This typeout occurs when the source program contains at least two identical labels. TRIM III equates all references to a duplicate label to the address of the first such label. TRIM III does not stop after the typeout. NNNNN is the sequential program line identifier number and XXXXXX is the duplicate label. If PROGRAM SKIP key 3 is set, the typeout does not occur.

4.5.10 UNALLOC TAGS
NNNNN XXXXXX AAAAA

By far the most common programmer error is the use of a tag for which no allocation was made and which does not appear anywhere in the source program as a label. TRIM III will stop after typing NNNNN XXXXXX (sequential line identifier and tag name). To correct, perform the following steps:

1) Set AL to the address at which the tag is to be allocated (if the tag is to be allocated to zero, leave AL clear).

2) If the tag refers to an instruction contained within the program being assembled, set AL_{17} to a 1.

3) If the user wishes he may allocate all future unallocated tags to the address in AL by setting AU to any nonzero value.

4) Start the computer. TRIM III types the manual allocation and uses it to continue assembly.

The typeout UNALLOC tags occurs once only. Thereafter, only the identifier and the tag are typed. If the user elects to allocate all unallocated tags to a fixed address, only the first such tag is typed.

If PROGRAM SKIP key 3 is set, TRIM III arbitrarily allocates all unallocated tags to address 00000.

4.5.11 TCS ERR XX TBL XX

This typeout indicates the table control system (TCS) of TRIM III has detected an error while attempting to operate on the indicated table. When meaningful, the number of the item being manipulated when the error occurred is displayed in AU. Table III-C-2 describes the errors which TCS detects.

TABLE III-C-3. TCS ERRORS

Error Number	Meaning	Usual Cause
1	Illegal Table Number*	
2	Illegal Media Designation*	
3	Illegal TCS Function Code*	Assembler error, Bad TRIM III tape
4	Misused Q-Replace	
5	Illogical TCS Function Sequence	
6	Table Not Found*	CALL used but no Library Directory on tape
7	Table Overflow	Too many labels, segments, or corrections
8	Too Many Tape Units Referenced or Item Length of Zero	Loose cabling
9	Unrecoverable Tape Error	
	Table 1	Bad scratch area behind TRIM III
	Table 3	Bad scratch tape

TRIM III has been designed so that a table overflow error will seldom occur. If a table overflow error does occur, the programmer may extend the limits of the table as set in the table design and restart at the TCS entrance 10012.

*Incorrect table design or control parameters.

If the limits cannot be extended, the programmer must eliminate the cause of the overflow or reassemble his program in smaller segments. **PROGRAM SKIP** key 3 has no effect on errors of this type.

4.5.12 POLY-CODE BANK OFL

This typeout indicates that generation resulting from a poly-code used in the source program has overflowed from one bank to the next. TRIM III does not stop following this typeout but will produce the selected outputs even though they will require correction. **PROGRAM SKIP** key 3 has no effect on this typeout.

SECTION IV-D. TRIM LIBRARY BUILDER (LIBBLD)

1. GENERAL INFORMATION

The TRIM III library builder, LIBBLD, is a program by means of which a user may add, delete, or replace programs on the assembler library. LIBBLD also has the capability to furnish listings of the library directory and any or all of the programs in the library. The library builder is designed for use with the following minimum equipment configuration:

1 computer with 16K words of memory
1 magnetic tape system with two transports
1 I/O console with paper tape reader, paper tape punch, and typewriter

Optional equipment includes:

1 UNIVAC 1004 card reader and high-speed printer
1 UNIVAC 1004 card punch

Input to and output from the library builder is identical to TRIM III source language formats.

A TRIM III library consists of two parts, the directory which records the names of the library subroutines ordered by their physical appearance on the tape, and the library of subroutines themselves. The library directory is stored on magnetic tape, preceding the library.

2. INPUTS

2.1 BUILDING OR UPDATING

The command tape or deck must always be preceded by a LIBUPD header control operation and a library number identification operation. For example:

→ LIBUPD • [name] • [date] →
→ LIBNO • [library number] →

When building a new library, the library number is an assignment of this number to the library. Once a library has been built, LIBBLD checks the assigned number against that of the LIBNO operation to ensure that they correspond. If they do not, an error typeout occurs.

The operations allowed in updating a library are:

REC (record - add program at beginning of library)

DEL (delete - delete program from library)

WDG (wedge - insert program in library, not at the beginning)

RPL (replace - program with new program)

Only the REC operation is used when building a new library.

These operations are accomplished in two steps:

1) Directory building or updating.
2) Library building or updating.

To insure proper calling, it is the responsibility of the user to enter all programs in the proper order in the directory. This is dictated by the fact that the magnetic tape can never be rewound when calling programs from the library during an assembly run. This requires any program internally calling on another to precede the called program on the library.

The LIBBLD checks all calls within a program to determine if they are listed in proper order in the directory, and if not, prints out a call error on the typewriter.

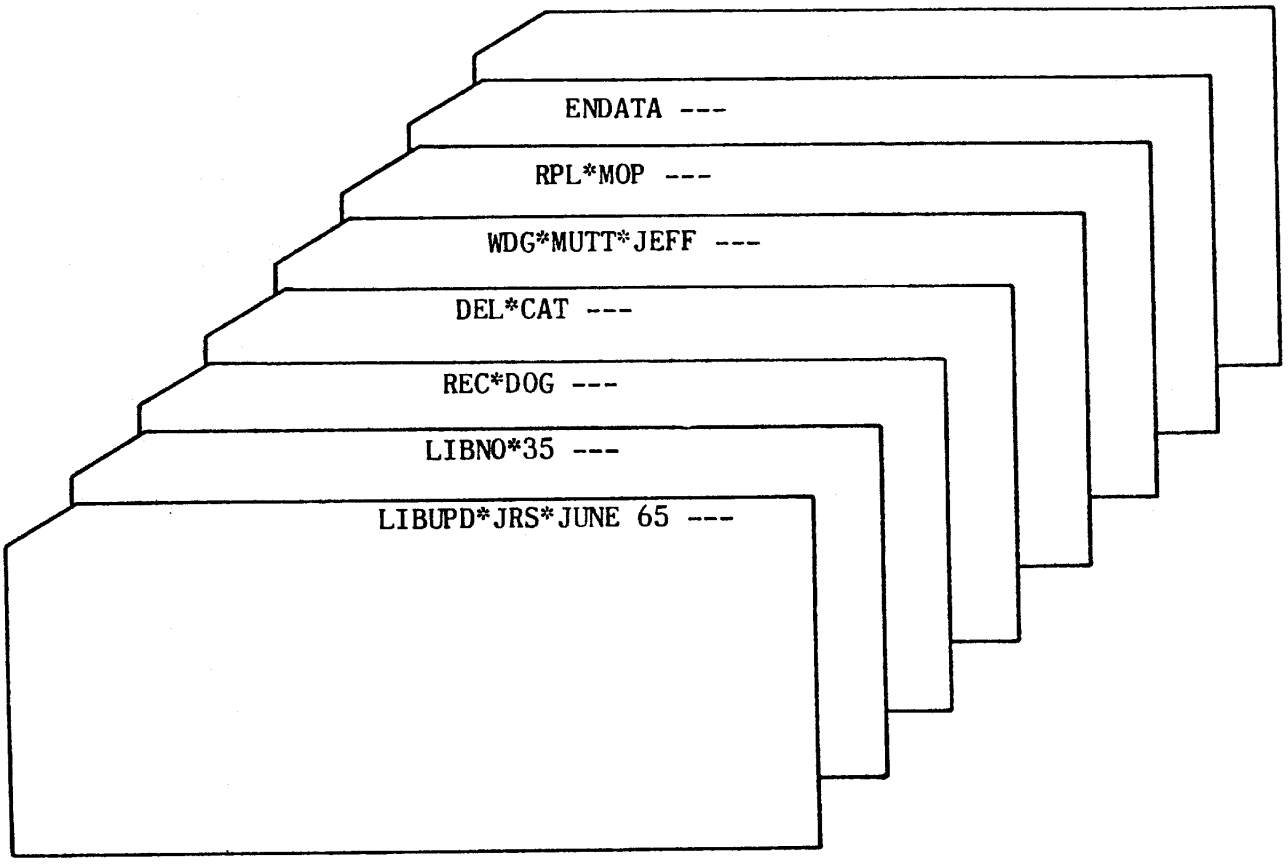
Input to LIBBLD may be either on paper tape or punched cards.

The format for paper tape input is illustrated in the following example of a command sequence:

→LIBUPD•JRS•JUNE 65 →	building or updating header.
→ LIBNO•35 →	command to update library 35.
→ REC•DOG →	command to enter program DOG at beginning.
→ DEL•CAT →	command to delete program CAT.
→ WDG•MUTT•JEFF →	command to insert program MUTT following program JEFF.
→ RPL•MOP →	command to replace old program MOP with new.
	end of commands.

Each entry must be a separate statement. The tab at the end of each statement may be omitted; however, a carriage return must precede each entry (each line shown above). A label may precede the beginning tab on the LIBUPD and LIBNO statements.

The format for card input is illustrated in the following example of command sequence:



Column 21 is the starting point of the command. A blank card must appear as the last card in the command deck.

2.2 LISTING

The command tape or deck must always be preceded by a LIBLST header control operation and a library number identification operation. For example:

→ LIBLST • [name] • [date] →

→ LIBNO • [library number] →

The user may request a listing of the directory (DIR), an individual subroutine, or the entire library of subroutines (LIB).

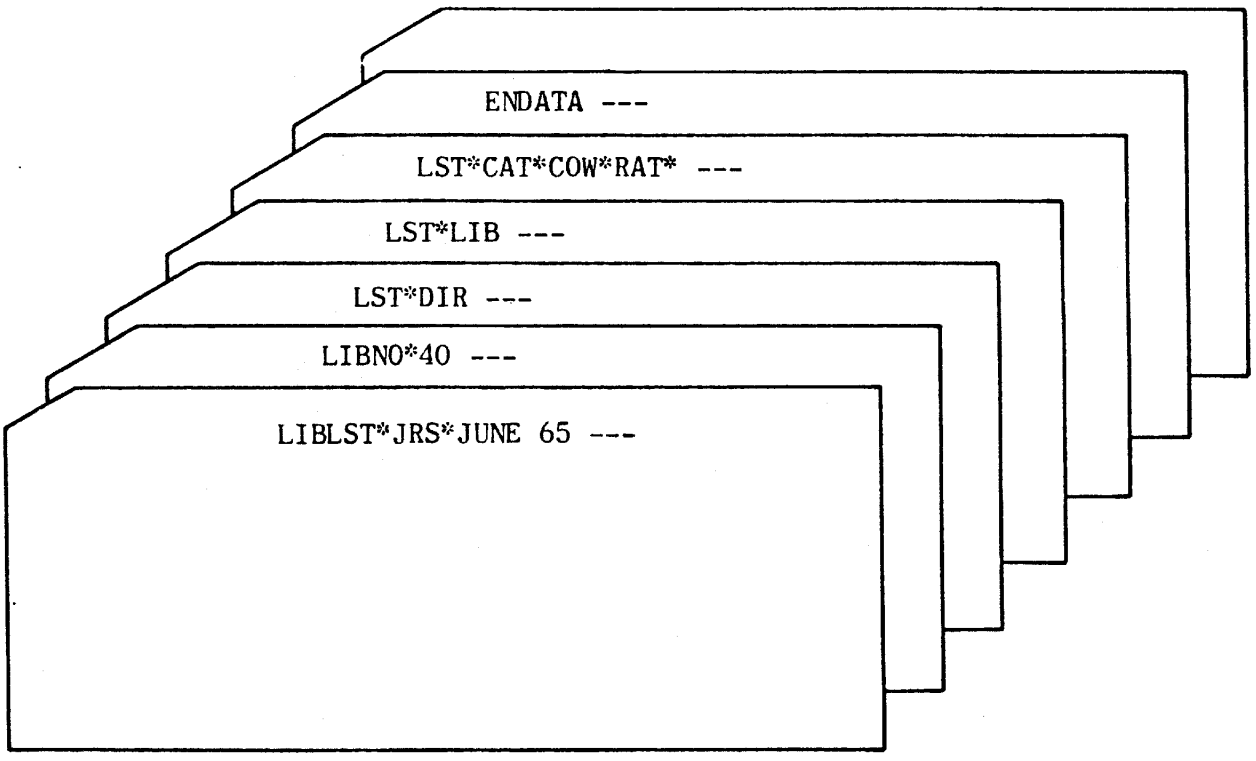
Input to LIBBLD may be either on paper tape or punched cards.

The format for paper tape input is illustrated in the following example of a command sequence:

→ LIBLST•JRS•JUNE 65 →	listing header.
→ LIBNO•40 →	command to list from library 40.
→ LST•DIR →	command to list directory.
→ LST•LIB →	command to list whole library.
→ LST•CAR•COW•RAT →	command to list programs, CAT, COW, and RAT.
	end of commands.

The tab at the end of each statement may be omitted; however, a carriage return must procede each entry (each line shown above). A label may precede the beginning tab on the LIBLST and LIBNO statements.

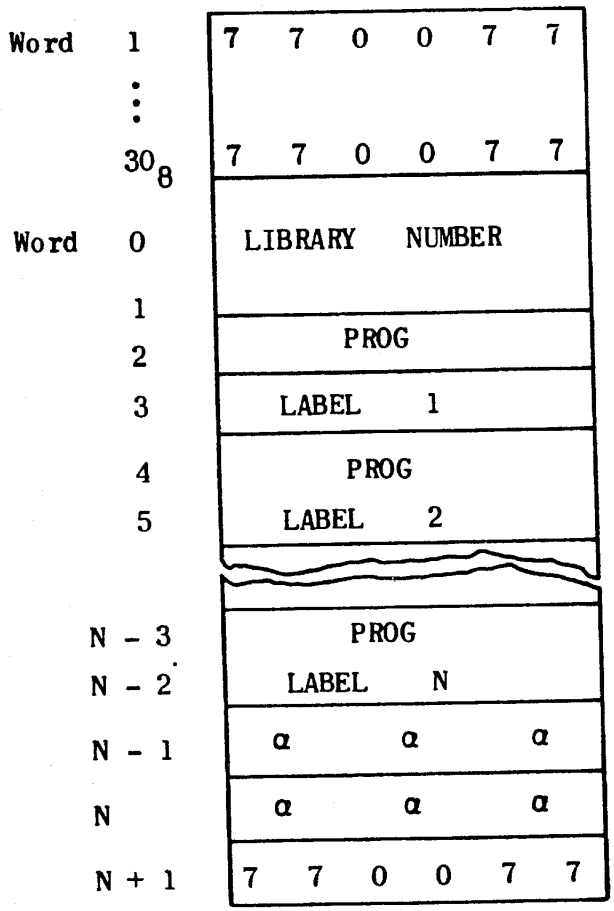
The format for card input is illustrated in the following example of a command sequence:



3. OUTPUTS

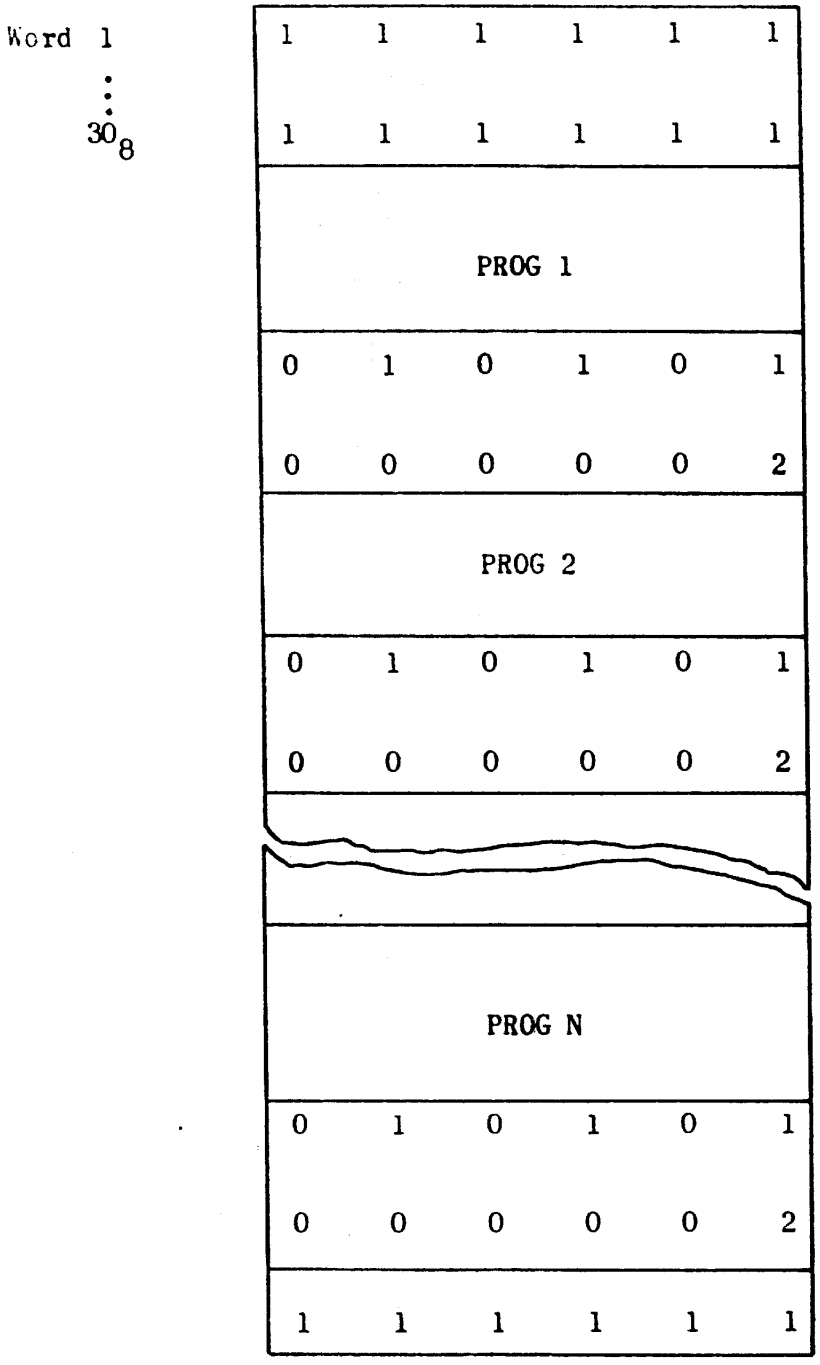
3.1 BUILDING OR UPDATING

When using LIBBLD to build or update a library, the output is the new library tape, in the format illustrated in Figures IV-D-1 and IV-D-2.



A 30_8 word block of 770077 precedes the library directory, and it is terminated by two words of alphas and one word of 770077.

Figure IV-D-1. Library Directory



A 30_8 word block of 111111 precedes the library, and it is terminated by one word of 111111. Each major program is followed by a word of DELTAS and a word of 000002.

Figure IV-D-2. Library Routines Format

3.2 LISTING

When using LIBBLD to list the directory or any or all of the library subroutines, the output may be requested on punched paper tape, punched cards, or a high-speed printer.

4. OPERATING PROCEDURES

Prior to operating the library builder, the computer, magnetic tape unit, and I/O console must be placed in the operational state with all switches in the normal operating position. If the 1004 card processor is to be used, it must also be placed in the normal operating position. The library builder absolute bioctal tape must be loaded into computer memory via a utility package. The procedures required to operate the library builder are given below. While operating the program the user must observe typeouts on the console typewriter and refer to the directions provided in paragraph 5.

4.1 LIBRARY BUILDING AND UPDATING PROCEDURES

1) Mount the tape, on which the new or updated library is to be written, on a tape transport.

2) If the process is an update, mount the old library tape on a second tape transport.

3) If input is to be from paper tape, mount the tape containing the control and command operations in the paper tape reader.

4) If input is to be from cards, mount the control and command card deck in the 1004 card input hopper and perform the following steps:

   a) Press the CLEAR, START, FEED, and RUN buttons on the 1004 console.
   b) Set PROGRAM SKIP key 3 on the computer.

5) If a new library is to built, set PROGRAM SKIP key 1.

6) If the operation is to be performed in dual-channel mode, set the appropriate channel mode switch to the dual position. If a library was written on tape in dual-channel mode, it must be updated in dual-channel mode.

7) Master clear the computer.

8) Identify the new and old library tapes in AU and AL, respectively, by setting the magnetic tape channel No. in bits 9 through 6, the cabinet No. in bits 5 through 3, and the transport No. in bits 2 through 0. If dual-channel mode is to be used, the channel No. must be odd.

9) Set P to 20000.

10) Start the computer. The control and command input is read and the computer stops.

11) Set PROGRAM SKIP key 2.

12) If any new programs are to be written on the new library, mount the input program on the appropriate input device (paper tape reader or card reader). If input is from cards, PROGRAM SKIP key 3 must be set;

if input is from tape, it must not be set. If no new programs are to be written, proceed with step 15).

13) Start the computer. The input program is read, and the computer stops.

14) Repeat steps 12) and 13) for each new program to be written on the library.

15) Release PROGRAM SKIP key 2.

16) Start the computer. The building or updating process is performed and the new or updated library is written on tape. When the process is complete, the computer stops after the typeout DONE occurs.

4.2 LIBRARY LISTING PROCEDURES

1) Mount the library tape on a tape transport.

2) If listing commands are on paper tape, mount the tape in the reader.

3) If listing commands are on cards, mount the cards in the 1004 card reader, and perform the following steps:

   a) Press the CLEAR, START, FEED and RUN buttons on the 1004 console.
   b) Set PROGRAM SKIP key 3.

4) Set PROGRAM SKIP key 0 to select output on the 1004 printer; set PROGRAM SKIP key 4 to select output on the 1004 card punch. If neither of these keys are set, output is on punched paper tape. Only one of the keys may be set during a given run.

5) Ensure that the output device is ready (paper in the printer, tape in the tape punch, cards in the card punch),

6) If card or printer output is selected and input is from tape, press the OFF, ON, CLEAR, START, FEED, and RUN buttons in sequence on the 1004 console.

7) If the operation is to be performed in dual-channel mode, set the appropriate channel mode switch to the DUAL position. If a library was written in dual-channel mode it must be listed in dual-channel mode.

8) Master clear the computer.

9) Identify the library tape address in AL by setting the magnetic tape channel No. in bits 9 through 6, the cabinet No. in bits 5 through 3, and the transport No. in bits 2 through 0. If dual-channel operation is selected, the channel No. must be odd.

10) Set P to 20000.

11) Start the computer. The listings defined by the input command operations are produced on the selected output device and the computer stops after the typeout DONE occurs.

5. TYPEOUTS

1) CALL ERR XXXXXX.

Program XXXXXX was called but is not found in the directory or is not in the proper order. It must follow the calling program on the library because the tape is never rewound while calling programs during an assembly run. Correction must be made before processing is restarted.

2) DIRECTORY.

This typeout, followed by a listing of the directory, occurs whenever a listing of the directory is requested on paper tape.

3) DONE.

The building, updating, or listing process is complete.

4) ILL-OPERATOR XXX.

An illegal updating command has been detected. Processing continues and the illegal command it ignored.

5) IMP COND.

An improper condition has been detected on a magnetic tape unit. Correct the improper condition and start the computer. The magnetic tape function will be attempted again.

6) LIBNO XXX.

Library XXX is the one being updated or listed.

7) NO HEADER.

No legal header has been read. Correct the tape or card deck, place the corrected tape or card deck in the reader, and start over.

8) NO LIBNO.

The library to be updated or listed does not contain an identifying library number. Correct the tape or card deck, place the corrected tape or card deck in the reader, and start over.

9) READ XXXXXX NEXT.

The wrong source program tape or deck was mounted in the reader. Mount the correct program and start the computer.

10) TCS ERR Z TBL Y.

The table control system has detected an error while attempting to operate on the indicated table Y. The error is unrecoverable and is the result of internal trouble or a magnetic tape error (X = 9).

11) WRONG LIBNO XXX.

XXX is the library mounted on the tape transport but it is not the one requested for updating or listing. However, if the user starts the computer, library XXX will be updated or listed. If updating is requested, the new library assumes the library number which was input via the command card deck or paper tape.

12) XXXXXX NOT IN DIR.

Program XXXXXX is either not found in the directory or has been deleted by command.