CHAPTER 7

Program Development: Software Commands— Descriptions and Formats

INTRODUCTION

The purpose of this chapter is to provide reference data for the various software development systems available for the 9900 family of microprocessors and microcomputers. Most of the information is reproduced in reference card form on heavy stock and inserted at the back of the book. Perforations allow easy removal. So pull out the ones you will use and fold for pocket reference. The specific choice of programming system will dictate which cards to pull.

Table 7-1 lists the sections in the chapter. One or more cards are made for those sections marked with a bullet. The section on Assembly Language programming describes the basic format for coding instructions and assembler directives. It is a general topic, applicable to all of the programming systems.

The 9900 reference card will come in handy for product design and programming activities for any of the processors. Explanation of the terms, mnemonics instruction execution rules, etc. can be found in Chapters 4, 5, and 6.

The complete TM 990/402 Line-by-Line Assembler User's Guide is included because this EPROM resident software is used in Chapter 9.It should serve as an illustration of the need for some form of an assembler in writing even the simplest programs. Contrast the programming effort of Chapter 3 will be the extended application of Chapter 9, and you will appreciate the power of this LBL assembler.

Reference material for the other programming systems is in the form of lists of commands and their syntax. These pages are not stand-alone documents. Software documentation is supplied with each of the programming systems and is required for full explanations of the commands and their use. Experienced designers always need assistance in recalling exact command mnemonics and their formats. Thus, this chapter supports you in any programming environment by appropriate reminders.

▸7

Table 7-1

Assembly language programming and assembler directives
• 9900 Reference Data
TM 990/402 Line-by-Line Assembler
• TIBUG Monitor
• TM 990/302 Software Development board
• TXDS Commands for the FS 990 PDS
• AMPL Reference data
• POWER BASIC Commands
• Cross Support reference data
Assembler
Simulator
Utilities

Assembly Language Programming: Formats and Directives

7◀

ASSEMBLY LANGUAGE PROGRAMMING

An assembly language is a computer oriented language for writing programs. The TMS9900 recognizes instructions in the form of 16 bit (or longer) binary numbers, called instruction or operation codes (Opcodes). Programs could be written directly in these binary codes, but it is a tedious effort, requiring frequent reference to code tables. It is simpler to use names for the instructions, and write the programs as a sequence of these easily recognizable names (called mnemonics). Then, once the program is written in mnemonic or assembly language form, it can be converted to the corresponding binary coded form (machine language form). The assembler programs described here indicate parts of PX9ASM, TXMIRA and SDSMAC, which operate on cassette, floppy disc, and moving head disc systems respectively. Several other assemblers are available from TI which provide fewer features, but operate with much smaller memory requirements.

Assembly Language Application

The assembly language programming and program verification through simulation or execution are the main elements involved in developing microprocessor programs. The overall program development effort consists of the following steps:

- Define the problem.
- Flowchart the solution to the problem.
- Write the assembly language program for the flowchart.
- Execute the Assembler to generate the machine code.
- Correct any format errors indicated by the Assembler.
- Execute the corrected machine code program on a TMS9900 computer or on a Simulator to verify program operation.

This program development sequence is defined in flowchart form in *Figure 7-1.*

▶7

Assembly Language Formats

The general assembly language source statements consists of four fields as follows:

LABEL MNEMONIC OPERANDS COMMENT

The first three fields must occur within the first 60 character positions of the source record. At least one blank must be inserted between fields.

Label Field

The label consists of from one to six characters, beginning with an alphabetic character in character position one of the source record. The label field is terminated by at least one blank. When the assembler encounters a label in an instruction it assigns the current value of the location counter to the label symbol. This is the value associated with the label symbol and is the address of the instruction in memory. If a label is not used, character position 1 must be a blank.

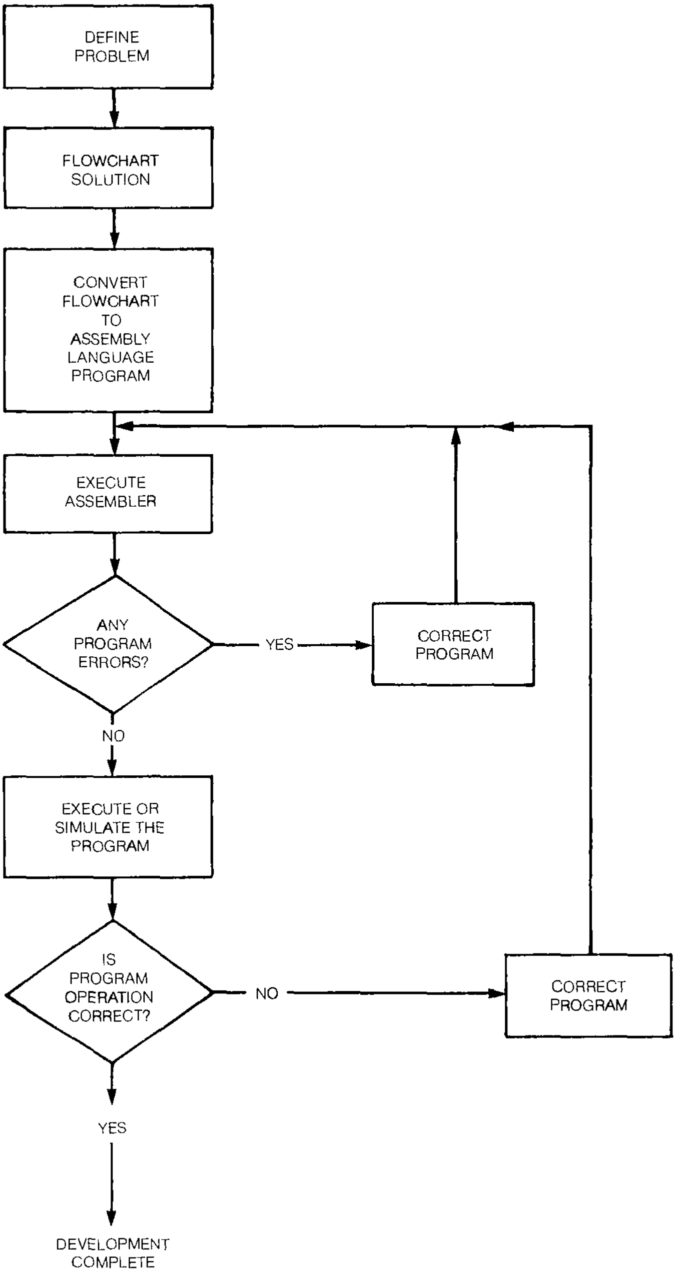


Figure 7-1. Program Development Flowchart

Mnemonic or Opcode Field

This field contains the mnemonic code of one of the instructions, one of the assembly language directives, or a symbol representing one of the program defined operations. This field begins after the last blank following the label field. Examples of instruction mnemonics include A for addition and MOV for data movement. The mnemonic field is required since it identifies which operation is to be performed.

Operands Field

The operands specify the memory locations of the data to be used by the instruction. This field begins following the last blank that follows the mnemonic field. The memory locations can be specified by using constants, symbols, or expressions, to describe one of several addressing modes available.

Comment Field

Comments can be entered after the last blank that follows the operands field. If the first character position of the source statement contains an asterisk (*), the entire source statement is a comment. Comments are listed in the source portion of the Assembler listing, but have no affect on the object code.

TERMS AND SYMBOLS

Symbols are used in the label field, the operator field, and the operand field. A symbol is a string of alphanumeric characters, beginning with an alphabetic character.
Terms are used in the operand fields of instructions and assembler directives. A term is a decimal or hexadecimal constant, an absolute assembly-time constant, or a label having an absolute value. Expressions can also be used in the operand fields of instructions and assembler directives.

▸7

Constants

Constants can be decimal integers (written as a string of numerals) in the range of −32,768 to +65,535. For example:

```
257
```

Constants can also be hexadecimal integers (a string of hexadecimal digits preceded by >). For example:

```
>09AF
```

ASCII character constants can be used by enclosing the desired character string in single quotes. For example:

```
'DX'
```

Throughout this book the subscript 16 is used to denote base 16 numbers. For example, the hexadecimal number 09AF is written $09AF_{16}$.

Symbols

Symbols must begin with an alphabetic character and contain no blanks. Only the first six characters of a symbol are processed by the Assembler.

The Assembler predefines the dollar sign ($) to represent the current location in the program. The symbols R0 through R15 are used to represent workspace registers 0 through 15, respectively.

A given symbol can be used as a label only once, since it is the symbolic name of the address of the instruction. Symbols defined with the DXOP directive are used in the OPCODE field. Any symbol in the OPERANDS field must have been used as a label or defined by a REF directive.

Expressions

Expressions are used in the OPERANDS fields of assembly language statements. An expression is a constant, a symbol, or a series of constants and symbols separated by the following arithmetic operators:

+	addition
−	subtraction
*	multiplication
/	division

Unary minus is performed first and then the expression is evaluated from left to right. A unary minus is a minus sign (negation) in front of a number or a symbol.

The expression must not contain any imbedded blanks or extended operation defined (DXOP directive) symbols.

The multiplication and division operations must be used on absolute code symbols. The result of evaluating the expression up to the multiplication or division operator must be an absolute value. There must not be more than one more relocatable symbol added to an expression than are subtracted from it.

The following are examples of valid expressions:

BLUE+1	The sum of the value of symbol BLUE plus 1.
GREEN−4	The result of subtracting 4 from the value of symbol GREEN.
2*16+RED	The sum of 32 and the value of symbol RED.
440/2−RED	220 minus the value of symbol RED.

7◂

ASSEMBLER DIRECTIVES

General Information

The assembler directives are used to assign values to program symbolic names, address locations, and data. There are directives to set up linkage between program modules and to control output format, titles, and listings.
The assembler directives take the general form of:

LABEL DIRECTIVE EXPRESSION COMMENT

The LABEL field begins in column one and extends to the first blank. It is optional on all directives except the EQU directive which requires a label. There is no label in the OPTION directive. When no label is present, the first character position in the field must be a blank. When a label is used (except in an EQU directive) the label is assigned the current value of the location counter.

The two required directives are:

IDT	Assign a name to the program
END	Terminate assembly

The most commonly used optional directives are:

EQU	Assign a value to a label or a data name.
RORG	Relocatable Origin
BYTE	Assign values to successive bytes of memory
DATA	Assign 16 bit values to successive memory words
TEXT	Assign ASCII values to successive bytes of memory

Other directives include:

▶7

AORG	Absolute (non-relocatable) Origin
DORG	Dummy Origin
BSS	Define bytes of storage beginning with symbol
BES	Define bytes of storage space ending with symbol
DXOP	Define an extended operation
NOP	No operation Pseudo-instruction
RT	Return from subroutine Pseudo-instruction
PAGE	Skip to new page before continuing listing
TITL	Define title for page headings
LIST	Allows listing of source statements
UNL	Prevents listing of source statements
OPTION	Selects output option to be used
DEF	Define symbol for external reference
REF	Reference to an external source

Required Directives

Two directives must be supplied to identify the beginning and end of the assembly language program. The IDT directive must be the first statement and the END directive must be the last statement in the assembly language program.

Program Identifier

IDT

This directive assigns a name to the program and must precede any directive that generates object code. The basic format is:

```
IDT   'Name'
```

The name is the program name consisting of up to 8 characters. As an example, if a program is to be named Convert, the basic directive would be:

```
IDT   'CONVERT'
```

The name is printed only when the directive is printed in the source listing.

Program End

END

This directive terminates the assembly. Any source statement following this directive is ignored. The basic format is:

```
END
```

Initialization Directives

These directives are used to establish values for program symbols and constants.

Define Assembly-Time Constant

EQU

Equate is used to assign values to program symbols. The symbol to be defined is placed in the label field and the value or expression is placed in the Expression field:

```
Symbol   EQU   Expression
```

The symbol can represent an address or a program parameter. This directive allows the program to be written in general symbolic form. The equate directive is used to set up the symbol values for a specific program application.

The following are examples of the use of the Equate directive:

```
TIME    EQU   HOURS+5
N       EQU   8
VAR     EQU   >8000
```

BYTE
DATA
TEXT

Initialize Memory

These directives provide for initialization of successive 8 bit bytes of memory with numerical data (BYTE directive) or with ASCII character codes (TEXT directive). The DATA directive provides for the initialization of successive 16 bit words with numerical data.

The formats are the same for all three directives:

```
Directive   Expression-list
```

The Label and Comment are optional. The expression or value list contains the data entries for the 8 bit bytes (BYTE directive), or the 16 bit words (DATA directive), or a character string enclosed in quotes (TEXT directive).

Examples of the use and effects of these directives are shown in *Figure 7-2.*

Program Location Directives

These directives affect the location counter by causing the instructions to be located in specified areas of memory.

AORG
RORG
DORG

Origin Directives

These directives set the address of the next instruction to the value listed in the expression field of the directive:

```
Directive   Expression
```

The expression field is required on all except the RORG directive. It is a value or an expression (containing only previously defined symbols). This value is the address of the next instruction and is the value that is assigned to the label (if any) and to the location counter. The AORG and DORG expressions must result in an absolute value and contain no character constants.

Example Directives:

```
        KONS    BYTE  >10, −1, 'A', 'B', N + 3
        WD1     DATA  >01FF, 3200, −'AF', 8, N+>1000
        MSG1    TEXT  'EXAMPLE'
```

AFFECTS ON MEMORY LOCATION	MEMORY DATA: DIRECTIVE ENTRY	RESULTING DATA (BINARY FORM)	RESULTING DATA (HEXADECIMAL)
KONS	>10,−1	0001 0000 1111 1111	1 OFF
KONS+2	'A', 'B'	0100 0001 0100 0010	4142
KNOS+4	N+3	0000 1011 X X	0B--
•	•	• • • •	•
•	•	• • • •	•
WD1	>01FF	0000 0001 1111 1111	01FF
WD1+2	3200	0000 1100 1000 0000	0C80
WD1+4	−'AF'	1011 1110 1011 1010	BEBA
WD1+6	8	0000 0000 0000 1000	0008
WD1+8	N+>1000	0001 0000 0000 1000	1008
•	•	• • • •	•
•	•	• • • •	•
•	•	• • • •	•
MSG1	'EX'	0100 0101 0101 1000	4558
MSG1+2	'AM'	0100 0001 0101 1101	414D
MSG1+4	'PL'	0101 0000 0100 1100	504C
MSG1+6	'E'	0100 0101 X X	4E--

XX (--) is original unaltered data in this location. N is assumed to be previously defined as 8.

Figure 7-2. Initialization Directive Examples

The AORG directive causes this value to be absolute and fixed. For example:

```
        AORG >1000+X
```

If X has been previously defined to have an absolute value of 6, the next instruction would be unalterably located at the address 1006_{16}. If a label had been included, it would have been assigned this same value.

The RORG directive causes this value to be relative or relocatable so that subsequent operations by the assembler or simulator can relocate the block of instructions to any desired area of memory. Thus, a relocatable block of instructions occupying memory locations 1000_{16} to 1020_{16} could be moved by subsequent simulator (or other software) operations to locations 2000_{16} to 2020_{16}. An example RORG statement is:

```
        SEG1   RORG >1000
```

This directive would cause SEG1 and the value of the location counter (address of the next instruction) to be set to 1000_{16}. This and all subsequent locations are relocatable.

SEG2 RORG

This directive would cause subsequent instructions to be at relocatable addresses. SEG2 and the address of the next instruction would be set to the value of the location counter.

The DORG directive causes the instructions to be listed but the assembler does not generate object code that can be passed on to simulators or other subsystems. However, symbols defined in the dummy section would then be legitimate symbols for use in the AORG or RORG program sections. For example:

DORG 0

The labels with the subsequent dummy section of instructions will be assigned values relative to the start of the section (the instruction immediately following this directive). No object code would be generated for this section.

An RORG directive is used after a DORG or AORG section to cause the subsequent instructions to be relocatable object code. If no origin directives are included in the assembly language program, all object code is relocatable starting at (referenced to) an address of 0.

Storage Allocation Directives

BES
BSS

These directives reserve a block of memory (range of addresses) for data storage by advancing the location counter by the amount specified in the expression field. Thus, the instruction after the directive will be at an address equal to the expression value plus the address of the instruction just before the directive.

▶7

Basic Formats:

BES Expression

BSS Expression

If a label is included in the BSS directive it is assigned the value of the location counter at the *first byte* if the storage block. If the label is included in the BES directive it is assigned the value of the location counter for the instruction *after* the block.

The Expression designates the number of bytes to be reserved for storage. It is a value or an expression containing no character constants. Expressions must contain only previously defined symbols and result in an absolute value.

Examples:

```
BUFF1   BES   >10
```

A 16 byte buffer is provided. Had the location counter contained the value 100_{16} (FF_{16} was the address of the previous instruction), the new value of the location counter would be 110_{16}, and this would be the value assigned to the symbol BUFF1. The next instruction after the buffer would be at address 110_{16}.

```
BUFF2   BSS   20
```

If the previous instruction is located at FF_{16}, BUFF2 will be assigned the value 100_{16}, and the next instruction will be located at 114_{16}. A 20 byte area of storage with addresses 100_{16} through 113_{16} has been reserved.

Word Boundary — EVEN

This directive causes the location counter to be set to the next even address (beginning of the next word) if it currently contains an odd address. The basic format is:

```
EVEN
```

The label is assigned the value of the location counter prior to the EVEN directive.

PROGRAM LISTING CONTROL DIRECTIVES

These directives control the printer, titling, and listing provided by the assembler.

Output Options — OPTION

The basic format of this directive is:

```
OPTION   Keyword-list
```

No label is permitted. The keywords control the listing as follows:

Keyword	*Listing*
XREF	Print a cross reference listing.
OBJ	Print a hexadecimal listing of the object code.
SYMT	Print a symbol table with the object code.

Example:

```
OPTION XREF,SYMT
```

Print a cross reference listing and the symbol table with the object code.

Advance Page PAGE

This directive causes the assembly listing to continue at the top of the next page. The basic format is:

```
PAGE
```

Page Title TITL

This directive specifies the title to be printed at the top of each page of the assembler listing. The basic format is:

```
TITL   'String'
```

The String is the title enclosed in single quotes. For example:

```
TITL   'REPORT GENERATOR'
```

Source Listing Control LIST UNL

These directives control the printing of the source listing. UNL inhibits the printing of the source listing: LIST restores the listing. The basic formats are:

```
UNL
LIST
```

Extended Operation Definition DXOP

This directive names an extended operation. Its format is:

▶7

```
DXOP   SYMBOL, Term
```

The symbol is the desired name of the extended operation. Term is the corresponding number of the extended operation. For example:

```
DXOP   DADD,13
```

defines DADD as extended operation 13. Once DADD has been so defined, it can be used as the name of a new operation, just as if it were one of the standard instruction mnemonics.

Program Linkage Directives

These directives enable program modules to be assembled separately and then integrated into an executable program.

External Definition DEF

This directive makes one or more symbols available to other programs for reference. Its basic format is:

```
DEF   Symbol-list
```

Symbol-list contains the symbols to be defined by the program being assembled. For example:

```
DEF   ENTER,  ANS
```

causes the assembler to include the Symbols ENTER and ANS in the object code so that they are available to other programs. When DEF does not precede the source statements that contain the symbols, the assembler identifies the symbols as multi-defined symbols.

External Reference REF

This directive provides access to symbols defined in other programs. The basic format is:

```
REF   Symbol-list
```

The Symbol-list contains the symbols to be included in the object code and used in the operand fields of subsequent source statements. For example:

```
REF   ARG1,ARG2
```

causes the symbols ARG1 and ARG2 to be included in the object code so that the corresponding address can be obtained from other programs.

Note: If a REF symbol is the first operand of a DATA directive causing the value of the symbol to be in 0 absolute location, the symbol will not be linked correctly in location 0.

ASSEMBLER OUTPUT

Introduction

The types of information provided by Assemblers include:

Source Listing	— Shows the source statements and the resulting object code.
Error Messages	— Errors in the assembly language program are indicated.
Cross Reference	— Summarizes the label definitions and program references.
Object Code	— Shows the object code in a tagged record format to be passed on to a computer or simulator for execution.

Source Listing

Assemblers produce a source listing showing the source statements and the resulting object code. A typical listing is shown in *Figure 7-3.*

```
0229                          *
0230                          *         DEMONSTRATE EXTERNAL REFERENCE LINKING
0231                          *
0232                                    REF      EXTR
0233     028C                           RORG
0234     028C      C820                 MOV      @EXTR, @EXTR
         028E      0000
         0290      028E'
0235     0292      28E0                 XDR      @EXTR, 3
         0294      0290'
0236     B000                           AORG       B000
0237     B000      3220                 LDCR     @EXTR, 8
         B002      0294'
0238     B004      0420                 BLWP     @EXTR
         B006      B002
0239     B008      0223                 AI       3, EXTR
         B00A      B006
0240     B00C      3BA0                 MPY      @EXTR, 2
         B00E      B00A
0241     0296                           RORG
0242     0296      CB20                 MOV      @EXTR, @EXTR
         029B      B00E
         029A      0298'
0243     029C      28E0                 XOR      @EXTR, 3
         029E      029A'
0244     C000                           AORG       C000
0245     C000      3220                 LDCR     @EXTR, 8
         C002      029E'
0246     C004      0420                 BLWP     @EXTR
         C006      C002
0247     C00B      0223                 AI       3, EXTR
         C00A      C006
0248     C00C      38A0                 MPY      @EXTR, 2
         C00E      C00A
```

Figure 7-3. Typical Source Listing.

The first line available in a listing is the title line which will be blank unless a TITL directive has been used. After this line, a line for each source statement is printed. For example:

```
0018      0156      C820      MOV      @INIT+3,@3
          0158      012B'
          015A      0003
```

In this case the source statement:

```
MOV    @INIT+3,@3
```

produces 3 lines of object code. The source statement number 18 applies to the entire 3 line entry. Each line has its own location counter value (0156, 0158, and 015A). C820 is the OPCODE for MOV with symbolic memory addressing.

012B' is the value for INIT+3. 0003 is for the direct address 3. The apostrophe (') after 012B indicates this address is program-relocatable. Source statements are numbered sequentially, whether they are listed or not (listing could be prevented by using the UNLIST directive).

9900 Reference Data

INSTRUCTION FORMAT

FORMAT (USE)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1 (ARITH)	OP CODE			B	T_D		D				T_S		S			
2 (JUMP)									SIGNED DISPLACEMENT*							
3 (LOGICAL)	OP CODE						D				T_S		S			
4 (CRU)	OP CODE						C				T_S		S			
5 (SHIFT)	OP CODE								C				W			
6 (PROGRAM)	OP CODE										T_S		S			
7 (CONTROL)	OP CODE											NOT USED				
	OP CODE											NU	W			
8 (IMMEDIATE)	IMMEDIATE VALUE															
9 (MPY,DIV,XOP)	OP CODE						D				T_S		S			

KEY

B = BYTE INDICATOR
(1 = BYTE, 0 = WORD)
T_D = D ADDR, MODIFICATION
D = DESTINATION ADDR.
T_S = ADDR. MODIFICATION

S = SOURCE ADDR.
C = XFR OR SHIFT LENGTH (COUNT)
W = WORKSPACE REGISTER NO.
* = SIGNED DISPLACEMENT OF −128 TO +127 WORDS
NU = NOT USED

T_D/T_S FIELD

	CODE	EFFECTIVE ADDRESS	MNEMONIC
00	REGISTER	WP + 2 · [S OR D]	Rn
01:	INDIRECT	(WP + 2 · [S OR D])	*Rn
10:	INDEXED (S OR D≠0)	(WP + 2 · [S OR D]) + (PC); PC ← PC + 2	NUM (Rn)
10:	SYMBOLIC (DIRECT, S OR D = 0)	(PC); PC ← PC + 2	NUM
11:	INDIRECT WITH AUTO INCREMENT	(WP + 2 · [S OR D]); INCREMENT EFF. ADDR.	*Rn+

STATUS REGISTER

0	1	2	3	4	5	6	7 … 11	12 … 15
L>	A>	=	C	O	P	X	RESERVED	INTERRUPT MASK

0 — LOGICAL GREATER THAN
1 — ARITHMETIC GREATER THAN
2 — EQUAL/TB INDICATOR
3 — CARRY FROM MSB
4 — OVERFLOW

5 — PARITY (ODD NO. OF BITS SET)
6 — XOP IN PROGRESS

INTERRUPT MASK
F = ALL INTERRUPTS ENABLED
0 = ONLY LEVEL 0 ENABLED

INTERRUPTS

TRAP ADDR	WP
TRAP ADDR + 2	PC

LEVEL	ID	TRAP ADDR	LEVEL	ID	TRAP ADDR
0	RESET	0000	8	EXTERNAL	0020
1	EXTERNAL	0004	9	EXTERNAL	0024
2	EXTERNAL	0008	10	EXTERNAL	0028
3	EXTERNAL	000C	11	EXTERNAL	002C
4	EXTERNAL	0010	12	EXTERNAL	0030
5	EXTERNAL	0014	13	EXTERNAL	0034
6	EXTERNAL	0018	14	EXTERNAL	0038
7	EXTERNAL	001C	15	EXTERNAL	003C

NOTES: 1) XOP VECTORS 0—15 OCCUPY MEMORY LOCATIONS 0040-007C
2) LOAD VECTOR OCCUPIES MEMORY LOCATIONS FFFC–FFFF

BLWP TRANSFERS	RTWP TRANSFERS	BL TRANSFER	XOP TRANSFER
WP → NEW W13	CURRENT W13 → WP	PC → W11	EFF. ADDR. → NEW W11
PC → NEW W14	CURRENT W14 → PC		WP → NEW W13
ST → NEW W15	CURRENT W15 → ST		PC → NEW W14
			ST → NEW W15
			1 → ST6

INSTRUCTIONS BY MNEMONIC

MNEMONIC	OP CODE	FORMAT	RESULT COMPARED TO ZERO	STATUS AFFECTED	INSTRUCTIONS
A	A000	1	Y	0-4	ADD(WORD)
AB	B000	1	Y	0-5	ADD(BYTE)
ABS	0740	6	Y	0-4	ABSOLUTE VALUE
AI	0220	8	Y	0-4	ADD IMMEDIATE
ANDI	0240	8	Y	0-2	AND IMMEDIATE
B	0440	6	N	—	BRANCH
BL	0680	6	N	—	BRANCH AND LINK (W11)
BLWP	0400	6	N	—	BRANCH LOAD WORKSPACE POINTER
C	8000	1	N	0-2	COMPARE (WORD)
CB	9000	1	N	0-2,5	COMPARE (BYTE)
CI	0280	8	N	0-2	COMPARE IMMEDIATE
CKOF	03C0	7	N	—	EXTERNAL CONTROL
CKON	03A0	7	N	—	EXTERNAL CONTROL
CLR	04C0	6	N	—	CLEAR OPERAND
COC	2000	3	N	2	COMPARE ONES CORRESPONDING
CZC	2400	3	N	2	COMPARE ZEROES CORRESPONDING
DEC	0600	6	Y	0-4	DECREMENT (BY ONE)
DECT	0640	6	Y	0-4	DECREMENT (BY TWO)
DIV	3C00	9	N	4	DIVIDE
IDLE	0340	7	N	—	COMPUTER IDLE
INC	0580	6	Y	0-4	INCREMENT (BY ONE)
INCT	05C0	6	Y	0-4	INCREMENT (BY TWO)
INV	0540	6	Y	0-2	INVERT (ONES COMPLEMENT)
JEQ	1300	2	N	—	JUMP EQUAL (ST2 = 1)

INSTRUCTIONS BY MNEMONIC

JGT	1500	2	N	—	JUMP GREATER THAN (ST1 = 1)
JH	1B00	2	N	—	JUMP HIGH (STO = 1 AND ST2 = 0)
JHE	1400	2	N	—	JUMP HIGH OR EQUAL (STO OR ST2 = 1)
JL	1A00	2	N	—	JUMP LOW (STO AND ST2 = 0)
JLE	1200	2	N	—	JUMP LOW OR EQUAL (STO = 0 OR ST2 =
JLT	1100	2	N	—	JUMP LESS THAN (ST1 AND ST2 = 0)
JMP	1000	2	N	—	JUMP UNCONDITIONAL
JNC	1700	2	N	—	JUMP NO CARRY (ST3 = 0)
JNE	1600	2	N	—	JUMP NOT EQUAL (ST2 = 0)
JNO	1900	2	N	—	JUMP NO OVERFLOW (ST4 = 0)
JOC	1800	2	N	—	JUMP ON CARRY (ST3 = 1)
JOP	1C00	2	N	—	JUMP ODD PARITY (ST5 = 1)
LDCR	3000	4	Y	0-2,5	LOAD CRU
LI	0200	8	N	0-2	LOAD IMMEDIATE
LIMI	0300	8	N	12-15	LOAD IMMEDIATE TO INTERRUPT MASK
LREX	00E0	7	N	12-15	EXTERNAL CONTROL
LWPI	02E0	8	N	—	LOAD IMMEDIATE TO WORKSPACE POINTER
MOV	C000	1	Y	0-2	MOVE (WORD)
MOVB	D000	1	Y	0-2,5	MOVE (BYTE)
MPY	3800	9	N	—	MULTIPLY
NEG	0500	6	Y	0-4	NEGATE (TWO'S COMPLEMENT)
ORI	0260	8	Y	0-2	OR IMMEDIATE
RSET	0360	7	N	12-15	EXTERNAL CONTROL
RTWP	0380	7	N	0-6,12-15	RETURN WORKSPACE POINTER
S	6000	1	Y	0-4	SUBTRACT (WORD)
SB	7000	1	Y	0-5	SUBTRACT (BYTE)
SBO	1D00	2	N	—	SET CRU BIT TO ONE
[illegible]	1E00	2	N	—	SET CRU BIT TO ZERO
[illegible]O	0700	6	N	—	SET ONES
SLA	0A00	5	Y	0-4	SHIFT LEFT (ZERO FILL)
SOC	E000	1	Y	0-2	SET ONES CORRESPONDING (WORD)
SOCB	F000	1	Y	0-2,5	SET ONES CORRESPONDING (BYTE)
SRA	0800	5	Y	0-3	SHIFT RIGHT (MSB EXTENDED)
SRC	0800	5	Y	0-3	SHIFT RIGHT CIRCULAR
SRL	0900	5	Y	0-3	SHIFT RIGHT (LEADING ZERO FILL)
STCR	3400	4	Y	0-2,5	STORE FROM CRU
STST	02C0	8	N	—	STORE STATUS REGISTER
STWP	02A0	8	N	—	STORE WORKSPACE POINTER
SWPB	06C0	6	N	—	SWAP BYTES
SZC	4000	1	Y	0-2	SET ZEROES CORRESPONDING (WORD)
SZCB	5000	1	Y	0-2,5	[illegible] ZE[illegible]S CORRESPONDING (BYTE)
TB	1F00	2	N	2	TEST CRU BIT
X	0480	6	N	—	EXECUTE
XOP	2C00	9	N	6	EXTENDED OPERATION
XOR	2800	3	Y	0-2	EXCLUSIVE OR
DCA	2C00	9	N	0-3,5,7	DECIMAL CORRECT ADD
DCS	2C00	9	N	0-3,5,7	DECIMAL CORRECT SUB
LIIM	2C00	9	N	14,15	LOAD INTERRUPT MASK

ILLEGAL OP CODES 0000-01FF;0320-033F;0780-07FF;0C00-OFFF

▶7

INSTRUCTIONS BY OP CODE

OP CODE	MNEMONIC	OP CODE	MNEMONIC
0000-01FF	ILLEGAL	1000	JMP
0200	LI	1100	JLT
0220	AI	1200	JLE
0240	ANDI	1300	JEQ
0260	ORI	1400	JHE
0280	LI	1500	JGT
0240	STWP	1600	JNE
02C0	STST	1700	JNC
02E0	LWPI	1800	JOC
0300	LIMI	1900	JND
0320-033F	ILLEGAL	1A00	JL
0340	IDLE	1B00	JH
0360	RSET	1C00	JOP
0380	RTWP	1D00	SBO
03A0	CKON	1E00	SBZ
03C0	CKOF	1F00	TB
03E0	LREX	2000	COC
0400	BWLP	2400	CZC
0440	B	2800	XOR
0480	X	2C00	XOP
04C0	CLR	3000	LDCR
0500	NEG	3400	STCR
0540	INV	3800	MPY
0580	INC	3C00	DIV
05C0	INCT	4000	SZC
0600	DEC	5000	SZCB
0640	DECT	6000	S
0680	BL	7000	SB
06C0	SWPB	8000	C
0700	SETO	9000	CB
0740	ABS	A000	A
0780-07FF	ILLEGAL	B000	AB
0800	SRA	C000	MOV
0900	SRL	D000	MOVB
0A00	SLA	E000	SOC
0B00	SRC	F000	SOCB
0C00	ILLEGAL		

7◀

PSEUDO-INSTRUCTIONS

MNEMONIC	PSEUDO-INSTRUCTIONS	CODE GENERATED
NOP	NO OPERATION	1000
RT	RETURN	0458

PIN DESCRIPTIONS

PIN #	FUNCTION	PIN #	FUNCTION	PIN #	FUNCTION
1	V_{BB}	23	A1	44	D3
2	V_{CC}	24	A0	45	D4
3	WAIT	25	$\phi4$	46	D5
4	$\overline{\text{LOAD}}$	26	V_{SS}	47	D6
5	HOLDA	27	V_{DD}	48	D7
6	$\overline{\text{RESET}}$	28	$\phi3$	49	D8
7	IAQ	29	DBIN	50	D9
8	$\phi1$	30	CRUOUT	51	D10
9	$\phi2$	31	CRUIN	52	D11
10	A14	32	$\overline{\text{INTREQ}}$	53	D12
11	A13	33	IC3	54	D13
12	A12	34	IC2	55	D14
13	A11	35	IC1	56	D15
14	A10	36	IC0	57	NC
15	A9	37	NC	58	NC
16	A8	38	NC	59	NC
17	A7	39	NC	60	CRUCLK
18	A6	40	NC	61	$\overline{\text{WE}}$
19	A5	41	D0	62	READY
20	A4	42	D1	63	$\overline{\text{MEMEN}}$
21	A3	43	D2	64	$\overline{\text{HOLD}}$
22	A2				

ASSEMBLER DIRECTIVES

MNEMONIC	DIRECTIVE
AORG	ABSOLUTE ORIGIN
BES	BLOCK ENDING WITH SYMBOL
BSS	BLOCK STARTING WITH SYMBOL
BYTE	INITIALIZE BYTE
DATA	INITIALIZE WORD
DEF	EXTERNAL DEFINITION
DORG	DUMMY ORIGIN
DXOP	DEFINE EXTENDED OPERATION
END	PROGRAM END
EQU	DEFINITE ASSEMBLY — TIME CONSTANT
EVEN	WORD BOUNDARY
IDT	PROGRAM IDENTIFIER
LIST	LIST SOURCE
PAGE	PAGE EJECT
REF	EXTERNAL REFERENCE
RORG	RELOCATABLE ORIGIN
TEXT	INITIALIZE TEXT
TITL	PAGE TITLE
UNL	NO SOURCE LIST

USASCII/HOLLERITH CHARACTER CODE

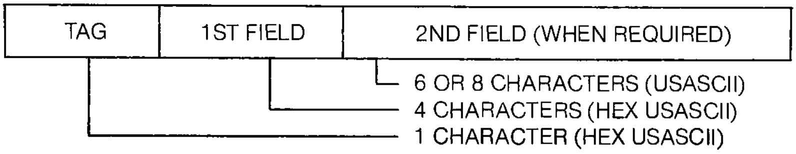
CHAR.	USASCII (HEXADECIMAL)	HOLLERITH*	CHAR.	USASCII (HEXADECIMAL)	HOLLERITH*
NUL	00		3	33	3
SOH	01		4	34	4
STX	02		5	35	5
ETX	03		6	36	6
EOT	04		7	37	7
ENQ	05		8	38	8
ACK	06		9	39	9
BEL	07		:	3A	2-8
BS	08		;	3B	11-6-8
HT	09		<	3C	12-4-8
LF	0A		=	3D	6-8
VT	0B		>	3E	0-6-8
FF	0C		?	3F	0-7-8
CR	0D		@	40	4-8
S0	0E		A/a	41/61	12-1
SI	0F		B/b	42/62	12-2
DLE	10		C/c	43/63	12-3
DC1	11		D/d	44/64	12-4
DC2	12		E/e	45/64	12-5
DC3	13		F/f	46/66	12-6
DC4	14		G/g	47/67	12-7
NAK	15		H/h	48/68	12-8
SYN	16		I/i	49/69	12-9
ETB	17		J/j	4A/6A	11-1
CAN	18		K/k	4B/6B	11-2
EM	19		L/l	4C/6C	11-3
SUB	1A		M/m	4D/6D	11-4
ESC	1B		N/n	4E/6E	11-5
	1C		O/o	4F/6F	11-6
GS	1D		P/p	50/70	11-7
RS	1E		Q/q	51/71	11-8
US	1F		R/r	52/72	11-9
SPACE	20	BLANK	S/s	53/73	0-2
!	21	11-2-8	T/t	54/74	0-3
"	22	7-8	U/u	55/75	0-4
#	23	3-8	V/v	56/76	0-5
$	24	11-3-8	W/w	57/77	0-6
%	25	0-4-8	X/x	58/78	0-7
&	26	12	Y/y	59/79	0-8
'	27	5-8	Z/z	5A/7A	0-9
(	28	12-5-8	[	[illegible]	12-2-8
)	29	11-5-8	\	5C	
*	2A	11-4-8	]	5D	12-7-8
+	2B	12-6-8	Λ	5E	11-7-8
,	2C	0-3-8	—	SF	0-5-8
-	2D	11	`	60	
.	2E	12-3-8	{	7B	
/	2F	0-1	>	7C	
0	30	0	}	7D	
1	31	1	~	7E	
2	32	2	DEL	7F	

*PUNCH IN CARD ROWS

HEX-DECIMAL TABLE

EVEN BYTE				ODD BYTE			
HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,766	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,066	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

OBJECT RECORD FORMAT AND CODE


TAG
1ST FIELD
2ND FIELD (WHEN REQUIRED)
6 OR 8 CHARACTERS (USASCII)
4 CHARACTERS (HEX USASCII)
1 CHARACTER (HEX USASCII)

TAG	FIRST FIELD	SECOND FIELD	MEANING
0	LENGTH OF ALL RELOCATABLE CODE	PROGRAM ID (8-CHARACTER)	PROGRAM START
1	ADDRESS	(NOT USED)	ABSOLUTE ENTRY ADDRESS
2	ADDRESS	(NOT USED)	RELOCATABLE ENTRY ADDRESS
3	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN RELOCATABLE CODE
4	LOCATION OF LAST APPEARANCE OF SYMBOL	6 CHARACTER SYMBOL	EXTERNAL REFERENCE LAST USED IN ABSOLUTE CODE
5	LOCATION	6 CHARACTER SYMBOL	RELOCATABLE EXTERNAL DEFINITION
6	LOCATION	6 CHARACTER SYMBOL	ABSOLUTE EXTERNAL DEFINITION
7	CHECKSUM FOR CURRENT RECORD	(NOT USED)	CHECKSUM
8	ANY VALUE	(NOT USED)	IGNORE CHECKSUM VALUE
9	LOAD ADDRESS	(NOT USED)	ABSOLUTE LOAD ADDRESS
A	LOAD SDDRESS	(NOT USED)	RELOCATABLE LOAD ADDRESS
B	DATA	(NOT USED)	ABSOLUTE DATA
C	DATA	(NOT USED)	RELOCATABLE DATA
D	LOAD BIAS	(NOT USED)	LOAD BIAS OR OFFSET (NOT A PART OF ASSEMBLER OUTPUT)
E			ILLEGAL
F	(NOT USED)	(NOT USED)	END OF RECORD

TM990/402 Line-by-Line Assembler User's Guide

7

GENERAL

The TM 990/402 Line-By-Line Assembler (LBLA) is a standalone program that assembles into object code the 69 instructions used by the TM 990/100M/101M/180M microcomputers. Comments can be a part of the source statement; however, assembler directives are not recognized. Assembler TM 990/402-1 consists of two EPROM's and supports the TM 990/100M microcomputer. TM 990/402-2 consists of one EPROM and supports the TM 990/180M microcomputer.

INSTALLATION

Remove the TMS 2708 chip(s) from the package and install as follows (see *Figure 1*):

(1) Turn off power to the TM 990/1XXM microcomputer.

(2) Place the chip(s) into the proper socket(s) as shown in *Figure 1.* The shaded components in *Figure 1* denote the LBLA EPROM's correctly placed in their sockets. The corresponding socket number (UXX number) is marked on the EPROM.

NOTES

1. Place the TMS 2708(s) into the socket(s) with pin 1 in the lower left corner as denoted by a 1 on the board and on the EPROM. Be careful to prevent bending of the pins.
2. Do not remove EPROM's containing the monitor as shown in *Figure 1.* The monitor is used by the assembler.

(3) Verify proper positioning in the sockets. Apply power to the microcomputer board.

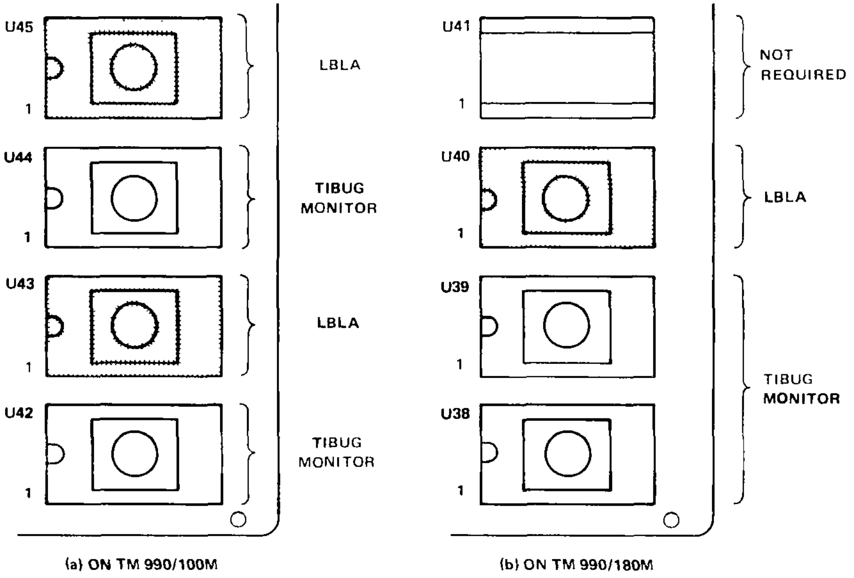


Figure 1. Placement of TMS 2708 Eprom's

OPERATION

Setup

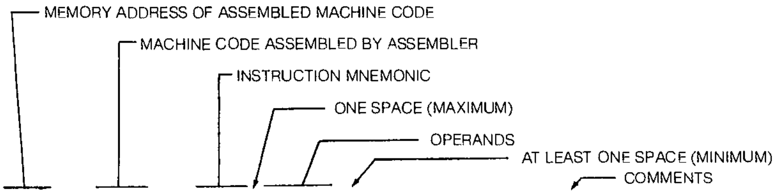
NOTE
The examples in this guide use memory addresses obtainable in RAM on the TM 990/100M microcomputer. To exemplify the TM 990/180M addressing scheme, the reader should substitute a 3 for the F in the most significant digit (left most) of a four-digit memory address in the following examples (e.g., $3EE0_{16}$ for $FEE0_{16}$).

- With the Line-By-Line Assembler EPROMs installed, call up the monitor by pressing the RESET switch in the upper left corner of the board and then pressing the A key at the terminal.
- Invoke the R keyboard command and set the Program Counter (PC) to $09E6_{16}$. This is the memory address entry point for the Line-By-Line Assembler.
- Invoke the E (execute) command. The assembler will execute and print the memory address (M.A.) $FE00_{16}$ for the TM 990/100 or $3E00_{16}$ for the TM 990/180M. The printhead will space to the assembly language opcode input column and wait for input from the keyboard.

```
?R
W=0BA4
P=000F       9E6 ←———— LBL A ENTRY ADDRESS
?E
FE00
```

Inputs To Assembler

The Line-By-Line Assembler accepts assembly language inputs from a terminal. As each instruction is input, the assembler interprets it, places the resulting machine code in an absolute address, and prints the machine code (in hexadecimal) next to its absolute address:


MEMORY ADDRESS OF ASSEMBLED MACHINE CODE
MACHINE CODE ASSEMBLED BY ASSEMBLER
INSTRUCTION MNEMONIC
ONE SPACE (MAXIMUM)
OPERANDS
AT LEAST ONE SPACE (MINIMUM)
COMMENTS

```
FE00   02E0   LWPI >FE80          SET UP WORKSPACE ADDRESS
FE02   FE80
FE04   0200   LI R0,10            SET UP COUNTER VALUE
FE06   000A
FE08   0201   LI R1,>FEA0         ADDRESS OF VALUES IN R1
FE0A   FEA0
FE0C   0202   LI R2,>FEB0         ADDRESS OF STORAGE AREA IN R2
FE0E   FEB0
FE10   CCB1   MOV *R1+,*R2+       MOVE VALUES TO STORAGE AREA
FE12   0600   DEC R0              DECREMENT COUNTER
FE14   1301   JEQ >FE18           EXIT IF COUNTER = ZERO
FE16   10FC   JMP >FE10           LOOP BACK UNTIL 10 VALUES MOVED
FE18
```

Use only one space between the mnemonic and the operand. If you use the comment field, use at least one space between the operand and comment. If no comment is used, complete the instruction with a space and carriage return. If a comment is used, only a carriage return is required.

No loader tags are created; code is loaded in contiguous memory addresses by the assembler. The location can be changed as desired (explained in paragraph 3.2.2).

Labels cannot be used. Addressing is by byte displacement (jump instructions) or by absolute memory address.

> NOTE
>
> Be aware that the workspace for the TIBUG monitor begins in RAM at address $FFB0_{16}$ for the TM 990/100M and begins at address $3FB0_{16}$ for the TM 990/180M. Understand that assembled object code should not be entered at or above these addresses.

Program Preparation

Set up your program using flow charts with code written on a coding pad. Do not use assembler directives.

Changing Absolute Load Address

Code is located at the address written on the assembler output. When initialized, the assembler loads code contiguously starting at M.A. $FE00_{16}$ ($3E00_{16}$ for TM 990/180M). This address can be changed at any time during assembly by typing a slash (/) followed by the desired M.A.:

▶7

```
FE80   8081   C R1,R2            COMPARE VALUES
FE82   1301   JEQ >FE86          IF EQUAL, SKIP ERROR ROUTINE
FE84   06A0   BL @>FF20          OTHERWISE DO ERROR ROUTINE
FE86   FF20
FE88          /FF20       ←————— CHANGE ADDRESS
FF20   2FA0   XOP @>FF26,14      SEND ERROR MESSAGE (See TIBUG Monitor)
FF22   FF26
FF24   045B   B ◆R11             RETURN TO CALLING PROGRAM
FF26   0A0D   +>0A0D
FF28   4552   $ERROR FOUND
FF2A   524F
FF2C   5220
FF2E   464F
FF30   554E
FF32   4420
FF34   0000   +0000
FF36          /FE86       ←————— CHANGE ADDRESS
FE86
```

Note that this is similar to using an AORG (absolute origin) 990 assembler directive.

Entering Instructions

Any of the 69 instructions applicable to the TM 990/1XXM microcomputers can be interpreted by the Line-By-Line Assembler. The following apply:

(1) Place one space between instruction mnemonic and operand.

(2) Terminate entire instruction with a space and a carriage return. Lines with comments need only a carriage return. Character strings require two carriage returns.

(3) Do not use labels; addressing is through byte displacement (jump instructions) or absolute addresses:

```
FE8C    1607    JNE     $+16
FC8E    10E8    JMP     >FE60
FE90    C8A2    MOV     @>FD20(R2), @>FE10(R2)
FE92    FD20
FE94    FE10
FE96
```

(4) Register numbers are in decimal and can be predefined (preceded by an R):

```
FE96    020C    LI 12,>D00
FE98    0D00
FE9A    020D    LI R13,>FFFF
FE9C    FFFF
FE9E
```

(5) Jump instruction operand can be $ + n, $ − n, or > M where n is a decimal value of bytes (+ 256≥n≥ − 254) and M is a memory address in hexadecimal. The dollar sign must be followed by a sign and number (JMP $ is not allowed).

```
FE20    1304    JEQ     $+10             EXIT
FE22    1304    JEQ     $+>A             EXIT
FE24    1304    JEQ     $+%1010          EXIT
FE26    1304    JEQ     >FE30            EXIT
FE28    10FF    JMP     $+0              LOOP AT THIS ADDRESS (>FE28)
FE2A    10FF    JMP     $-0              LOOP AT THIS ADORESS
```

(6) Absolute numerical values can be in binary, decimal, or hexadecimal.

- Binary values are preceded by a percent sign (%). One to 16 ones and zeroes can follow; unspecified bits on the left will be zero filled:

```
FE58    0204    LI R4,%10101010          >AA IN R4
FE5A    00AA
FE5C    000A    +%1010                   DATA STATEMENT
FE5E    FFF6    -%1010                   DATA STATEMENT
FE60
```

7◀

- Decimal values have no prefix in an operand:

```
FE6C    0205    LI R5,100                 LOAD COUNTER
FE6E    0064
FE70    0206    LI R6,32768               SET LIMIT
FE72    8000
FE74    8000    +32768
FE76    8000    -32768
FE78    7FFF    +32767
FE7A    8001    -32767
FE7C    FFFF    -1
FE7E
```

- Hexadecimal values are preceded by the greater-than sign (>):

```
FE7E    02E0    LWPI>FF00                 SET WP ADDRESS
FE80    FF00
FE82    FFFF    +>FFFF                    DATA STATEMENT
FE84    0001    ÷>FFFF                    DATA STATEMENT
FE86
```

NOTE

In operands, absolute value must be unsigned values only. However, there is a method for using the assembler to compute and assemble a negative value; this method is especially useful with the immediate instructions (e.g., AI, CI, LI). Enter the instruction using the negative value. The assembled value will be all zeroes in the last assembled word. Use the slash command (paragraph 3.2.2) to assemble at the previous address, then enter the negative value as a data statement as shown in the following example:

```
FE1A    0201    LI R1,->100       ←USE SIGNED OPERAND
FE1C    0000                      ←SIGNED NUMBER ASSEMBLIES AS 0000 (IN M.A.>FE1C)
FE1E            /FE1C             ←SET OBJECT LOAD ADDRESS TO PREVIOUS ADDRESS
FE1C    FF00    ->100             ←->100(>FF00) NOW IN M.A.>FE1C
FE1E
```

▶7

(7) Absolute addresses are used instead of labels:

```
FEA0    C820    MOV     @>FE10,@>FED0           MOVE TO STORAGE
FEA2    FE10
FEA4    FED0
FEA6    16FC    JNE     >FEA0                   LOOP BACK TO MOVE INSTRUCTION
FEA8
```

(8) Character strings are preceded by a dollar sign and are terminated with two carriage returns.

```
FF10    4142    $ABCD        1233
FF12    4344
FF14    2020
FF16    2031
FF18    3233
FF1A    3320                               ←— UNUSED RIGHT BYTE FILLED WITH>20 (SPACE)
```

(9) Character strings of one or two characters can be designated by encoding the string in quotes. If not part of an operand, a plus or minus sign must precede the value. If the string is larger than two characters, the last two characters are interpreted.

```
FEAA    3132    +'12'          CHARACTERS ONE AND TWD
FEAC    000C    +12            VALUE DF POSITIVE TWELVE
FEAE    FFF4    -12            VALUE OF NEGATIVE TWELVE
FEB0    0000    +              + FOLLOWED BY CTRL KEY AND NULL KEY PRESSED
FEB2    0202    LIR2, 'ABCD''  ASSEMBLED LAST TWO CHARACTERS [C AND D]
FEB4    4344
FEB6    0202    LI R2, 'E'     CHARACTER E IN RIGHT BYTE
FEB8    0045
FEBA    0202    LI R2,>E       VALUE >E IN RIGHT BYTE
FEBC    D00E
FEBE
```

(10) Signed numerical values of up to 16 bits can be designated by preceding the value with a plus or minus sign. If more than 16 bits are entered in binary or hexadecimal, the last 16 bits entered are used. If more than 16 bits are entered in decimal, the assembled value is the same as the remainder had the number between divided by 2^{15} ($65{,}536_{10}$).

```
FE18    00FF    +%111111110000000011111111
FE1A    FF01    -%111111110000000011111111
FE1C    AAEE    +>AAAAAAEE
FE1E    8000    +32768
FE20    80D1    +32769
FE22    0000    +65536
FE24    FFFF    +131071
FE26    0000    +131072
FE28    B000    -32768
FE2A    8001    -32767
FE2C    7FFF    -32769
FE2E
```

ERRORS

When the assembler detects an error, it types an error symbol and readies the terminal for re-entering data at the same memory address. The following error symbols are used:

- D (Displacement error). The jump instruction destination is more than +256 or −254 bytes away.

```
FF38          JNC     $+300◆D
FF38          JNC     >F000◆D
FF38   170B   JNC     >FF50
FF3A
```

- R (Range error). The operand is out of range for its field:

```
FF30          LI      R44,◆R
FE30   0204   LI      R4,200
FF32   00C8
```

- S (Syntax error). The instruction syntax was incorrect:

```
FF34          MOZ◆S  }
FF34          MOS◆S  }  INCORRECT MNEMONICS
FF34   C802   MOV R2, @>FE90
FF36   FE90
```

EXITING TO THE MONITOR

Return control to monitor by pressing the escape (ESC) key.

PSEUDO-INSTRUCTIONS

The TM 990/402 also interprets two pseudo-instructions. These pseudo-instructions are not additional instructions but actually are additional mnemonics that conveniently represent two members of the instruction set:

- The NOP mnemonic can be used in place of a JMP $ + 2 instruction which is essentially a no-op (no operation). This can be used to replace an existing instruction in memory, or it can be included in code to force additional execution time in a routine. Both NOP and JMP $ + 2 assemble to the machine code 1000_{16}.
- The RT mnemonic can be used in place of a B *R11 instruction which is a common return from a branch and link (BL) subroutine. Both RT and B *R11 assemble to the machine code $045B_{16}$.

Note the following examples:

```
FE00 1000 JMP $+2                  JUMP TO NEXT INSTRUCTION
FE02 1000 NOP                      ALSO ASSEMBLES TO >1000
FE04 045B B ◆R11                   RETURN COMMAND
FE06 045B RT                       ALSO A RETURN COMMAND
```

TIBUG Monitor

7◀

TIBUG COMMANDS

INPUT	RESULTS
B	Execute under Breakpoint
C	CRU Inspect/Change
D	Dump Memory to Cassette/Paper Tape
E	Execute
F	Find Word/Byte in Memory
H	Hex Arithmetic
L	Load Memory from Cassette/Paper Tape
M	Memory Inspect/Change
R	Inspect/Change User WP, PC, and ST Registers
S	Execute in Step Mode
T	1200 Baud Terminal
W	Inspect/Change Current User Workspace

COMMAND SYNTAX CONVENTIONS

CONVENTION SYMBOL	EXPLANATION
< >	Items to be supplied by the user. The term within the angle brackets is a generic term.
[ ]	Optional Item — May be included or omitted at the user's discretion. Items not included in brackets are required.
{ }	One of several optional items must be chosen.
(CR)	Carriage Return
∧	Space Bar
LF	Line Feed
R or Rn	Register (n = 0 to 15)
WP	Current User Workspace Pointer contents
PC	Current User Program Counter contents
ST	Current User Status Register contents

USER ACCESSIBLE UTILITIES

XOP	FUNCTION
8	Write 1 Hexadecimal Charter to Terminal
9	Read Hexadecimal Word from Terminal
10	Write 4 Hexadecimal Characters to Terminal
11	Echo Character
12	Write 1 Character to Terminal
13	Read 1 Character from Terminal
14	Write Message to Terminal
	NOTE All characters are in ASCII code.

TIBUG ERROR MESSAGES

ERROR	CONDITION
0	Invalid tag detected by the loader.
1	Checksum error detected by the loader.
2	Invalid termination character detected.
3	Null input field detected by the dump routine.
4	Invalid command entered.

COMMAND	SYNTAX
Execute under Breakpoint (B)	B<address><(CR)>
CRU Inspect/Change (C)	C<base address>{∧,}<count><(CR)>
Dump Memory to Cassette/Paper Tape (D)	D<start address>{∧,}<stop address>{∧,}<entry address>{∧,}IDT=<name><∧> MONITOR PROMPT
Execute Command (E)	E
Find Command (F)	F<start address>{∧,}<stop address>{∧,}<value>{$(\overline{\text{CR}})$}
Hexadecimal Arithmetic (H)	H<number 1>{∧,}<number 2><(CR)>
Load Memory from Cassette or Paper Tape (L)	L<bias><(CR)>
Memory Inspect/Change, Memory Dump (M)	Memory Inspect/Change Syntax M<address><(CR)> Memory Dump Syntax M<start address>{∧,}<stop address><(CR)>
Inspect/Change User WP,PC, and ST Registers (R)	R<(CR)>
Execute In Single Step Mode (S)	S
TI 733 ASR Baud Rate (T)	T
Inspect/Change User Workspace (W)	W [Register Number] <(CR)>

TM 990/302
Software Development Board

7◀

EPROM's which may be programmed by the '302

2708
2716
2516
2532
9940

SOFTWARE COMPONENTS

	Access Command
Executive	(CR)
Text Editor	TE
Symbolic Assembler	SA
Debug Package	DP
EPROM Programmer	EP
Relocating Loader	RL
EIA Interface	EI
I/O Scheduler/Handler	SR

LUNO ASSIGNMENTS

Device	Logical Unit No.
Dummy	0
Terminal (LOG)	1
Audio Cassette 1	2
Audio Cassette 2	3
Second EIA Connector	4
Memory	5

SOFTWARE COMPONENT CALLS

Text Editor	TEb̸(input device),(output device)
Symbolic Assembler	SAb̸(source device), (object device), (listing device)
Debug Package	DPb̸(output device)
EPROM Programmer	EP
Relocating Loader	RLb̸(input device)
Set Baud Rate	SRb̸(nnnn)
Escape	ESC (return to executive)

TEXT EDITOR COMMANDS

D	Delete lines n thru m
I	Insert at line n with optional auto increment by m
K	Keep buffer and print new top line in the buffer
G	Get buffer and print new bottom line in the buffer
P	Print lines n thru m
Q	Flush the input file until end of input file and return to executive
R	Resequence input to output, n is initialized line # and m is the increment

COMMAND	SYNTAX
Delete Lines n thru m (Rn,m)	D (starting line #)[,(ending line #)]
Insert After Line n with optional auto increment by m (In,m)	I (line number after which new data is entered) [,(auto increment value)]
Get Buffer (G)	G
Keep Buffer (K)	K
Print lines n thru m (Pn,m)	P (first line # to be printed) [,(last line # to be printed)]
Quit Text Editor (Q)	Q
Resequence Output (Rn,m)	R (initial line number) [,(increment value)]

ASSEMBLER DIRECTIVES

AORG	[label]ƀAORGƀ(value)ƀ[comment]
BSS	[label]ƀBSSƀ(value)ƀ[comment]
BYTE	[label]ƀBYTEƀ(value),(value),(value),....ƀ[comment]
DXOP	[label]ƀDXOPƀ(symbol),(value)ƀ[comment]
END	[label]ƀENDƀ(symbol)ƀ[comment]
EQU	[label]ƀEQUƀ(expression)ƀ[comment]
DATA	[label]ƀDATAƀ(exp),(exp),...ƀ[comment]
EVEN	[label]ƀEVENƀ[comment]
IDT	[label]ƀIDTb(string)ƀ[comment]
TEXT	[label]ƀTEXTƀ(–),'string'ƀ[comment)

DEBUG Package

Verb	Command
SB	Set Software Breakpoint and Execute
IM	Inspect/Change Memory
IC	Inspect/Change CRU
IR	Inspect/Change MPU Registers
ST	Set Software Trace
RU	Single Step for 1 or more instructions with or without trace
DM	Dump Memory

▸7

DEBUG COMMANDS

Set Breakpoint and Execute	SBƀ(address)
Inspect/Change Memory	IMƀ(address)
Inspect/Change CRU	ICƀ(CRU base addr.)(no. of bits)
Inspect/Change MPU registers	IR
Set Software Trace	STƀ(0 or 1)
Run 1 or more Instructions	RUƀ(no. of instructions in decimal)
Dump Memory	DMƀ(starting addr.),(ending addr.)

EPROM PROGRAMMING CRU ASSIGNMENTS

CRU BASE ADDRESS$_{16}$	INPUT/OUTPUT	FUNCTION
1710	I/O	EPROM DATA BIT 0
1712	I/O	:
1714	I/O	:
1716	I/O	:
1718	I/O	:
171A	I/O	:
171C	I/O	:
171E	I/O	EPROM DATA BIT 7
1720	O	EPROM ADDRESS LSB
1722	O	:
1724	O	:
1726	O	:
1728	O	:
172A	O	:
172C	O	:
172E	O	:
1730	O	:
1732	O	:
1734	O	:
1736	O	:
1738	O	EPROM ADDRESS MSB
173A	O	EPROM PROGRAM ENABLE
173E	O	EPROM PROGRAMMING PULSE

EPROM PROGRAMMING RESPONSES

PP = Program EPROM
RE = Read EPROM to Memory
CE = Compare EPROM to Memory
Memory Bounds: MEM BDS? (start addr.),(stop addr.)
EPROM Start addr: EPROM START? (start addr.)
Programming Mode: MODE? P(parallel) or I(in line)
Starting Byte: ST byte ? (0 or 1 if P above)

PREDEFINED CRU ADDRESSES FOR I/O DEVICES

Device	CRU Address
Users Terminal (9902)	80_{16}
Timer (9901)	100_{16}
EIA Interface (9902)	180_{16}
Recorder 1 Forward	1700_{16}
Recorder 2 Forward/9940 Flag 1	1702_{16}
Recorder 2 Write Data/9940 Flag 2	1704_{16}
Recorder 1 Read Data/9940 Flag 3	1706_{16}
Personality Card Code Bit 0	1708_{16}
Personality Card Code Bit 1	$170A_{16}$
Personality Card Code Bit 2	$170C_{16}$
Switch Code Bit	$170E_{16}$
EPROM Data	1710_{16} — $171E_{16}$
EPROM Address	1720_{16} — 1738_{16}
EPROM Program Enable	$173A_{16}$
EPROM Programming Pulse	$173C_{16}$

TXDS Commands for FS 990 Software Development System

Examples of manuals available in support of the TXDS System:

TXDS PROGRAMMER'S GUIDE (#946258-9701)

This manual enables the user to employ the Terminal Executive Development System (TXDS) in conjunction with the TX990 Operating System and the Model 990/4 and 990/10 Computer System hardware configuration to develop, improve, change, or maintain (1) the user's customized Operating System and the user's applications programs or (2) any other type of user-produced programs (e.g., the user's own supervisor call processors or the user's own utility programs). It is assumed the reader is familiar with the Model 990 Computer System assembly language and the concepts of the TX990 Operating System.

The sections and appendixes of this manual are organized as follows:

I Introduction — Provides a general description of the TXDS utility programs and their capabilities. Also includes a description of the control functions of the TXDS Control Program.

II Loading and Executing a Program — Provides a step-by-step procedure for loading and executing (1) each of the TXDS and TX990 Operating System utility programs and (2) a user program. Also describes the TXDS Control Program and how to correctly respond to its prompts.

III Verification of Operation — Provides several short step-by-step procedures to checkout proper operation of the TXDS software.

IV Creating and Editing Program Source Code — Describes the capabilities of the TXEDIT utility program and how the user can employ those capabilities to edit or generate the text of source programs and object programs.

V Assembling Source Programs — Describes how the user can employ the TXMIRA utility program to assemble source files (i.e., source code programs).

VI TX990 Cross Reference (TXXREF) Utility Program — Describes how the user can employ the TXXREF utility program to produce a listing of each user-defined symbol in a 990 assembly source program along with the line numbers on which the symbol is defined and all of the line numbers on which the symbol is referenced.

VII Linking Object Modules — Describes how the user can employ the TXDS Linker utility program to form a single object module from a set of independently assembled object modules (in the form of object code or compressed object code.)

VIII TXDS Copy Concatenate (TXCCAT) Utility Program — Describes how the user can employ the TXCCAT utility program to copy one to three files to a single output file.

IX TXDS Standalone Debug Monitor (TXDBUG) Utility Program — Describes how the user can employ the TXDBUG utility program to debug programs which have been designed to operate in a "standalone" situation without support of an operating system.

X TXDS PROM (TXPROM) Programmer Utility Program — Describes how the user can employ the TXPROM programming utility program to control the Programming Module (PROM) hardware to make customized ROMs containing user-created data or programs.

XI TXDS BNPF/High Low (BNPFHL) Dump Utility Program — Describes how the user can employ the BNPFHL utility program to produce a BNPF or high/low file format.

XII TXDS IBM Diskette Conversion Utility (IBMUTL) Program — Describes how the user can employ the IBMUTL utility program to transfer standard IBM-formatted diskette datasets to TX990 Operating System files and to transfer TX990 Operating System files to standard IBM-formatted diskette datasets.

XIII TXDS Assign and Release LUNO Utility Program — Describes how the operator can assign and release LUNOs in systems which do not include OCP.

A Glossary — Clarifies selected words used in this TX990 Operating System Programmer's Guide.

B Compressed Object Code Format — Describes the compressed object code format.

C Task State Codes — Lists and describes the task state codes.

D I/O Error Codes — List and describes the I/O error codes available to the user, when coding a program, for printout or display on a terminal device.

The following documents contain additional information related to the TX990 Operating System and are referenced herein this manual:

TITLE	PART NUMBER
Model 990 Computer TX990 Operating System Programmer's Guide	946259-9701
Model 990 Computer TMS9900 Microprocessor Assembly Language Programmer's Guide	943441-9701
Model 990 Computer Model FD800 Floppy Disc System Installation and Operation	945253-9701
Model 990 Computer Model 913 CRT Display Terminal Installation and Operation	943457-9701
Model 990 Computer Model 911 Video Display Terminal Installation and Operation	943423-9701
Model 990 Computer Model 733 ASR/KSR Data Terminal Installation and Operation	945259-9701
Model 990 Computer Model 804 Card Reader Installation and Operation	945262-9701
Model 990 Computer Models 306 and 588 Line Printers Installation and Operation	945261-9701
Model 990 Computer PROM Programming Module Installation and Operation	945258-9701
990 Computer Family Systems Handbook	945250-9701
Model 990 Computer Communications Systems Installation and Operation	945409-9701

7

List of Commands and Special Keys/Characters

COMMAND SYNTAX **DESCRIPTION**

SETUP COMMANDS

SL Start Line Numbers (SL) command causes line numbers to be printed with each line of text.

SN Stop Line Numbers (SN) comman causes line numbers not to be printed.

SP Set Print Margin (SP) command sets the right boundary for print display.

SM Set Margin (SM) for Find command sets the left and right boundaries for the Find command.

ST Set Tabs (ST) command sets up to five tab stops.

PRINTER-MOVEMENT COMMANDS

D Down (D) command moves the pointer down toward the bottom of the buffer.

U Up (U) command moves the pointer up towards the first line in the buffer.

T Top (T) command moves the pointer to the first line in the buffer.

B Bottom (B) command moves the pointer to the last line in the buffer.

EDIT COMMANDS

C Change (C) command removes lines from the buffer and inserts new ones in their place. The new lines are input from the terminal.

I Insert (I) command takes input from the terminal and places the new lines into the buffer.

M Move (M) command moves lines from one place in the buffer to another.

R Remove (R) command deletes lines from the buffer.

F Find string (F) command searches for the first occurrence of a character string in a line and replaces it with another string of characters.

PRINT COMMANDS

L Limits (L) command causes the first line and the last line to be displayed.

P Print (P) command displays lines of text.

List of Commands and Special Keys/Characters (Continued)

COMMAND SYNTAX	DESCRIPTION
	OUTPUT COMMANDS
K	Keep (K) command takes lines of text out of the buffer and puts them in the output file.
Q	Quit (Q) command takes lines of text out of the buffer or the input files and puts them in the output file.
E	An (E) command terminates without writing an EOF to the output file.
	TERMINATE-SEQUENCE COMMANDS
T or C	Allows the user to make multiple single directional editing passes on a source or object program.
	SPECIAL KEYS/CHARACTERS
CTRL-H	Pressing the control key and the H key simultaneously on the hard copy terminal causes the terminal to backspace a character to enable rewriting over an entered character-error.
RUB OUT	The RUB OUT key causes the line just entered to be deleted so that a new line can replace it.
CTRL-I	Pressing the control (CTRL) key and the I key simultaneously on a hard-copy terminal causes a tab stop to be entered in the input string, although only one space will be echoed on the terminal.
ESC/RESET	Pressing the ESCape or RESET key on the system console causes a display to be aborted.
position keys	When using a VDT, only the left position key (←) and the right (→) position key are recognized. The up and down position keys cause garbage to be entered into the input string. The left position key causes characters to be deleted from the character string; a right position key causes whatever was under the cursor to be entered.
DELETE LINE	DELETE LINE on a VDT acts the same as a RUB OUT on a hardcopy terminal.
TAB	A SPACE character is echoed. The TAB is interpreted by the text editor and spaces are inserted to fill the text line to the next TAB setting.

TXMIRA Options

OPTION	DESCRIPTION
Mnnnnn	Overrides memory size default; default is 2400 bytes
X	Produce cross-reference
L	Produce assembly listing
T	Expand TEXT code on listing
S	Produce sorted symbol list
C	Produce compressed object output where n is a decimal digit

TXLINK Options

OPTION	DESCRIPTION
Mnnnnn	Override default memory size, default is 11800 bytes.
C	Compressed object output.
Iaaaaaaaa	IDT for linked object.
P	Partial link desired.
L	Print load map and symbol list.

Note: n is a decimal digit and a is an alphanumeric character.

TXCCAT Options

OPTION	DESCRIPTION
TRnnnn	Truncate record to length nnnn.
FLnnnn	Fix records to size nnnn by padding with blanks or by truncation.
SKnnnn	Skip nnnn input records, prior to output.
LFnn	List file, page length = nn, default = 55.
SLnn	Space lines on listing, nn = space count, default = 0.
NL	Number lines on listing.
RI	Do not rewind input on open.
RO	Do not rewind output on open.

Note: n is a decimal digit and the maximum field size is given by the number of n's.

TXDBUG Keyboard Commands

DEBUG Commands

IC	Inspect Communications Register Unit (CRU)
IM	Inspect Memory
IR	Inspect AU Register (WP, PC, ST)
IS	Inspect Snapshot
IW	Inspect Workspace Registers
MC	Modify Communications Register Unit (CRU)
MM	Modify Memory
MR	Modify Registers
MW	Modify Workspace Registers
SB	Set Breakpoint
SP	Set H/W Write Protect Option
SR	Set Trace Region
SS	Set Snapshot
ST	Set Trace
CB	Clear Breakpoint
CP	Clear H/W Write Protect Option
CR	Clear Trace Region
CS	Clear Snapshot

AMPL Reference Data

▶ 7

EXPLANATION OF THE NOTATION USED IN THIS CARD

	Notation	Explanation
Optional Items	[item]	Bracketed item may be omitted.
	{item 1 item 2}	Exactly one item must be selected. from the items in braces.
Substitution	expr	Any expression may be used.
	'file'	File or device name required.
Repetition	item . . .	A list of items may be used.
Required	<item>	Replace with item.

CHARACTER SET

Type	Characters	Use
Special	RETURN SPACE ! " $ / ( ) * + , — . / : ; < = > ? @	Any printable character may be used in a quoted string. RETURN terminates line and statement. ";" may separate statements. SPACE separates adjacent numbers and identifiers.
Numerals	0 — 9	
Letters	A — Z,a — z	

NOTE: All AMPL reserved words use only upper case (UPPER CASE LOCK).

SYMBOL NAMES

Type	Example	Definition
System	RO ETRC	Up to four alphanumeric characters; all system symbols are predefined.
User-defined	USRVAR X3 BRKADR GO	Up to six alphanumeric characters; assignment defines a variable. ARRAY statement defines an array. PROC/FUNC statement defines a procedure/function.
Program label	IDT. .DEF	Up to six alphanumeric characters. Period after IDT and before DEF labels, defined by LOAD command.

CONSTANTS

Type	Example	Range
Decimal	10833	1 . . . 32767
Hexadecimal	02A51, >2A51	>0 . . . >FFFF
Octal	125121	!0 . . . !177777
Binary	<10101001010001	<0 . . . <1111111111111111
ASCII	"*Q"	
Instruction	#XOR *R1,R9#	
Keyword	IAQ	See keyword constant table.

EXPRESSIONS

Type	Example	Definition
Subexpression	(expr)	
Identity	+ expr	Value of <expr>.
Negation	– expr	Two's complement of <expr>.
Target memory	@addr	<addr> used as word address into emulator or target memory.
Proc/Func Argument	ARG expr	Argument in position <expr> of call list; ARG 0 is number of arguments in list.
Proc/Func local variable	LOC expr	Word <expr> of local variable array; LOC 0 is length of local variable array.
Multiplication	expr1*expr2	Signed product (warning on overflow).
Division	expr1/expr2	Signed quotient (warning on divide by zero).
Remainder	expr1 MOD EXPR2	Signed remainder of division (warning on divide by zero).
Addition	expr1 + expr2	Signed sum.
Subtraction	expr1 - expr2	Signed difference.
NOTE: Result of relational operator is either FALSE (0) or TRUE (-1).		
Equality	expr1 EQ expr2 expr1 NE expr2	16-bit comparison.
Arithmetic inequality	expr1 LT expr2 expr1 LE expr2 expr1 GT expr2 expr1 GE expr2	Signed, 16-bit comparison.
Logical inequality	expr1 LO expr2 expr1 LOE expr2 expr1 HI expr2 expr1 HIE expr2	Unsigned, 16-bit comparison.
Complement	NOT expr	16-bit one's complement.
Conjunction	expr1 AND expr2 expr 1 NAND expr2	16-bit boolean AND. 16-bit boolean not AND.
Disjunction	expr1 OR expr2 expr1 XOR expr2	16-bit boolean OR. 16-bit boolean exclusive OR.

NOTE: Operators are given in order of precedence, highest to lowest. Solid lines separate precedence groups; within each group, precedence is equal and evaluation is left to right. Evaluation results in a 16-bit integer value.

UNSIGNED ARITHMETIC

Syntax	Definition
MPY (expr1, expr2)	Low-order 16 bits of unsigned product. <expr1>* <expr2>; high order 16 in MDR.
DIV (divisor, dividend)	Unsigned quotient of 32-bit number (MDR, <dividend>) over <divisor>; remainder in MDR.
MDR	High-order 16-bits of MPY product and of DIV dividend; remainder of DIV; unsigned carry of + and-.

ARRAY DEFINITION

ARRAY name(expr1[,expr2]), . . .	User <name> (previously undefined or name of deleted array) is defined as one- or two-dimension array.

DISPLAY STATEMENTS

expr[:f . . . f]	Value of expression
'LITERAL STRING'	Literal string
add1 [TO addr2] [:f . . . f] ? [:f . . . f]	Target memory

Format specification/[:f . . . f]

ASCII	A	set default	G	octal	O[i]
binary	B[i]	hexadecimal	H[i]	symbolic	S
decimal	D[i]	instruction	I	unsigned	U[i]
name =	E	newline	N[j]	space	X[j]

Note:	1< =i< =9	field width 'i' digits, then two blanks
	i=0	default field width, no trailing blanks
	1< =j< =9	repeat 'j' times
	j=0	repeat 10 times

Response to display/modify mode(?):

forward step	RETURN, +	replace contents	<expr>
back step	—	open new address	@<addr>
exit	;	change display	:f . . . f

DISASSEMBLER

Instruction	DST	Destination address.
operands	SRC	Source address.

NOTE: Additional instructions of the TMS9940 (DCA, DCS, LIIM, SM) will assemble correctly (#DCA *RC1 #) but will disassemble as XOP instructions. See TMS9940 specifications for details.

ASSIGNMENT STATEMENTS

Type	Example	Definition
Variable	sym = expr	User-defined or writable system symbol or REF program label.
Target memory	@addr = expr	Put value of <expr> at target <addr>
Proc/Func argument	ARG n = expr	Local copy of argument in position <n> of call list.
Command local	LOC n = expr	Word <n> of local storage array.
Array	A[(i1[,i2])] = e	User defined array name; zero, one, or two index expressions.

NOTE: Precedence of @, ARG, and LOC may require parenthesis around following expression.

COMPOUND STATEMENTS

Syntax	Definition
BEGIN statements END	Statements are executed sequentially. Use in place of any single statement syntax.

CONTROL STATEMENTS

IF expr
THEN s1
[ELSE s2]
	<s1> is executed if <expr> is TRUE (nonzero). Otherwise, <s2> is executed, if included.

CASE expr OF expr 1::s1; . . . . exprn::sn [ELSE s] END	Statement <si> at first label expression <expr> equal to <expr> is executed. If none, statement <s> is executed, if included.
WHILE expr DO statement	While <expr> is TRUE (nonzero), <statement> is executed.
REPEAT statement UNTIL expr	<statement> is executed. If <expr> FALSE (zero), <statement> is executed until <expr> is TRUE.
FOR var = expr1 TO expr 2[BY expr3] DO statement	Value of <expr1> is assigned to <var>. <statement> is executed until <var> is equal to <expr2>; <expr3> is added to <var>, and <statement> repeated. Default value of <exp3> is 1.
ESCAPE	Exit from innermost enclosing WHILE, REPEAT, or FOR statement.

PROCEDURE/FUNCTION/FORM DEFINITION

PROC name [(args[,locs])] statements END
FUNC name [(args[,locs])] statements END

User-defined <name> (previously undefined or deleted procedure/function) is bound to <statements>.
<args> is the required number of arguments.
<locs> is the size of local storage array.

RETURN [expr]

Pass control back to calling statement. In a procedure, <expr> is ignored. In a function, value of <expr> replaces the function call in the calling expression.

FORM name 'prompt' [= [$\left\{\begin{array}{l}\text{constant}\\ \text{'string'}\end{array}\right\}$]]; . . . END

<name> must be a previously defined procedure or function, semicolon required between prompts.

PROCEDURE/FUNCTION CALLS

proc name [(expr, . . .)]

User-defined or system procedure/function with list of argument expressions.

func name [(expr, . . .)]

Command definition determines number of arguments required. Some system commands require quoted strings as arguments.

NOTE: Procedure/functions with defined FORM when called with no arguments will prompt for arguments using the FORM.

example FORM:

COMMENTARY ENTRY

PROMPT 1 = default value
PROMPT 2 =
PROMPT 3* =

comment, not a prompt required argument, with default value required argument, must enter value default given if value not entered

FORM control function keys:

Next prompt:	TAB,↓,→FIELD, SKIP, RETURN
Previous prompt:	↓,←FIELD
First prompt:	HOME
Erase value:	ERASE FIELD, ERASE INPUT
Redisplay default:	INSERT LINE
Duplicate previous value:	F4
Complete form:	ENTER
Abort form:	CMD

INPUT/OUTPUT COMMANDS

```
Syntax                              Definition
HCRB                                Host computer CRU base address.
HCRR (offset,width)                 Read host computer CRU field.
HCRW (offset,width,value)           Write <value> into host CRU field.

       ( ( 'file'  ) )              AMPL input from 'file'
COPY   ( { edit id } )              AMPL input from edit buffer

       ( ( 'file' ) )               Initialize listing device or file. Disable listing output.
LIST   ( { OFF    } )               Enable listing output. Close listing device or file with
       ( { ON     } )               EOF.
       ( ( EOF    ) )

NL                                  Print newline.
```

```
unit = OPEN [ ( { 'file'  } [ , { 0   } [ , { 0      } [ , { SEQ } ] ] ] ) ]
                { edit id }      { IN  }      { REWIND }      { REL }
                                 { OUT }      { EXTEND }
                                 { IO  }
```

no arguments — list all open units and edit buffers.
initialize 'file'/<edit id> I/O unit
  0 — device IO, file IN only
  IN — for input only
  OUT — for output only
  IO — for input/output
  REWIND — position to beginning of file
  EXTEND — position to end of file
  SEQ — auto-create sequential file
  REL — auto-create rel-rec file

▶7

```
event-READ [ ( unit [ , { 0      } [ , { 0     }
                        { DIRECT }      { GRAPH }

        [ , { VDT , { 0     } [ , { 0     } [ , s col ] ] } ] ] ] ) ]
            { SEQ   { f row }     { f col }               }
            { REL [ , rec # ]                              }
```

no arguments — read console
Read record from (unit)
  0 — issue read ASCII
  DIRECT — issue read direct
  GRAPH — read graphics on 922 VDT
  VDT — read in cursor positioning mode
    f row — field start row
    f col — field start column
    s col — cursor start column

INPUT/OUTPUT COMMANDS (continued)

SEQ — read sequentially
REL — read sepecified record
rec # — record number to read
<event>/256 = cursor column after read if VDT
<event> AND 255 = event key value if VDT,
else >OD for end of record,
>13 for end of file.

value = EVAL [(unit)]
Evaluate expression in <unit>'s buffer;
if no <unit>, READ/EVAL the console.

DPLY [(unit)]
AMPL display unit for output to <unit>;
if no <unit>, to console.

okay = MOVE (from unit, to unit)
Move contents of <from unit>'s buffer to <to unit>'s buffer
<okay> = 0 if moved
= >FFFF if too big and not moved.

REW[(unit)]
Rewind (unit) — repositions, file clears console
no argument — clears console

$$\text{Cursor} = \text{WRIT}\left(\text{unit}\left[,\begin{Bmatrix}0\\ \text{DIRECT}\end{Bmatrix}\left[,\begin{Bmatrix}0\\ \text{GRAPH}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\text{VDT} & \begin{Bmatrix}0\\ \text{f row}\end{Bmatrix} & [,[\text{f col}]]\\ \text{SEQ} & , & \\ \text{REL} & [,\ \text{rec}\ \#] & \end{Bmatrix}\right]\right]\right)$$

no arguments — write console
Write record to (unit),
0 — issue write ASCII
DIRECT — issue write direct
GRAPH — write graphics on 911 VDT
VDT — write in cursor positioning mode
f row — field start row
f col — field start column
SEQ — write sequentially
REL — read specified record
rec # — record number to read
<cursor>/256 = cursor column after write if VDT

$$\text{CLSE (unit}\left[,\begin{Bmatrix}\text{EOF}\\ \text{UNLOAD}\end{Bmatrix}\right]$$

Release I/O <unit>,
EOF — write end-of-file mark
UNLOAD — unload unit

SYSTEM SYMBOLS

V — variable F — function P — procedure

CLR	P	— clear
CLSE	P	— I/O close
COPY	P	— copy
CRUB	V	— CRU base
CRUR	F	— CRU read
CRUW	P	— CRU write
DAY	V	— day
DBUF	P	— delete buffer
DELE	P	— delete symbol
DIV	F	— divide
DPLY	P	— display
DR	P	— registers
DST	V	— destination
DUMP	P	— dump
EBRK	P	— emulator
ECLK	V	— emulator
EDIT	F	— edit
EHLT	F	— emulator
EINT	P	— emulator
EMEM	V	— emulator
ERUN	P	— emulator
EST	F	— emulator
ETB	F	— emulator
ETBH	F	— emulator
ETBO	V	— emulator
ETRC	P	— emulator
ETYP	V	— emulator
EVAL	F	— evaluate
EXIT	P	— exit AMPL
HCRB	V	— host CRU
HCRR	F	— CRU read
HCRW	P	— CRU write
HR	V	— hour
IOR1	V	— I/O
KEEP	P	— keep edit
LIST	P	— list
LOAD	P	— load object
MDEL	P	— symbols
MDR	V	— arithmetic
MIN	V	— minutes
MOVE	F	— I/O buffer
MPY	F	— multiply
MSYM	P	— symbols
NL	P	— newline
OPEN	F	— I/O open
PC	V	— registers
R0-R15	V	— registers
READ	F	— I/O read
REW	P	— I/O rewind
RSTR	P	— restore
SAVE	P	— save
SEC	V	— seconds
SRC	V	— source
ST	V	— register
TBRK	P	— trace module
TEVT	P	— trace module
THLT	F	— trace module
TINT	P	— trace module
TNCE	V	— trace module
TNE	V	— trace module
TRUN	P	— trace module
TST	F	— trace module
TTB	F	— trace module
TTBH	F	— trace module
TTBN	V	— trace module
TTBO	V	— trace module
TTRC	P	— trace module
USYM	P	— user symbols
VRFY	P	— verify
WAIT	F	— delay AMPL
WP	V	+ register
WRIT	P	— I/O write
YR	V	— year

▶7

EDIT

Syntax	Definition
edit id = EDIT[($\left\{ \begin{array}{l} \text{'file'} \\ \text{edit id} \end{array} \right\}$ [,record])]	Create edit buffer with 'file'. Edit existing buffer. No argument creates an empty buffer.
KEEP (edit id, 'file')	Save edit buffer onto 'file' and delete edit buffer.
DBUF (edit id)	Delete edit buffer.

EDIT CONTROL FUNCTION KEYS

Function	911 KEY	913 KEY	CONTROL CHARACTER
edit/compose mode	F7	F7	V
quit edit mode	CMD	HELP	X
roll up	F1	F1	A
roll down	F2	F2	B
set tab	F3	F3	C
clear tab	F4	F4	D
tab	TAB (shift SKIP)	TAB	I
back tab	FIELD	BACK TAB	T
newline	RETURN	NEWLINE	RETURN
insert line	unlabeled gray	INSERT LINE	O
delete line	ERASE INPUT	DELETE LINE	N
erase line	ERASE FIELD	CLEAR	W
truncate line	SKIP	SET	K
insert character	INS CHAR	INSERT CHAR	
delete character	DEL CHAR	DELETE CHAR	
cursor up	↓	↓	U
cursor down	↓	↓	J
cursor right	→	→	R
cursor left	←	←	H
top of screen	HOME	HOME	

GENERAL COMMANDS

Syntax	Definition
USYM	List all user symbols, procedures, functions, and arrays.
DELE ('name'....)	Delete user procedure, function, or array.
SAVE ('file')	Save all user defined symbols, functions, and arrays on 'file'.
RSTR ('file')	Restore user defined symbols, procedures, functions, and arrays from 'file'.
CLR	Delete all user symbols, procedures, functions and arrays.
MSYM	List object program labels.
MDEL	Delete all object program labels.
EXIT	Exit from AMPL back to operating system.

TIMING

YR	Year (1976 to 1999)
DAY	Julian day (1 to 366)
HR	Hour (0 to 23)
MIN	Minute (0 to 59)
SEC	Second (0 to 59)
WAIT (expr)	Suspend AMPL for <expr>*50 milliseconds (<expr> = 20 is one second).

TARGET MEMORY COMMANDS

▶7

EMEM	Emulator memory mapping: 9900/9980 map 8K bytes (0->1FFF) 9940 define RAM and ROM sizes.
LOAD ('file'[,bias[,IDT] [+ DEF] [+ REF]]]):	Load object program by bias and enter program labels into table.
VRFY ('file' [,bias])	Verify object program, listing differences between object and target memory.
DUMP ('file',low,high[,start])	Dump program from target <low> to <high> in nonrelocatable format.

EMULATOR CONTROL COMMANDS

Syntax	Definition
EINT ('EM0n' [, $\left\{\begin{matrix}1\\0\end{matrix}\right\}$ [,'TM0n']])	Initialize Emulator device, clock 0 = prototype/ 1 = emulator.
ECLK	Processor clock.
ETYP	Processor type: -1 = TMS9940, 0 = SBP9900, 1 = TMS9900, 2 = TMS9980.
ETRC ($\left\{\begin{matrix}\text{MA}\\\text{IAQX}\\\text{IAQ}\end{matrix}\right\}$ [,count[,low,high]])	Trace qualifier, completion break count (OFF-255), address range.
EBRK ($\left\{\begin{matrix}\text{MA}\\\text{IAQ}\\\text{MR}\\\text{MW}\end{matrix}\right\}$ [+ ILLA] [,address]...)	Address breakpoint(s) (ILLA only valid for TMS9940).
ERUN	Run emulation at PC, WP, ST.
EST	Emulation status (3 LSBits): HOLD, IDLE, Running
EHLT	Halt emulation, return status.
ETBH (index[, $\left\{\begin{matrix}\text{MR}\\\text{MW}\\\text{IAQ}\end{matrix}\right\}$])	Indexed bus signal from buffer. (TRUE if expression matches).
ETB (index)	Indexed address from trace buffer.
ETBO, ETBN	Emulator Trace buffer limits: Oldest, Newest sample indices.

TRACE MODULE CONTROL

Syntax **Definition**

TINT ('TM0n') Initialize trace module

```
                  (       OFF                          )
TTRC   ([INT] < [±Q0] [±Q1  ][±Q2  ][±Q3] > [,count[, ( ON  ) ]])
                  (         [±IAQ][±DBIN]              )           ( OFF )
```

Qualify data samples, trace completion counter (OFF-255), latch option on D0-D3.

```
            (           OFF              )
TEVT   (  ( [±D0]  [±D1] [±D2 ] [±D3] ) [,value[,mask]])
            (       [±IAQ] [±DBIN)       )
            (           EXT              )
```

Qualify D0-D3 event (or EXTernal), <value> and <mask> for D4-D19.

TBRK (count [,<delay>[,INV] [+EDGE]]])

Set event counter (OFF-FFFF), set delay counter (OFF-244), count INVerted/EDGE events.

TRUN Start Trace module tracing.

TST Trace module status (3 LSB's), event occurred, trace full, tracing.

THLT Halt trace module, return status.

TNE Number of events since last TRUN.

TNCE Number of event count overflows.

```
TTBH (index[, ( [±D0] [ ±D1  ][±D2   ]  [±D3] ) ])
              (        [±IAQ][±DBIN]          )
```

D0-D3 of indexed samples, (TRUE if expression matches).

TTB (<index>) D4-D19 indexed samples (data bus)

▶7 TTBO, TTBN Trace module trace buffer limits: Oldest, Newest sample indices.

TRACE MODULE INTERCONNECT TO EMULATOR

Q0	Memory address bit 15 (TMS9940 only).
D0	Byte memory cycle (TMS9940 only).
Q1,D1, IAQ	Instruction Acquisition.
Q2,D2,DBIN	DataBusIN = MR(read), MW = -DBIN(write).
Q3	Emulator trace qualifier and range (ETRC).
D3, External Event	Emulator address breakpoint (EBRK).
D4-D19	Emulator data bus (bits 0-15).
External Clock	Emulator memory cycle clock.
Control Cable	Synchronizes emulation and tracing. Trace module will halt emulator for EINT ('EM0n', clock 'TM0n').

TARGET REGISTERS

PC,WP,ST	Processor registers.
R0-R15	Workspace registers.
DR	Display all registers.

CRU READ/WRITE

CRUB	CRU interface base address.
CRUR (offset,width)	Read target CRU field.
CRUW (offset,width,value);	Write <value> into target CRU field

KEYWORDS

ARG
ARRAY
BEGIN
BY
CASE
DO
ELSE
END
ESCAPE
FOR
FORM
FUNC
IF
LOC
MOD
NULL
OF
PROC
REPEAT
RETURN
THEN
TO
UNTIL
WHILE
AND
NAND
OR
XOR
NOT
EQ
GE
GT
HI
HIE
LE
LO
LOE
LT
NE

KEYWORD CONSTANTS

D0
D1
D2
D3
DBIN
DEF
DIRECT
EDGE
EOF
ETBN
EXT
EXTEND
GRAPH
IAQ
IAQX
IDT
ILLA
IN
INT
INV
IO
MA
MR
MW
N
OFF
ON
OUT
Q0
Q1
Q2
Q3
REF
REL
REWIND
SEQ
UNLOAD
VDT
Y

7◄

ERROR MESSAGES

0 — ! UNDEFINED ERROR CODE !
1 — I/O ERROR, OS ERROR CODE RETURNED
2 — INSUFFICIENT MEMORY TO CONTINUE
3 — ! SEGMENT VIOLATION !
4 — I/O ERROR: INVALID UNIT ID
5 — I/O ERROR: READ/WRITE VIOLATION
6 — I/O ERROR: INSUFFICIENT MEMORY FOR OPEN
7 — ! DELETE UNIT CONTROL BLOCKS ERROR !
8 — TOO MANY IDT DEF/REF SYMBOLS IN LOAD
9 — EXCEEDED 15 LOAD OPERATIONS SINCE LAST CLR
10 — CANNOT ALLOCATE MEMORY FOR USER SYMBOL TABLE
11 — ! ERROR IN I/O UNIT CHAIN POINTERS !
12 — OVERLAY ERROR
101 — VARIABLE CANNOT BE READ
102 — VARIABLE CANNOT BE WRITTEN
103 — SYMBOL IS UNDEFINED
104 — ! INVALID CODEGEN BRANCH TABLE INDEX !
105 — INSUFFICIENT MEMORY TO COMPILE STATEMENT
106 — SYMBOL IS DEFINED; CANNOT BE REDEFINED
107 — INSUFFICIENT MEMORY TO COMPILE PROC/FUNC
108 — INPUT RECORD CANNOT BE CLASSIFIED
109 — INPUT STRING EXCEEDS MAXIMUM ALLOWED LENGTH
110 — ! INVALID SCANNER BRANCH TABLE INDEX !
111 — UNRECOGNIZABLE INPUT ITEM
112 — ! UNDEFINED OPERATOR !
114 — SYMBOL NOT AN IDT/DEF/REF LOAD SYMBOL
115 — USER SYMBOL TABLE FULL
116 — CONSTANT EXCEEDS 16 BITS
117 — SYNTAX ERROR
118 — ! INVALID KEYWORD STRING LENGTH !
119 — SYNTAX ERROR IN ONE-LINE-ASSEMBLY STATEMENT
120 — INCORRECT NUMBER OF ARRAY SUBSCRIPTS
121 — ESCAPE SPECIFIED OUTSIDE A LOOP CONSTRUCT
122 — ARRAY REDEFINED WITH INCORRECT SUBSCRIPTS

NOTE: A hexadecimal number is also printed with some error messages. Refer to the AMPL System Operation Guide for complete explanation.

▶7

ERROR MESSAGES

201 — SYMBOL NOT FOUND TO DELETE
202 — SYMBOL CANNOT BE DELETED
203 — INVALID DISPLAY FORMAT CHARACTER FOLLOWING:
204 — NO LIST DEVICE ASSIGNED
205 — EMULATOR I/O ERROR CODE RETURNED
209 — INVALID INDEX INTO EMULATOR TRACE BUFFER
210 — !CANNOT ALLOCATE FORM CURRENT VALUE SEGMENT!
211 — INSUFFICIENT MEMORY TO SAVE FORM PARAMETERS
214 — INVALID RESTORE FILE
215 — INSUFFICIENT MEMORY TO COMPLETE THE RESTORE
216 — BAD TRACE OR COMPARISON MODE SELECTED
219 — TRACE MODULE I/O ERROR CODE RETURNED
220 — CANNOT EDIT ON THIS DEVICE TYPE
221 — TRACE INTERFACE CHANGE ILLEGAL WHILE TRACING
222 — INVALID INDEX INTO TRACE MODULE BUFFER
223 — INSUFFICIENT ARGUMENTS IN PROC/FUNC CALL
224 — STACK OVERFLOW; DELETE PROC/FUNC/ARRAY
225 — DELETED PROC/FUNC/ARRAY REFERENCED
226 — INSUFFICIENT ARGUMENTS IN FORM FOR PROC/FUNC
227 — ! INVALID FORM SEGMENT ID !
228 — ! INVALID FORM CURRENT VALUE SEGMENT ID !
229 — INVALID CHARACTER IN LOAD FILE
230 — CHECKSUM ERROR IN LOAD FILE
231 — ARITHMETIC OVERFLOW
233 — PROC/FUNC CALL ARGUMENT OUT OF RANGE
234 — INVALID "ARG" OR "LOC" INDEX FOR WRITING
235 — INVALID "ARG" OR "LOC" INDEX FOR READING
237 — ARRAY ALREADY DEFINED
238 — INVALID ARRAY DIMENSION
240 — REFERENCE TO UNDECLARED ARRAY
241 — INVALID ARRAY SUBSCRIPT
242 — ! ERROR ARRAY SEGMENT LENGTH !
243 — DELETED IDT/DEF/REF LOAD SYMBOL REFERENCED
244 — ALL IDT/DEF/REF LOAD SYMBOLS DELETED
245 — INVALID DEVICE TYPE TO "EINT" OR "TINT"

NOTE: Error messages withing exclamation marks (!) are AMPL internal system errors. Contact Texas Instruments if problem persists.

POWER BASIC
MP 307

▶7

REFERENCE CARD FOR DEVELOPMENT AND EVALUATION BASIC

This card contains a summary of all POWER BASIC† statements and commands for Development and Evaluation BASIC. An explanation preceded by an asterisk (*) indicates the statement or command is not supported by Evaluation BASIC. A ⋆ indicates the statement is supported only by the Development BASIC software enhancement package.

COMMANDS

CONtinue

*Execution continues from last break.

LIST

LIST the user's POWER BASIC program. In LIST will list from specified line number through end of program or until ESC key is struck.

LOAD

Reads a previously recorded POWER BASIC program from an auxiliary device or configures POWER BASIC to execute a BASIC program in EPROM.
LOAD reads program from 733ASR digital cassette.
LOAD 1 or LOAD 2 ⋆ reads program from audio cassette drive No. 1 or No. 2.
LOAD <address>⋆ configures POWER BASIC to execute BASIC program in EPROM at specified address.

NEW

Prepare for entry of NEW POWER BASIC program or set the lower RAM memory bound after auto-sizing.
NEW clears pointers of POWER BASIC and prepares for entry of new program.
NEW <address>* sets the lower RAM memory bound used by POWER BASIC after auto-sizing or power-up.

PROGRAM

Program current POWER BASIC application program into EPROM.⋆

7◀

RUN

Begin program execution at the lowest line number.

SAVEn (n is interpreted as in LOADn command)

Record current user program on auxiliary device.

SIZE

Display current program size, variable space allocated, and available memory in bytes.

†*Trademark of Texas Instruments*

EDITING

The phrase "(ctrl)" indicates that the user holds down the control key while depressing the key corresponding to the character immediately following.

(CR)	Enter edited line.
(ctrl)In	*Insert n blanks.
(ctrl)Dn	*Delete n characters.
(ctrl)H	Backspace one character.
(ctrl)F	Forward space one character.
In(ctrl)E	Display for editing source line indicated by line number (In).
(ctrl)T	Toggle from one partition to the other partition (only in Evaluation BASIC).
(esc)	Cancel input line or break program execution.
(Rubout) or (DEL)	Backspace and delete character.

STATEMENTS

InBAUD <exp 1,> <exp 2>

*sets baud rate of serial I/O port(s).

InBASE <(exp)>

Sets CRU base address for subsequent CRU operations

InCALL Name<subroutine address>[, <var 1>, <var 2>, <var 3>, <var 4>]

*Transfers to external subroutines. If variable is contained in parentheses, the address will be passed; otherwise, the value will be passed.

$$\text{InDATA} \left\{ \begin{matrix} \text{<exp>} \\ \text{<string const>} \end{matrix} \right\} \left[, \left\{ \begin{matrix} \text{<exp>} \\ \text{<string const>} \end{matrix} \right\} \right] \ldots$$

defines internal data block.

▶7

In DEF FN<x>[(<arg 1> [, arg 2] [, arg 3])] = <exp>

*Defines user arithmetic function.

InDIM <var (dim[, dim] . . .)> [, ...]

Allocates user variable space for dimensioned or array variables.

InEND

Terminates program execution and returns to edit mode.

In ERROR<In>

*Specifies a subroutine that will be called via a GOSUB statement when an error occurs.

In ESCAPE
InNOESC

*Enables or disables the excape key to interrupt program execution (always enabled in Evaluation BASIC).

lnFOR <sim-var> = <exp> TO <exp> [STEP <exp>]
lnNEXT <sim-var>

Open and close program loop. Both identify the same control variable. FOR assigns starting, ending, and optionally stepping values.

lnGOSUB<ln>

Transfer of control to an internal subroutine beginning at the specified line.

lnPOP

*Removal of most previous return address from GOSUB stack without an execution transfer.

lnRETURN

Return from internal subroutine.

lnGOTO<ln>

Transfers program execution to specified line number.

lnIF<exp>THEN<statement>
lnELSE<statement>

Causes conditional execution of the statement following THEN. *ELSE statements execute when IF condition is false.

lnIMASK<LEVEL>

*Set interrupt mask of TMS 9900 processor to specified level.

lnTRAP<level>TO<ln>

*Assign interrupt level to interrupt subroutine.

lnIRTN

*Return from BASIC interrupt service routine.

lnINPUT<var> $\left[\begin{Bmatrix} , \\ ; \end{Bmatrix} \text{<var>}\right] \cdots \left[\begin{Bmatrix} , \\ ; \end{Bmatrix}\right]$

Accesses numeric constants and strings from the keyboard into variables in the INPUT list.

7◀

ln [LET] <var> = <exp>

Evaluates and assigns values to variables or array elements.

lnON $\begin{Bmatrix} \text{<var>} \\ \text{<exp>} \end{Bmatrix}$ THEN GOTO ln [,ln] . . .

lnON $\begin{Bmatrix} \text{<var>} \\ \text{<exp>} \end{Bmatrix}$ THEN GOSUB ln [,ln] . . .

*Transfers execution to the line number specified by the expression or variable.

lnPRINT <exp> [,exp] . . .

Print (format free) the evaluated expressions.

lnRANDOM [exp]

*Set the seed to the specified expression value.

lnREAD $\begin{Bmatrix} \text{<numeric var>} \\ \text{<string var>} \end{Bmatrix} \left[, \begin{Bmatrix} \text{<numeric var>} \\ \text{<string var>} \end{Bmatrix} \right] \ldots$

Assigns values from the internal data list to variables or array elements.

InREM [text]

Inserts comments.

InRESTOR [exp]

Without an argument, resets pointer to beginning of data sequence; with an argument, resets pointer to line number specified.

InSTOP

Terminates program execution and returns to Edit mode.

InTIME

Sets, displays, or stores the 24 hour time of day clock.

InTIME <exp>, <exp>, <exp>
Sets and starts clock.
InTIME <string-var>
Enables storing clock time into a string variable.
InTIME
Prints clock time as HR:MN:SD.

InUNIT <exp>

*Designates device(s) to receive all printed output.

FUNCTIONS

ABS <(exp)>
*Absolute value of expression.

ASC <(string var)>
*Returns decimal ASCII code for first character of string variable.

ATN <(exp)>
Arctangent of expression in radians.

BIT <(var, exp)>
BIT <(var, exp 1)> = <exp 2>
*Reads or modifies any bit within a variable.
Returns a 1 if bit is set and 0 if not set.
Selected bit is set to 1 if assigned value is non-zero and to zero if the assigned value is zero.

▶7

COS >(exp)>
Cosine of the expression in radians.

CRB <(exp)>
Reads CRU bit as selected by CRU base + exp. Exp is valid for −127 thru 128.

CRB <(exp 1)> = <(exp 2)>
Sets or resets CRU bit as selected by CRU base + exp 1. If exp 2 is non-zero, the bit will be set, else reset. Exp 1 is valid for −127 thru 128.

CRF <(exp)>
Reads n CRU bits as selected by CRU base where exp evaluates to n. Exp is valid for 0 thru 15. If exp = 0, 16 bits will be read.

CRF <(exp 1)> = <(exp 2)>
Stores exp 1 bits of exp 2 to CRU lines as selected by CRU BASE. Exp 1 if valid for 0 thru 15. If exp 1 = 0, 16 bits will be stored.

EXP <(exp)>
*Raise the constant e to the power of the evaluated expression.

INP <(exp)>
Returns the signed integer portion of the expression.

LOG <(exp)>	*Returns natural logarithm of the expression.
MEM <(exp)>	Reads byte from user memory at address specified by exp. Exp must be in the integer range, (0 to 65535).
MEM <(exp 1)> = <(exp 2)>	Stores byte exp 2 into user memory specified by exp 1. Exp 1 and exp 2 must be in the integer range.
MCH <(string 1), (string 2)>	*Returns the number of characters to which the two strings agree.
NYK <(exp)>	Conditionally samples the keyboard in run time mode. If exp < >0, return decimal value of last key struck and clear key register. (0 if no key struck.) If exp = 0, return a 1 if the last key struck has the same decimal value as the expression. Clear key register if TRUE, else return 0 if FALSE.
RND	Returns a random number between 0 and 1.
SIN <(exp)>	Sine of the expression in radians.
SQR <(exp)>	Square root of expression.
SRH <(string 1), (string 2)>	*Refurn the position of string 1 in string 2, 0 if not found.
SYS <(exp)>	*Obtains system parameters generated during program execution. Example: SYS(0) = INPUT control character, SYS(1) = Error code number, SYS(2) = error line number.
TIC <(exp)>	Returns the number of time tics less the expression value. One TIC equals 40 milliseconds (1/25 second).

STRINGS

ASCII Character	ASC (string-var)
Conversion Function	*Convert first character of string to ASCII numeric representation.
Assignment	$\text{<string-var>} = \left\{\begin{matrix}\text{<string-var>}\\ \text{<string-constant>}\end{matrix}\right\}$ Store string into string-var ending with a null.
Character Match Function	MCH (<string 1>, <string 2>) *Return the number of characters to which the 2 strings agree.
Character Search Function	SRH (<string 1>, <string 2>) *Return the position of string 1 in string 2. Zero is returned if not found.
Concatenate	$\text{<string-var>} =$ $\left\{\begin{matrix}\text{<string-var>}\\ \text{<string-constant>}\end{matrix}\right\} + \left\{\begin{matrix}\text{<string-var>}\\ \text{<string-constant>}\end{matrix}\right\} \left[+ \left\{ \cdots \right\} \right]$

Convert to ASCII

<string-var> = <exp>
<string-var> = # <string>, <exp>

*Convert exp to ASCII characters ending with a null. # string specifies a formatted conversion.

Convert to Binary

<var 1> = <string>, <var 2>

*Convert string into binary equivalent. Var 2 receives the delimiting non-numeric character in first byte.

Deletion

<String-var> = /<exp>

*Delete exp characters from string-var.

Insertion

<string-var> = /<string>

*Pick byte into string-var.

Pick

$<\text{string-var}> = \left\{\begin{matrix} <\text{string-var}> \\ <\text{string-constant}> \end{matrix}\right\}, <\text{exp}>$

Pick number of characters specified by exp into string-var ending with a null.

Replace

$<\text{string-var}> = \left\{\begin{matrix} <\text{string-var}> \\ <\text{string-constant}> \end{matrix}\right\}; <\text{exp}>$

Replace number of characters specified by exp of string-var with string.

String Length
Function

<var> = LEN <(string-var)>
<var> = LEN "string"

*Return the length of string.

INPUT OPTIONS

string-var

Prompt with colon and input character data.
Example: INPUT $A

▶7

,

Delimit expressions. Example A, B

;

Suppress prompting or CR LF if at end of line.
Examples: INPUT ;A
INPUT A;

# exp

Allow a maximum of exp characters to be entered.
Example: INPUT #1"Y or N"$1

%exp

*Must enter exactly exp number of characters.
Example: INPUT %4"CODE"C

?<ln>

*Upon an invalid input or entry of a control character, a GOSUB is performed to the line #. SYS(0) will be equal to −1 if there was an invalid input. Otherwise, SYS(0) will equal the decimal equivalent of the control character.
Example: INPUT ?100;A

OUTPUT OPTIONS

;	Delimit expressions or suppress CR LF if at end of line. Examples: PRINT A;B PRINT A;
,	Tab to next print field. Example: PRINT A, B
TAB <(exp)>	Tab to exp column. Example: PRINT TAB (50);A
string	Print string or string-var. Example: PRINT "HI";$A(0)
# exp	*Print exp as hexadecimal in free format. Example: PRINT # 123
# ,exp	*Print exp as hexadecimal in byte format. Example: PRINT # ,50
# ;exp	*Print exp as hexadecimal in word format. Example: PRINT # ,A
<hex value>	*Direct output of ASCII codes. Example: PRINT "<OD> <OA>"
# string	⋆Print under specified format where: PRINT # "9999"I 9 = digit holder PRINT # "000-00-0000"SS 0 = digit holder or force 0 PRINT # "$$$,$$$.00"DLR $ = digit holder and floats $ PRINT # "SSS.0000"4*ATN1 S = digit holder and floats sign PRINT # "<<<.00>"I < = digit holder and float on negative >number PRINT # "990.99E"N E = sign holder after decimal PRINT # "990.99"N . = decimal point specifier PRINT # "999,990.99"N , = suppressed if before significant digit PRINT # "999,990∧00"I ∧ = translates to decimal point PRINT # "HI = 99"I any other character is printed.

7◀

GENERAL INFORMATION

ARITHMETIC OPERATIONS

A = B	Assignment
A − B	Negation or subtraction
A + B, $A + $B	Addition or string concatenation
A*B	Multiplication
A/B	Division
A∧B	Exponentiation
− A	Unary Minus
+ A	Unary Plus

LOGICAL OPERATORS

LNOT A	*1's complement of integer.
A LAND B	*Bit wise AND.
A LOR B	*Bit wise OR.
A LXOR B	*Bit wise exclusive OR.

RELATIONAL OPERATORS

1 if TRUE and 0 if FALSE

A = B	TRUE if equal, else FALSE.
A = = B	*TRUE if approximately equal (1E-7), else FALSE
A<B	TRUE if less than, else FALSE.
A< = B	TRUE if less than or equal, else FALSE.
A>B	TRUE if greater than, else FALSE.
A> = B	TRUE if greater than or equal, else FALSE.
A<>B	TRUE if not equal, else FALSE.
NOT A	*TRUE if zero, else FALSE.
A AND B	*TRUE if both non-zero, else FALSE.
A OR B	*TRUE if either non-zero, else FALSE.

OPERATOR PRECEDENCE

1. Expressions in parentheses
2. Exponentiation and negation
3. *, /
4. +, −
5. < =, <>
6. > =, <
7. =, >
8. = =, LXOR
9. NOT, LNOT
10. AND, LAND
11. OR, LOR
12. (=)ASSIGNMENT

SPECIAL CHARACTERS

:: Separates statements typed on same line.

! Tail remark used for comments after program statement

; Equivalent to PRINT.

ERROR CODES

1 =	SYNTAX ERROR
2 =	UNMATCHED PARENTHESIS
3 =	INVALID LINE NUMBER
4 =	ILLEGAL VARIABLE NAME
5 =	TOO MANY VARIABLES
6 =	ILLEGAL CHARACTER
7 =	EXPECTING OPERATOR
8 =	ILLEGAL FUNCTION NAME
9 =	ILLEGAL FUNCTION ARGUMENT
10 =	STORAGE OVERFLOW
11 =	STACK OVERFLOW
12 =	STACK UNDERFLOW
13 =	NO SUCH LINE NUMBER
14 =	EXPECTING STRING VARIABLE
15 =	INVALID SCREEN COMMAND
16 =	EXPECTING DIMENSIONED VARIABLE
17 =	SUBSCRIPT OUT OF RANGE
18 =	TWO FEW SUBSCRIPTS
19 =	TOO MANY SUBSCRIPTS
20 =	EXPECTING SIMPLE VARIABLE
21 =	DIGITS OUT OF RANGE (0< # of digits <12)
22 =	EXPECTING VARIABLE
23 =	READ OUT OF DATA
24 =	READ TYPE DIFFERS FROM DATA TYPE
25 =	SQUARE ROOT OF NEGATIVE NUMBER
26 =	LOG OF NON-POSITIVE NUMBER
27 =	EXPRESSION TOO COMPLEX
28 =	DIVISION BY ZERO
29 =	FLOATING POINT OVERFLOW
30 =	FIX ERROR
31 =	FOR WITHOUT NEXT
32 =	NEXT WITHOUT FOR
33 =	EXP FUNCTION HAS INVALID ARGUMENT
34 =	UNNORMALIZED NUMBER
35 =	PARAMETER ERROR
36 =	MISSING ASSIGNMENT OPERATOR
37 =	ILLEGAL DELIMITER
38 =	UNDEFINED FUNCTION
39 =	UNDIMENSIONED VARIABLE
40 =	UNDERFINED VARIABLE
41 =	EXPANSION EPROM NOT INSTALLED
42 =	INTERRUPT W/O TRAP
43 =	INVALID BAUD RATE
44 =	TAPE READ ERROR
45 =	EPROM VERIFY ERROR
46 =	INVALID DEVICE NUMBER

Cross Support

▶7

The Cross Assembler data base which is assigned to PUNIT, is read by the FORTRAN program as the first file at execution time. It is the actual Cross Assembler program written in internal code, and it is suggested that it be assigned to a permanent disk file.

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION
IUNIT	5	CR,CS MT,DF	80	TMS 9900 Source Input
LUNIT	6	CS,MT	80	Listing Output
OUNIT	7	CS,MT	80	TMS9900 Object Output
SUNIT	10	MT,DF	80	Assembly Scratch
PUNIT	11	CR,CS	80	Data Base INPUT

CR—CARD READER; CS—CASSETTE TAPE; MT—MAGNETIC TAPE; DF—DISKFILE; CP—CARD PUNCH; LP—LINE PRINTER

CROSS ASSEMBLER SYSTEM FILES

AORG places the expression value in the location counter, and defines the succeeding locations as absolute.

ABSOLUTE ORIGIN **AORG**

Syntax Definition:

[<label>]b̸ . . . AORGb̸ . . . <wd-exp>b̸ . . .[<comment>]

RORG places the expression value in the location counter, and defines the succeeding locations as relocatable.

RELOCATABLE ORIGIN **RORG**

Syntax Definition:

[<label>b̸ . . . RORGb̸ . . . [<exp>]b̸ . . .[<comment>]

DORG places the expression value in the location counter, and defines the succeeding locations as a dummy section. No object code is generated in a dummy section.

DUMMY ORIGIN **DORG**

Syntax Definition:

<label>b̸ . . . DORGb̸ . . . <exp>b̸ . . .[<comment>]

BSS first assigns the label, if present, and increments the location counter by the value of the expression.

BLOCK STARTING WITH SYMBOL **BSS**

Syntax Definition:

[<label>]b̸ . . . BSSb̸ . . . <wd-exp>b̸ . . .[<comment>]

BSS first increments the location counter by the value of the expression, and then assigns the label, if present.

BLOCK ENDING WITH SYMBOL **BES**

Syntax Definition:

[<label>]b̸ . . . BESb̸ . . . <wd-exp>b̸ . . .[<comment>]

EQU assigns an assembly-time constant to the label.

▶7

DEFINE ASSEMBLY-TIME CONSTANT **EQU**

Syntax Definition:

<label>b̸ . . . EQUb̸ . . . <exp>b̸ . . .[<comment>]

EVEN first assigns the label, if present, and then aligns the location counter on a word boundary (even address).

WORD BOUNDARY **EVEN**

Syntax Definition:

[<label>]b̸ . . . EVENb̸ . . .[<comment>]

OPTIONS allows cross referencing when XREF is specified, and allows printing of the symbol table when SYMT is present.

OUTPUT OPTIONS **OPTION**

Syntax Definition:

b̸ . . . OPTIONb̸ . . . <keyword>[,<keyword>] . . . b̸ . . . [<comment>]

IDT assigns a name to the program, and must precede any code-generating directive or instruction.

PROGRAM IDENTIFIER

IDT

Syntax Definition:

```
[<label>]b̸ . . . IDTb̸ . . . <string>b̸ . . .[<comment>]
```

TITL supplies a string to be printed at the top of each subsequent source listing page.

PAGE TITLE

TITL

Syntax Definition:

```
[<label>]b̸ . . . TITLb̸ . . . <string>b̸ . . . [<comment>]
```

LIST restores printing of the source listing.

LIST SOURCE

LIST

Syntax Definition:

```
[<label>]b̸ . . . LISTb̸ . . . [<comment>]
```

UNL inhibits printing of the source listing.

NO SOURCE LIST

UNL

Syntax Definition:

```
[<label>]b̸ . . . UNLb̸ . . . [<comment>]
```

PAGE directs the assembler to continue the source listing on the next page.

PAGE EJECT

PAGE

Syntax Definition:

```
[<label>]b̸ . . . PAGEb̸ . . . [<comment>]
```

BYTE places expressions in successive bytes, optionally assigning the label the address of the first byte.

INITIALIZE BYTE

BYTE

Syntax Definition:

```
[<label>]b̸ . . . BYTEb̸ . . . <exp>[,<exp>] . . . b̸ . . . [<comment>]
```

DATA places expressions in successive words, optionally assigning the label the address of the first word.

7◀

INITIALIZE WORD

DATA

Syntax Definition:

```
[<label>]b̸ . . . DATAb̸ . . . <exp>[,<exp>] . . . b̸ . . . [<comment>]
```

TEXT places characters in successive bytes, arithmetically negating the last character, and optionally assigns the label the address of the first character.

INITIALIZE TEXT

TEXT

Syntax Definition:

```
[<label>]b̸ . . . TEXTb̸ . . . [ – ]<string>b̸ . . . [<comment>]
```

DEF makes symbols available to other programs as external references.

EXTERNAL DEFINITION **DEF**

Syntax Definition:

[<label>]ƀ . . . DEFƀ . . . <symbol>[,<symbol>] . . . ƀ . . . [<comment>]

REF directs the assembler to look externally for symbols.

EXTERNAL REFERENCE **REF**

Syntax Definition:

[<label>]ƀ . . . REFƀ . . . <symbol>[,<symbol>] . . . ƀ . . . [<comment>]

DXOP assigns an extended operation to a symbol.

DEFINE EXTENDED OPERATIONS **DXOP**

Syntax Definition:

[<label>]ƀ . . . DXOPƀ . . . <symbol>,<term>ƀ . . . [<comment>]

END terminates the assembly

PROGRAM END **END**

Syntax Definition:

[<label>]ƀ . . . ENDƀ . . . [<symbol>]ƀ . . . [<comment>]

NOP places a no-operation code in the object file.

NO OPERATION **NOP**

Syntax Definition:

[<label>]ƀ . . . NOPƀ . . . [<comment>]

RT assembles as a return from subroutine by substituting a branch through register 11.

RETURN **RT**

Syntax Definition:

[<label>]ƀ . . . RTƀ . . . [<comment>]

SIMULATOR FILES

INTERNAL NAME	DEFAULT UNIT	DEVICE TYPE	RECORD LENGTH	FUNCTION	WHERE USED
INCOPY	4	MT,DF	80	Batch copy file	C
INCOM	5	TE,CR MT,DF	80	Simulation command	C
OUTPRT OUTTRC	6	MT,DF TE,CR	80 or 136	Listing output	L,C,R
INLOD	10	TE,CR MT,DF	80	Linker commands	L
OUTCOM	11	TE,LP	80 or 136	Prompts and error msg. for linker output	L
OUTSAV	17	MT,CP DF	80	Absolute object	L,S
INSCR	20	MT,DF	136	Input scratch file	C,R,S
OUTSCR	21	MT,DF	136	Output scratch file	L,C,R

Device type legend
TE—terminal; CR—card reader; MT—magnetic tape; DF—disk file; CP—card punch

Where used legend
L—link processor; C—command processor; R—run processor; S-save processor

In addition to the above unit number assignments, the user must also assign unique FORTRAN logical unit numbers to each TMS9900 object code module to be included in the LINK processor.

SIMULATOR DIRECTIVES

ORIGIN COMMAND. The "ORIGIN" command can be used to specify where relocatable code is to be loaded.

ORIGIN hex-number

INCLUDE COMMAND. The "INCLUDE" command directs the loader to load an object module from a data set (e.g., card reader, disc, tape). The data set must be a sequential data set and may contain one or more object modules. At least one "INCLUDE" command should be used in the LINK processor command stream. The format for the command is as follows:

INCLUDE n

ENTRY COMMAND. The "ENTRY" command specifies the program entry point to the loader. The format for the command is as follows:

ENTRY name

SUMMARY OF CONTROL LANGUAGE STATEMENTS

The formats of the control statements for the "COMMAND" processor are shown below, with a brief description following:

[label] $\begin{Bmatrix}\text{R}\\\text{RUN}\end{Bmatrix}$ [*] $\begin{Bmatrix}\text{F}\\\text{FOR}\end{Bmatrix}$ n $\left[\begin{Bmatrix}\text{FR}\\\text{FROM}\end{Bmatrix}\text{ i1}\right]\left[\begin{Bmatrix}\text{T}\\\text{TO}\end{Bmatrix}\right]$i2 [,label]

	Specifies where to start and stop simulation. Control passes to statement at label operand when a breakpoint occurs.
[label] $\begin{Bmatrix}\text{T}\\\text{TRACE}\end{Bmatrix}$ [list]	Specifies locations to be traced.
[label] $\begin{Bmatrix}\text{NOT}\\\text{NOTRACE}\end{Bmatrix}$ [list]	Disables trace for specified locations.
[label] $\begin{Bmatrix}\text{RE}\\\text{REFER}\end{Bmatrix}$ [list]	Specifies locations for reference breakpoint.
[label] $\begin{Bmatrix}\text{NOR}\\\text{NOREFER}\end{Bmatrix}$ [list]	Disables reference breakpoint at specified locations.
[label] $\begin{Bmatrix}\text{A}\\\text{ALTER}\end{Bmatrix}$ [list]	Specifies locations for alteration breakpoint.
[label] $\begin{Bmatrix}\text{NOA}\\\text{NOALTER}\end{Bmatrix}$ [list]	Disables alteration breakpoint at specified locations.
[label] $\begin{Bmatrix}\text{P}\\\text{PROTECT}\end{Bmatrix}$ [list]	Specifies areas for memory protection.
[label] IF (logical expression) label	Conditional transfer of control program.
[label] $\begin{Bmatrix}\text{J}\\\text{JUMP}\end{Bmatrix}$ label	Unconditional transfer of control program.
[label] $\begin{Bmatrix}\text{TI}\\\text{TIME}\end{Bmatrix}$ [n]	Prints the value of 9900 time and optionally sets a new value.
[label] $\begin{Bmatrix}\text{D}\\\text{DISPLAY}\end{Bmatrix}$ [D] $\begin{Bmatrix}\text{CP}\\\text{CPU}\end{Bmatrix}$ [register list]	Prints contents of registers.
[label] $\begin{Bmatrix}\text{D}\\\text{DISPLAY}\end{Bmatrix}$ $\begin{bmatrix}\text{D}\\\text{C}\end{bmatrix}$ $\begin{Bmatrix}\text{M}\\\text{MEMORY}\end{Bmatrix}$ list	Prints contents of memory.

▶7

SIMULATOR DIRECTIVES

[label] $\begin{Bmatrix}\text{D}\\ \text{DISPLAY}\end{Bmatrix}\begin{Bmatrix}\text{S}\\ \text{SYMBOL}\end{Bmatrix}\begin{bmatrix}\text{\$}\\ \text{symbol}\\ \text{number}\end{bmatrix}$	Prints values from symbol table.
[label] $\begin{Bmatrix}\text{D}\\ \text{DISPLAY}\end{Bmatrix}\begin{Bmatrix}\text{CR}\\ \text{CRU}\end{Bmatrix}\begin{Bmatrix}\text{I}\\ \text{INPUT}\\ \text{O}\\ \text{OUTPUT}\end{Bmatrix}$ list	Prints CRU values.
[label] $\begin{Bmatrix}\text{S}\\ \text{SET}\end{Bmatrix}\begin{Bmatrix}\text{C}\\ \text{CPU}\end{Bmatrix}$ register-value list	Places values into registers.
[label] $\begin{Bmatrix}\text{S}\\ \text{SET}\end{Bmatrix}\begin{Bmatrix}\text{M}\\ \text{MEMORY}\end{Bmatrix}$ location-value list	Places values into memory.
[label] $\begin{Bmatrix}\text{S}\\ \text{SET}\end{Bmatrix}\begin{Bmatrix}\text{I}\\ \text{INT}\end{Bmatrix}$ level, n_1 [,n_2,n_3]	Sets up one or more interrupts.
[label] $\begin{Bmatrix}\text{E}\\ \text{END}\end{Bmatrix}$	Disables breakpoints and traces, and initializes simulation. Passes control to next control statement.
[label $\begin{Bmatrix}\text{I}\\ \text{INPUT}\end{Bmatrix}\begin{Bmatrix}n_1\ \overset{n}{\text{TO}}\ n_2\end{Bmatrix}\begin{bmatrix}\text{F}\\ \text{FIRST}\\ \text{L}\\ \text{LAST}\\ \text{A}\\ \text{ALL}\end{bmatrix}$ [data]	Defines input lines and fields, and supplies data for program being simulated.
[label] $\begin{Bmatrix}\text{O}\\ \text{OUTPUT}\end{Bmatrix}\begin{Bmatrix}n_1\ \overset{n}{\text{TO}}\ n_2\end{Bmatrix}$	Defines output lines and fields, or prints output of program being simulated.
[label] $\begin{Bmatrix}\text{CONN}\\ \text{CONNECT}\end{Bmatrix}$ list	Connects input CRU lines to output CRU lines.
[label] $\begin{Bmatrix}\text{C}\\ \text{CONVERT}\end{Bmatrix}$ expression list	Evaluates and prints values of expressions in decimal and hexadecimal form.
$\begin{Bmatrix}\text{B}\\ \text{BATCH}\end{Bmatrix}$	Specifies batch mode.
[label] $\begin{Bmatrix}\text{L}\\ \text{LOAD}\end{Bmatrix}$	Loads Wp and PC from locations FFFC$_{16}$ and FFFE$_{16}$.
[label] $\begin{Bmatrix}\text{CL}\\ \text{CLOCK}\end{Bmatrix}$ t	Specify clock period.
[label] $\begin{Bmatrix}\text{M}\\ \text{MEMORY}\end{Bmatrix}\begin{Bmatrix}\text{RA}\\ \text{RAM}\\ \text{RO}\\ \text{ROM}\end{Bmatrix}\begin{Bmatrix}\text{R}\\ \text{READ}\end{Bmatrix} = n_1 \left[\begin{Bmatrix}\text{W}\\ \text{WRITE}\end{Bmatrix} = n_2\right]$ list	Define available memory. Default is 32K RAM.
[label] $\begin{Bmatrix}\text{SA}\\ \text{SAVE}\end{Bmatrix}$	Create absolute object module.
[label] $\begin{Bmatrix}\text{W}\\ \text{WIDTH}\end{Bmatrix}$ n	Specifies number of columns available for printing.

MONITOR COMPLETION CODES

The simulator signals completion by executing and writing an appropriate STOP I statement, where I takes on one of the following values:

CODE	MEANING
0	Normal completion
1	Abnormal completion from LNKPRC
2	Premature EOF —If this error occurs it indicates that a premature EOF was encountered while attempting to reposition the BATCH command file.
3	Internal error; invalid label value
4	Roll memory overflow
5	Loader error —If this error occurs it means an attempt was made to load an object file into simulated memory and it failed causing termination of the link processor.
8	Abnormal completion from LOADER
9	Abnormal completion from CMDPRC
99	Internal error —Illegal completion from CMDPRC Internal error
999	Internal error —Illegal parameter passed to WRITER

If an error of 99 or 999 results, an internal error has occurred and the error should be reported to TEXAS INSTRUMENTS INC.

LINK PROCESSOR ERRORS

CODE	MESSAGE
L01	Load not completed
L02	Multiply defined external symbol (name)
L03	Empty object file on unit
L04	Attempt to load undefined memory
L05	Tag D follows tag 0
L06	Invalid tag character
L09	Undefined external memory
L13	Empty memory on save
L14	(name) not in external symbol table
L18	Maximum memory size exceeded
L19	Missing end
L21	Checksum error (computed value)
L22	Odd origin value specified—even value used
L24	Ref chain loop
L25	Object module does not start with tag 0
L26	Odd value (value) specified for tag (tag) even value used
L27	Missing F tag in record (number)
L28	Bad REF chain for (name)
L29	Bad object format in object module
L30	Illegal hex digit in field (digit)

COMMAND PROCESSOR ERRORS

CODE			CODE		
NUMBER	NAME	MESSAGE	NUMBER	NAME	MESSAGE
1	BADCHR	Bad character	18	RANGE	Range error
2	BADCMD	Unrecognizable command	19	SYNTAX	Syntax error
3	BADIGT	Bad digit	20	TOOMNY	Too many values
4	BADMOD	Bad module name	21	UNDEF	Undefined symbol
5	BADREG	Bad register mnemonic			
6	BADVAL	Bad value			
7	CRUSPC	CRU specification error			
8	FLDCNT	Too few/many fields			
9	HITEOF	Hit EOF			
10	HITEOL	Hit end-of-line			
11	MEMDEF	Undefined			
12	MISSEQ	Missing equal sign			
13	NODATA	No data found			
14	NOROL	No data rolls available			
15	NOSET	Set not performed			
16	NOTIMP	Command not implemented			
17	ORDER	Command out of order			

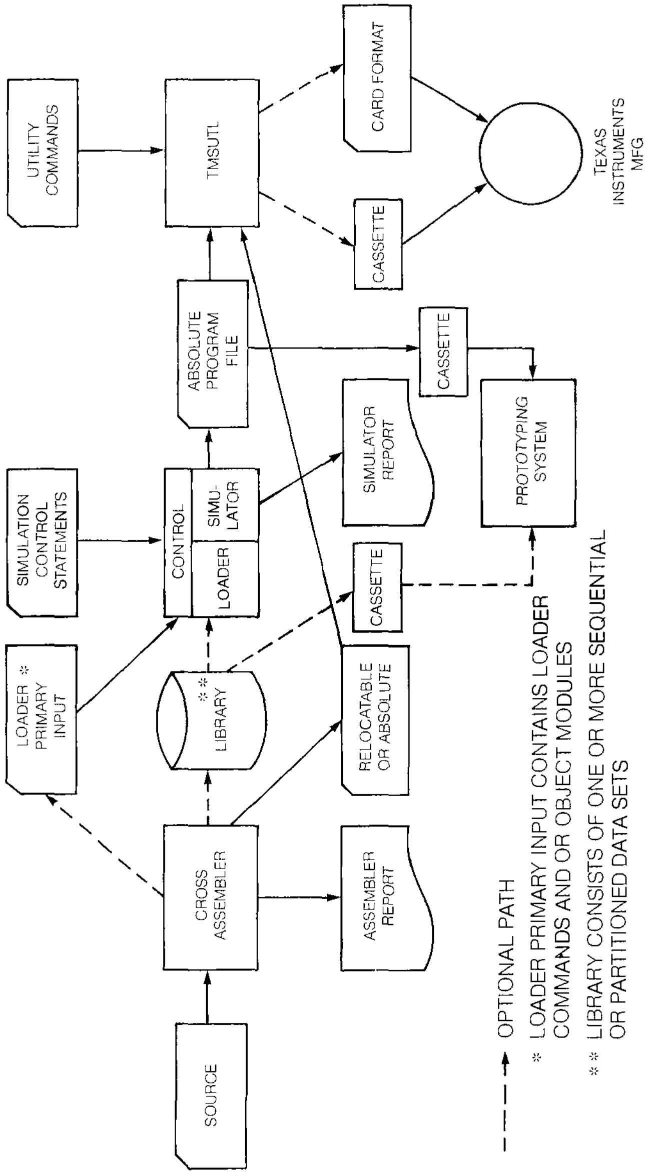
RUN PROCESSOR ERRORS

CODE	MESSAGE
1	PC interrupt vector entry in undefined memory
2	WP interrupt vector entry in undefined memory
3	Register out of address space (WP 65502)
4	Registers in undefined memory
5	Registers in ROM
6	PC interrupt vector refer breakpoint
7	WP interrupt vector refer breakpoint
8	Register alter breakpoint
9	Register protect breakpoint
10	Register refer breakpoint
11	Undefined opcode
12	Undefined memory reference
13,14	Unused
15	PC refer breakpoint
16	Unimplemented opcode
17,18,19	Unused
20	Destination address in undefined memory
21	Destination refer breakpoint
22	Destination alter breakpoint
23	Destination ROM breakpoint
24	Unused
25	Source address in undefined memory
26	Source refer breakpoint
27	Source alter breakpoint
28	Source ROM breakpoint

TMSUTL

CONCEPT

TMSUTL is a general purpose ultility program that accepts as input TI microprocessor object format, PROM manufacturing formats, or ROM manufacturing formats. This data is syntax checked, output options are gathered, the input data converted and an output file is produced.



OPTIONAL PATH

* LOADER PRIMARY INPUT CONTAINS LOADER COMMANDS AND OR OBJECT MODULES

** LIBRARY CONSISTS OF ONE OR MORE SEQUENTIAL OR PARTITIONED DATA SETS

INPUT, OUTPUT CONTROL CARD FORMATS

GENERAL DESCRIPTION

INPUT frmt [addr1 addr2] [WIDTH = x] [PARTITION = y]

frmt — is the format number (integer 1-12).
addr1 — is the starting address where input data is to be stored.
addr2 — is the maximum address where data is to be stored.
x — is the bit width of the input words.
y — is the number of input data set partitions 1 Y 4

OUTPUT num addr1 addr2 WIDTH = x PARTITION = y

num — is the format number (integer 1-12).
addr1 — is the minimum address to be output.
addr2 — is the maximum address to be output.
x — is the bit width of an output word.
y —

EOF—End of COMMAND FILE indicator

AVAILABLE FORMATS

FORMAT #	FORMAT	INPUT	OUTPUT
1	Hexadecimal #1 (PROM)	X	X
2	Hexadecimal #2 (ROM)	X	X
3	BNPF	X	X
4	271 & 371 ROM/HILO of prototyping System	X	X
5	TMS8080/TMS1000 Absolute Object from SIM8080/SIM1000 Loader/Simulator	X	X
6	TMS1000 Absolute ROM Object from Assembler	X	X
7	TMS1000 Listed Absolute Object	X	X
8	TMS1000 OPLA Data	X	
9	TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	X	X
10	TMS9900 Compressed Absolute Object of Prototyping System	X	X
11	TI4700 ROM	X	X
12	TI4800 ROM	X	X

▶7

TMSUTL FORMAT PATHS

Output Format ⟶	1	2	3	4	5	6	7	8	9	10	11	12
1) Hexadecimal #2 (PROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
2) Hexadecimal #2 (ROM)	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
3) BNPF	YES	YES	YES	YES	YES	YES	YES	NO	YES	YES	YES	YES
4) 271 & 371 ROM/ HILO of Prototyping System	YES	YES	YES	YES	NO	NO	YES	NO	NO	NO	YES	YES
5) TMS1000 / TMS8080 Absolute Object from Loader/Simulator	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
6) TMS1000 Absolute ROM Objects from Assembler for masking	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
7) TMS1000 Listed Absolute Object	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	YES	YES
8) TMS1000 OPLA Data	YES	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO	NO
9) TMS9900 Standard Absolute Object of Cross Support System (Assembler or Loader/Simulator) & Prototyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
10) TMS9900 Compressed Absolute Object of Protoyping System	YES	YES	YES	YES	NO	NO	NO	NO	YES	YES	YES	YES
11) TI4700 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES
12) TI4800 ROM	YES	YES	YES	YES	YES	NO	YES	NO	NO	NO	YES	YES

DATA DELIMITERS

The following is a table of data delimiters or end-of-module records for Input Data.

FORMAT #	TYPES
1. Hex format 1	End of file record (:00)
2. Hex format 2	Trailer record — "END OF TEXT" (hollerith code 12-9-3) character followed by 79 non-blank characters (without asterisks)
3. BNPF	End of file record ($ in column 1)
4. 271/371 ROM and HILO of Prototyping System	End of file record ($END)
5. TMS8080/TMS1000 Absolute Object from Loader/Simulator	End record (+ END)
6. TMS1000 Absolute ROM Object	End of file record ($END)
7. TMS1000 Listed Absolute Object	End of file record ($END)
8. TMS1000 OPLA Data	End of file record ($END)
9. TMS9900 Standard Absolute Object	End of module record (:)
10. TMS9900 Binary Compressed Absolute Object	End of file record ($END)
11. TI4700 ROM	End of file record ($END)
12. TI4800 ROM	End of file record ($END)

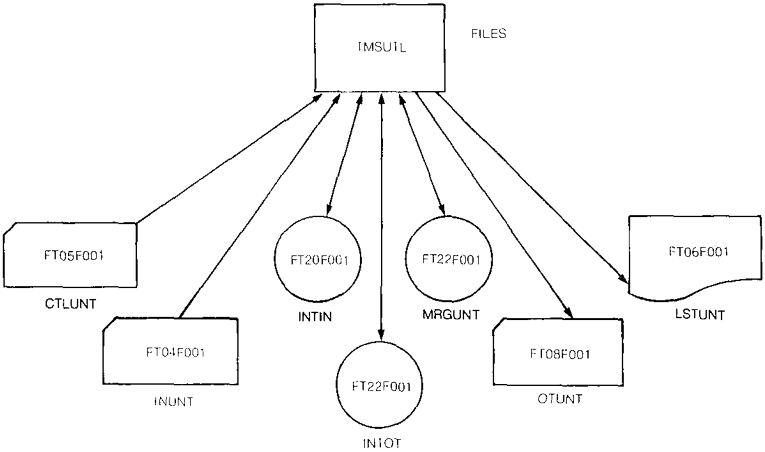
ADDRESS RANGES FOR FORMATS

FORMAT#	FORMAT	ADDRESS RANGE
1	Hexadecimal #1 (PROM)	(0-FFFF)$_H$
2	Hexadecimal #2 (ROM)	None
3	BNPF	None
4	271 & 371 ROM/HILO of Prototyping System	None
5	TMS8080/TMS1000 Absolute Object from Loader/ Simulator	(0-255)
6	TMS1000 Absolute ROM Object	(0-800)$_H$
7	TMS1000 Listed Absolute Object	(0-1 Chapter 0-15 page 0-3F location)$_H$
8	TMS1000 OPLA Data	(0-1F)$_H$
9	TMS9900 Standard Absolute Object	(0-FFFF)$_H$
10	TMS9900 Compressed Absolute Object	(0-FFFF)$_H$
11	TI4700 ROM	(0-400)$_H$
12	TI4800 ROM	(0-400)$_H$

INPUT AND OUTPUT WIDTHS FOR FORMATS

FORMAT#	FORMAT	WIDTH (BITS)
1	Hexadecimal #1 (PROM)	8
2	Hexadecimal #2 (ROM)	8
3	BNPF	2 or 4 or 8 or 16
4	271 & 371 ROM/HILO of Prototyping System	4 or 8
5	TMS8080/TMS1000 Absolute Object from Loader/ Simulator	8
6	TMS1000 Object from Assembler	8
7	TMS1000 Listed Absolute Object	8
8	TMS1000 OPLA Data	8 or 16
9	TMS9900 Standard Absolute Object	16
10	TMS9900 Compressed Absolute Object	16
11	TI4700 ROM	8
12	TI4800 ROM	4 or 8

FILES DEFINITIONS & DESCRIPTIONS



CTLUNT — Input file for control cards.

INUNT — Input file for data.

INTIN — Intermediate file for storage of input data. It must be a rewindable file with a logical record length of 80 bytes.

INTOT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

OTUNT — Output file for translated data.

LSTUNT — Print file for listing of data and error messages.

MRGUNT — Intermediate file for storage of internal data. It must be a rewindable file with a logical record length of 80 bytes.

TMSUTL ERROR MESSAGES

- INPUT CONTROL CARD MISSING. Input control card missing or misplaced; it should be the first control card.
- INVALID CONTROL CARD FIELD. Control card has an invalid field. Dollar signs point to the beginning and the end of the field.
- OUTPUT FORMAT INCOMPATIBLE WITH INPUT FORMAT. The output format specified can not be converted from the input format specified.
- OUTPUT FORMAT MISSING. Output control card missing or misplaced; it should follow the Input card.
- ADDR2 ADDR1 OR BOTH NOT SPECIFIED. Either minimum or maximum address is invalid. Addr1 must be less than or equal to Addr2.
- WIDTH INVALID FOR I/O FORMAT SPECIFIED. For the format specified the bit width is invalid.
- PARTITION ERR. The Input bit width times the number of input partitions is not equal to the width times the number of output partitions.
- ERROR DETECTED ON INPUT CARD. The format of a data card is invalid, check the field pointed to by the dollar signs.
- INPUT OUT OF SEQUENCE. The addresses of the input data are not in sequential order.
- # OF WORDS INPUT FOR CURRENT PARTITION NOT EQUAL TO THAT IN PREVIOUS PARTITION. The number of words input for each partition is not equal. Check the input data.
- ADDRESS OUT OF RANGE. Either Addr1 or Addr2 is out of range or the address read on the input data is out of range of the format specified.

STOP CODES	ERROR
1	Input data error. (A message describing the error is output before this is issued.)
2	Format not implemented yet in EOF.
3	Format not implemented yet in TRANS.

STOP CODES	ERROR
90	DECHEX unable to find H or blank.
91	Data will not fit in card field passed to AFORMT.
92	Invalid format number in EOF.
93	Invalid width passed to INWORD.
94	SHFTR called with invalid arguments.
95	TRANS called with an invalid format number.