TEXAS INSTRUMENTS

TM 990

TM 990/302
Software Development
Module User's Guide

μP

MICROPROCESSOR SERIES™

June 1980

TABLE OF CONTENTS

SECTION IV. SYMBOLIC ASSEMBLER

SECTION V. RELOCATING LOADER PROGRAM

SECTION VI. PROGRAM DEBUGGER

SECTION VII. EPROM PROGRAMMER

SECTION VIII. DUMP MEMORY COMMAND

SECTION IX. SETTING BAUD RATE AT SECOND EIA PORT

SECTION X. USER UTILITY CALLS

SECTION XI. UPLINK BETWEEN TM 990/302 AND HOST COMPUTER

APPENDICES

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION 1

INTRODUCTION

1.1 GENERAL

The TM 990/302 Software Development Board (SDB), used with a TM 990/10X micro-computer board, provides the basic software development utilities needed to develop software for a TM 990 microcomputer system. This manual provides information on the installation as well as the operation of the TM 990/302, shown in Figure 1-1. A software development system (Figure 1-2) consists of the following:

- TM 990/302 Software Development Board

- TM 990/100M or TM 990/101M microcomputer

- Power supply (such as the TM 990/518)

- Data terminal (EIA or TTY interface)

- Board interface (such as the TM 990/510 chassis)

- One or two audio cassette recorder/players for storage of source or object records (it is possible to operate the system with one recorder)

- Proper cabling.

- Possible expansion memory (TM 990/201 or TM 990/206)

Equipment configurations and setup are explained in Section 2 of this manual. Software utilities provided in EPROM on the TM 990/302 board include:

- Text Editor (Section 3)

- Symbolic Assembler (Section 4)

- Relocating Loader (Section 5)

- Debugger (Section 6)

- EPROM Programmer (Section 7)

- Memory Dump (Section 8)

- Second EIA Baud Rate (Section 9)

- User Utility Calls (Section 10)

- Downloading Object Program From Host Computer To TM 990/302 (Section 11)

WAIT/
NO WAIT
JUMPERS

LOAD/
NO LOAD
JUMPERS

SOCKET XU12 WITH
DECODE PROM FOR
TM 990/100M BOARD

SWITCH S1
FOR MEMORY
MAPPING

CONNECTOR P3

CONNECTOR P1

LED AT CRU ADDRESS 1706_{16}
LIGHTS WHILE EPROM
PROGRAMMER IS EXECUTING

CONNECTOR P2

EPROM
AREA

RAM
AREA
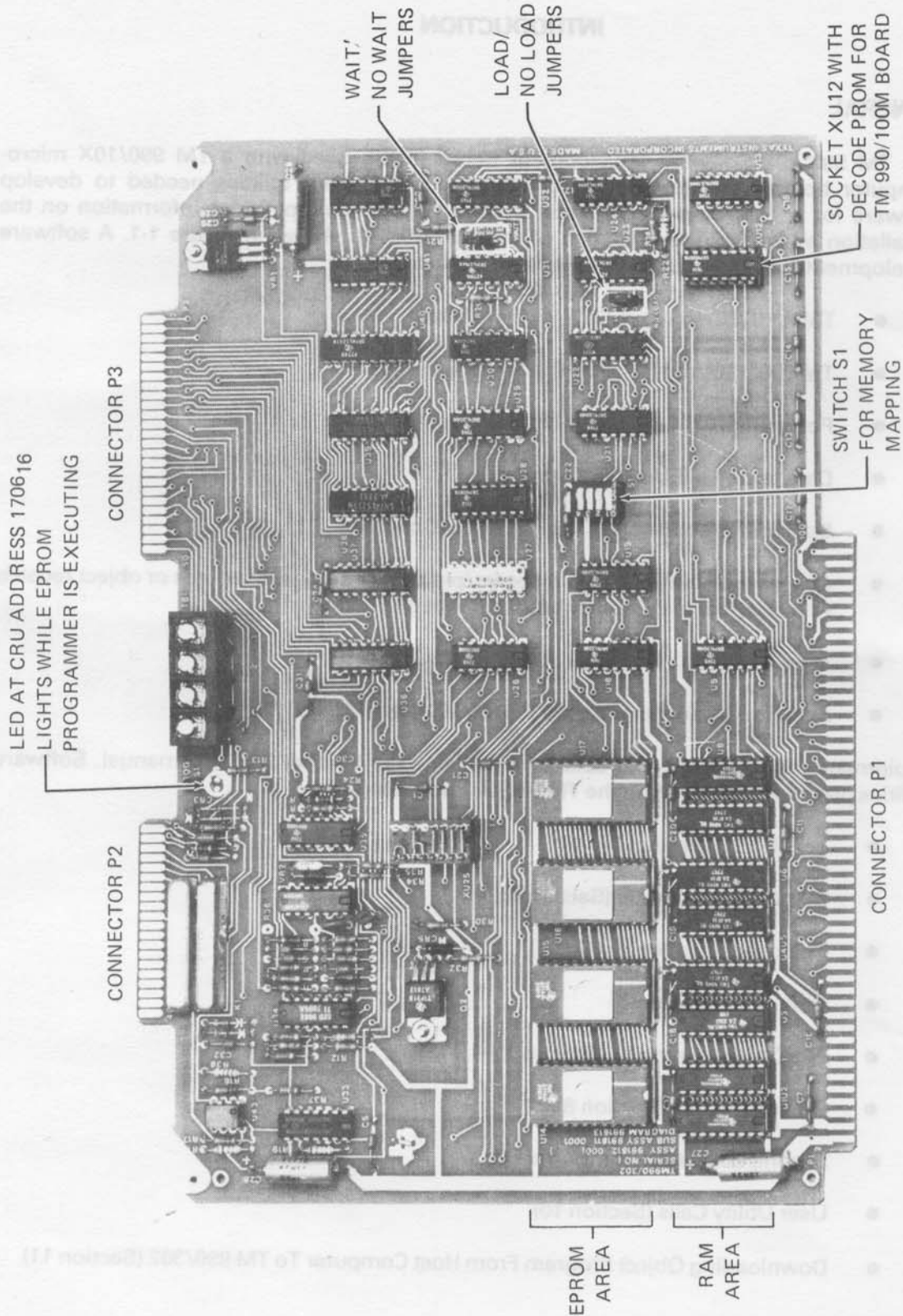
Figure 1-1. TM 990/302 Software Development Board

POWER SUPPLY
+5V, +12V,
-12V,
+30 TO +50V

TERMINAL FOR
COMMAND INPUT,
SOURCE LISTING,
ERRORS

SDB MONITOR
IN EPROM

TM 990/302
SOFTWARE
DEVELOPMENT
BOARD

RAM

TM 990/10X
MICROCOMPUTER
BOARD
(CPU)

RAM

CASSETTE
RECORDER/
PLAYER TO
CONTAIN
SOURCE AND
OBJECT FILES

EPROM PROGRAMMER
PERSONALITY CARD
FOR TMS 2708, TMS
2716, TMS 2508, TMS 2516,
TMS 2532

OPTIONAL
SECOND CASSETTE
RECORDER/PLAYER
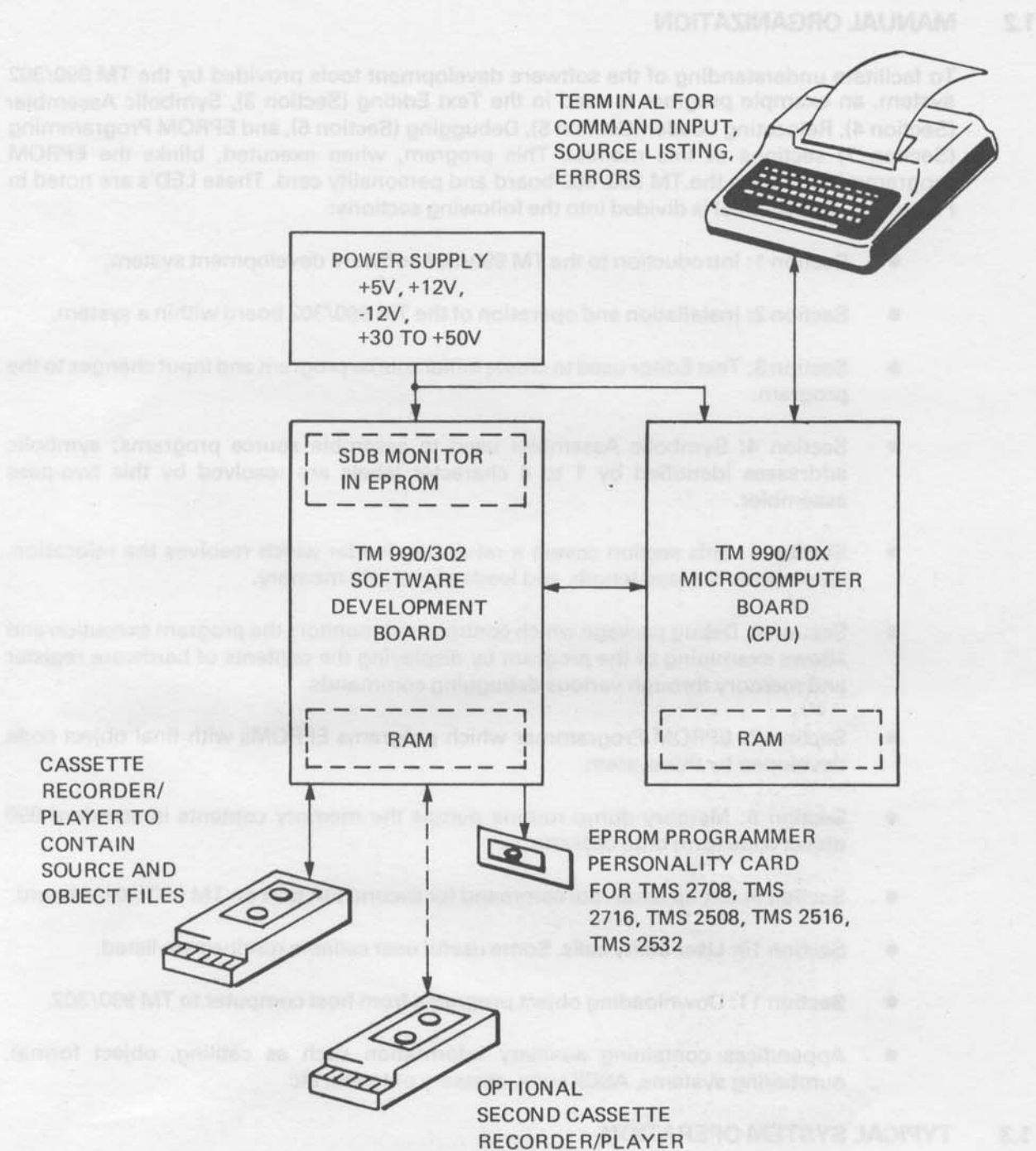
Figure 1-2. Software Development System Diagram

1.2 MANUAL ORGANIZATION

To facilitate understanding of the software development tools provided by the TM 990/302 system, an example program is used in the Text Editing (Section 3), Symbolic Assembler (Section 4), Relocating Loader (Section 5), Debugging (Section 6), and EPROM Programming (Section 7) sections of this manual. This program, when executed, blinks the EPROM programming LED on the TM 990/302 board and personality card. These LED's are noted in Figure 1-1. This manual is divided into the following sections:

- Section 1: Introduction to the TM 990/302 software development system.

- Section 2: Installation and operation of the TM 990/302 board within a system.

- Section 3: Text Editor used to create initial source program and input changes to the program.

- Section 4: Symbolic Assembler used to assemble source programs; symbolic addresses identified by 1 to 4 character labels are resolved by this two-pass assembler.

- Section 5: This section covers a relocating loader which resolves the relocation, determines the load length, and loads object into memory.

- Section 6: Debug package which controls and monitors the program execution and allows examining of the program by displaying the contents of hardware register and memory through various debugging commands.

- Section 7: EPROM Programmer which programs EPROMs with final object code developed by this system.

- Section 8: Memory dump routine dumps the memory contents in standard 990 object code form onto cassette.

- Section 9: Set up baud rate command for second EIA port on TM 990/101M board.

- Section 10: User utility calls. Some useful user callable routines are listed.

- Section 11: Downloading object programs from host computer to TM 990/302.

- Appendices containing auxiliary information such as cabling, object format, numbering systems, ASCII code, glossary of terms, etc.

1.3 TYPICAL SYSTEM OPERATION

A flowchart depicting typical system operation with the TM 990/302 Software Development Board is shown in Figure 1-3. Operation steps include:

(1) Initialize System. This includes setting up the system as described in Section 2: system connections, powerup, cassette ready, system reset, and call-up utilities.

(2) Text Editing. The Text Editor allows creation of source programs as well as updating, insertion, or deletion of source lines. After developing the source programs,

in TMS 9900 assembly language the Text Editor writes the developed programs to cassette upon command. Programming is made easier by the use of symbolic labels and 11 assembler directives which are explained in Chapter 4 and Appendix F.

(3)   Assemble Source Program. After generating a source program of assembly language statements, the Symbolic Assembler is called to assemble the program into object code. The assembler will provide an assembly listing, with errors noted, and a cassette of assembled code called object. In the call to the assembler, the user can specify what options are desired from the assembly such as (1) listing to check source code, or (2) an object code, (3) or both. The SDB program calls allow flexibility for the user in configuring his system. If a particular device in the system is not desired (such as the object cassette recorder when a listing-only is desired), that device can be assigned to "dummy" during SDB program call; thus "deleting" it from the system configuration. One feature of the assembler is that if an error is found, no-op instructions (ignore this line) are substituted for the object code in question. This will permit complete assembly of the program with space provided for later updating the object. Assembler object is not relocatable.

(4)   Load Object Code. When the object code has been assembled by the Symbolic Assembler, it can be loaded into memory by the Relocating Loader. Even though a program can be assembled as if it was to be loaded into memory at memory address (M.A.) 0000_{16}, the Relocating Loader will load the program where directed in memory, and resolve any conflicts in addressing. These conflicts occur when code is assembled for loading at one address but is loaded at a different address.

(5)   Debug the Program. Debugging the program means to run the program in a controlled environment, checking on its performance at selected places. This could mean running the program until a specific instruction is executed, then stopping the program to inspect various memory locations or hardware registers. This same check could be made after running the program for a limited number of instructions. If an error is found in the program, the user can substitute different values in memory, changing instructions or data values, and re-execute the program again, checking for proper operation. Equipped with an assembler listing showing memory locations, the user can relate machine instructions and data to the assembled source. He can also check and change workspace register contents, CRU values, and hardware registers (Program Counter, Workspace Register, and Status Register).

(6)   Further Editing/Loading/Debugging. If the program needs further changes, the source program can be brought in by the Text Editor to incorporate changes, reassemble, and then reloaded into memory and executed under the Debugger. This cycle can be run until the user is satisfied with the results.

(7)   Program the EPROM. When satisfied with program performance, the object program can be reloaded into memory and then programmed onto one of several types of eraseable programmable read-only memories (EPROMS). The EPROM can then be placed in the user's system ("target" sytem) and executed. "Personality" cards are provided with the TM 990/302 SDB to accommodate a variety of EPROM types. These cards plug into connector P2 on the left side of the SDB (left side facing the board in a card cage). One personality card has a plug for insertion of a TMS 2708 or TMS 2716 PROMs, the other card can accommodate the TMS 2508 or TMS 2516 or TMS 2532 EPROM's.

START

INITIAL INPUT,
EDITING

TEXT
EDITOR

SOURCE TAPE

SOURCE INPUT,
OBJECT OUTPUT

ASSEMBLER

LISTING,
ERRORS

ERRORS
?

YES

NO

RELOCATING
LOADER

OBJECT INPUT

INTERACTIVE
CONTROL OF
EXECUTION

DEBUGGING
PROGRAM

NO

PROGRAM
WORK
?

YES

SAVE OBJECT
ON CASSETTE

OBJECT INPUT

PROGRAM
EPROM

TO EPROM

PLACE EPROM
IN TARGET
SYSTEM
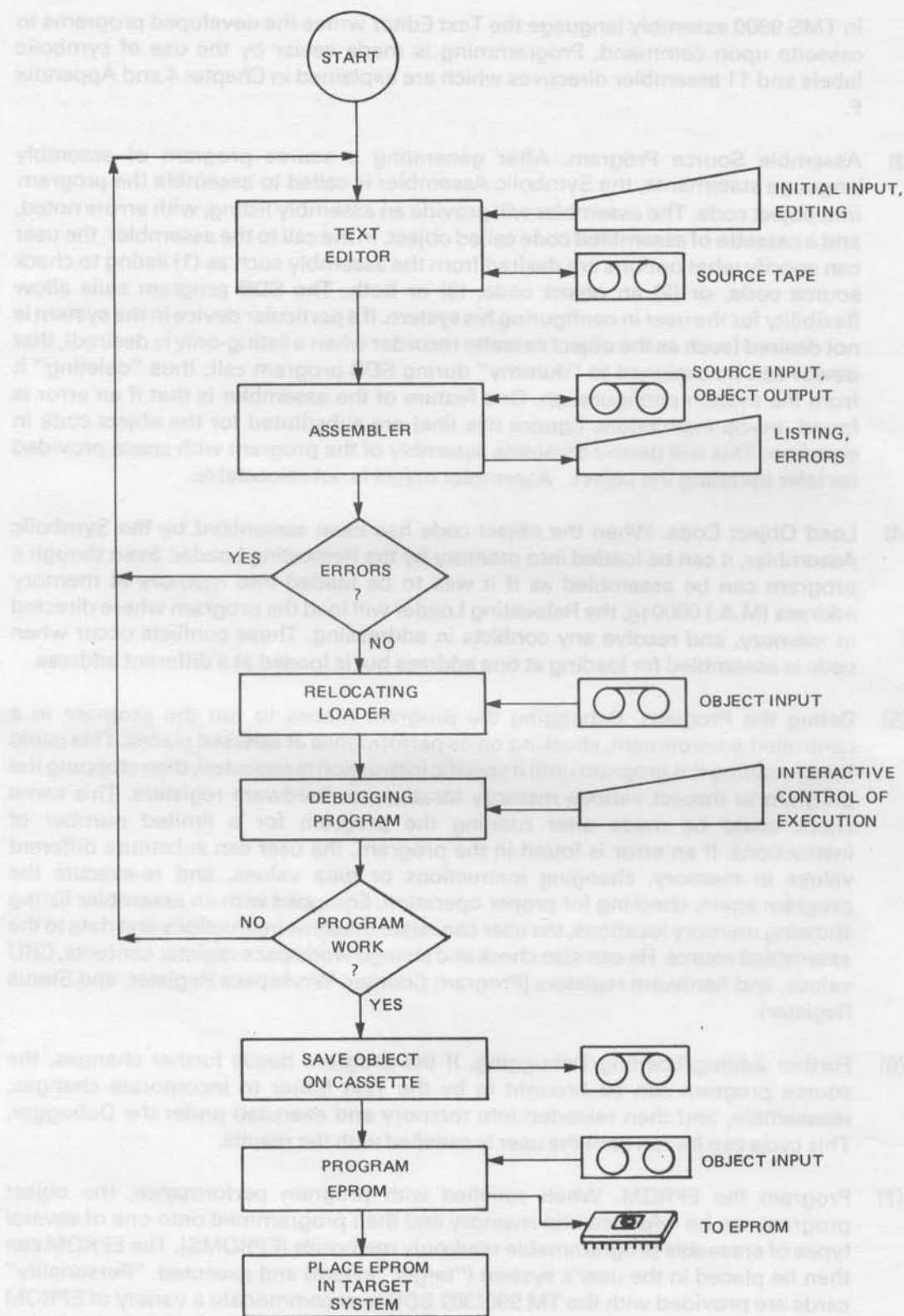
Figure 1-3. Typical Software Development Sequence

1.4   MEMORY AREAS RELOCATED

Position of random access memory (RAM) and erasable programmable memory (EPROM) on the microcomputer boards will be relocated at different addresses than as shipped from the factory. To accomplish this, the TM 990/100M board requires a new address-decode PROM while the TM 990/101M board requires several jumper placements (these are described in detail in Section 2). EPROM on the microcomputer board is available for user generated application software, but is not included in the SDB system memory mapping.

1.5   SPECIFICATIONS

Power:

The following are power requirements for the different boards in the system:

	+5V	−12V	+12V	35-55V
TM 990/302	1.5 A	50 mA	50 mA	0.1 A*
TM 990/100M	1.4 A	0.1 A	0.2 A	0
TM 990/101M	1.7 A	0.1 A	0.3 A	0

*Required only for EPROM programmer on TM 990/302

Temperature range: 0 to 55° C

Humidity: Up to 95%, noncondensing

1.6   APPLICABLE DOCUMENTS

- TM 990/100M Microcomputer User's Guide

- TM 990/101M Microcomputer User's Guide

- TMS 9900 Microprocessor Data Manual

- TMS 9901 Programmable Interface Data Manual

- TMS 9902 Asynchronous Communication Controller

- The MOS Memory Data Book For Design Engineers

1.7   COMMAND FORMAT SYNTAX

Throughout this manual, formats for the different SDB commands are provided in abbreviated form as well as in the form of examples. Table 1-1 defines the syntax used in the abbreviated command formats.

TABLE 1-1. COMMAND SYNTAX CONVENTIONS

SYMBOL	EXPLANATION
< >	Item to be supplied by the user
[ ]	Optional item(s) in brackets may be included or excluded at the user's discretion
{ }	Choose one of several optional items from the items in brackets
(CR)	Carriage return
∧	Space bar
>	Hexadecimal value

1.8   CHARACTER CHANGES USING CONTROL H AND CONTROL F KEYS

In communicating with the SDB monitor, errors to keyboard commands can be corrected before the user enters a carriage return to execute the command. To correct one or more keystrokes entered, use the CONTROL H key (press the CONTROL key, then the H key)to backspace to the key entry in error. Then press the correct key entry. If valid key entries were backed over during this operation, use the CONTROL F (press the CONTROL key, then the F key) to forward space over correct entries (an alternate method would be to repeat the keystrokes that were backspaced over).

INSTALLATION AND OPERATION

2.1 GENERAL

This section contains information on installing the TM 990/302 and general operation of the system. Figure 2-1 is a diagram showing system hookup.

2.2 UNPACKING

Check the carton for any outside breakage. If any is found, report this to your supplier or carrier. If signing for receipt of carton from a carrier, note any carton breakage on the receipt paperwork. Remove the TM 990/302 board from its carton and packing. Examine the board for any discrepancies; if found, report these to your supplier or distributor.

2.3 INSTALLATION

2.3.1 MINIMUM CONFIGURATION

The minimum system configuration for the TM 990/302 should include the following:

- TM 990/302 Software Development Board

- TM 990/100M or TM 990/101M microcomputer, fully populated with RAM (TMS 4042's on the TM 990/100M, TMS 4045's on the TM 990/101M)

- Card cage (TM 990/510 or TM 990/520 or equivalent)

- Power Supply (TM 990/518 or equivalent)

- Data terminal such as a:
    - Decwriter II
    - Hazeltine 1500 series
    - Lear Siegler ADM-1, ADM-2, or ADM-3
    - Soroc IQ 120
    - Teletype model 3320 5JE
    - Texas Instruments models 733 KSR* or 743 KSR*

- Cassette Recorder/Players (two preferred) of the following recommended models:
    - General Electric 3-5121B
    - Panasonic RQ-413 AS
    - Realistic CTR-40 (Radio Shack)
    - Realistic CTR-41 (Radio Shack)
    - Sears 799.21683700
    - Sharp RD-610

CAUTION

Operation with models other than those above may yield unreliable data transfers or cause damage to the TM 990/302 control relay due to excessive inrush currents. Use of the TM 990/302 with other than the above tape units voids the factory warranty. Subsection 2.3.7 covers tape unit checkout.

*Trademark of Texas Instruments Incorporated

● Cassette tapes of the following recommended brands:
  —Verbatim R300H
  —Radio Shack Digital Tape C20-260301
  —Texas Instruments Digital Tape 360333-0001
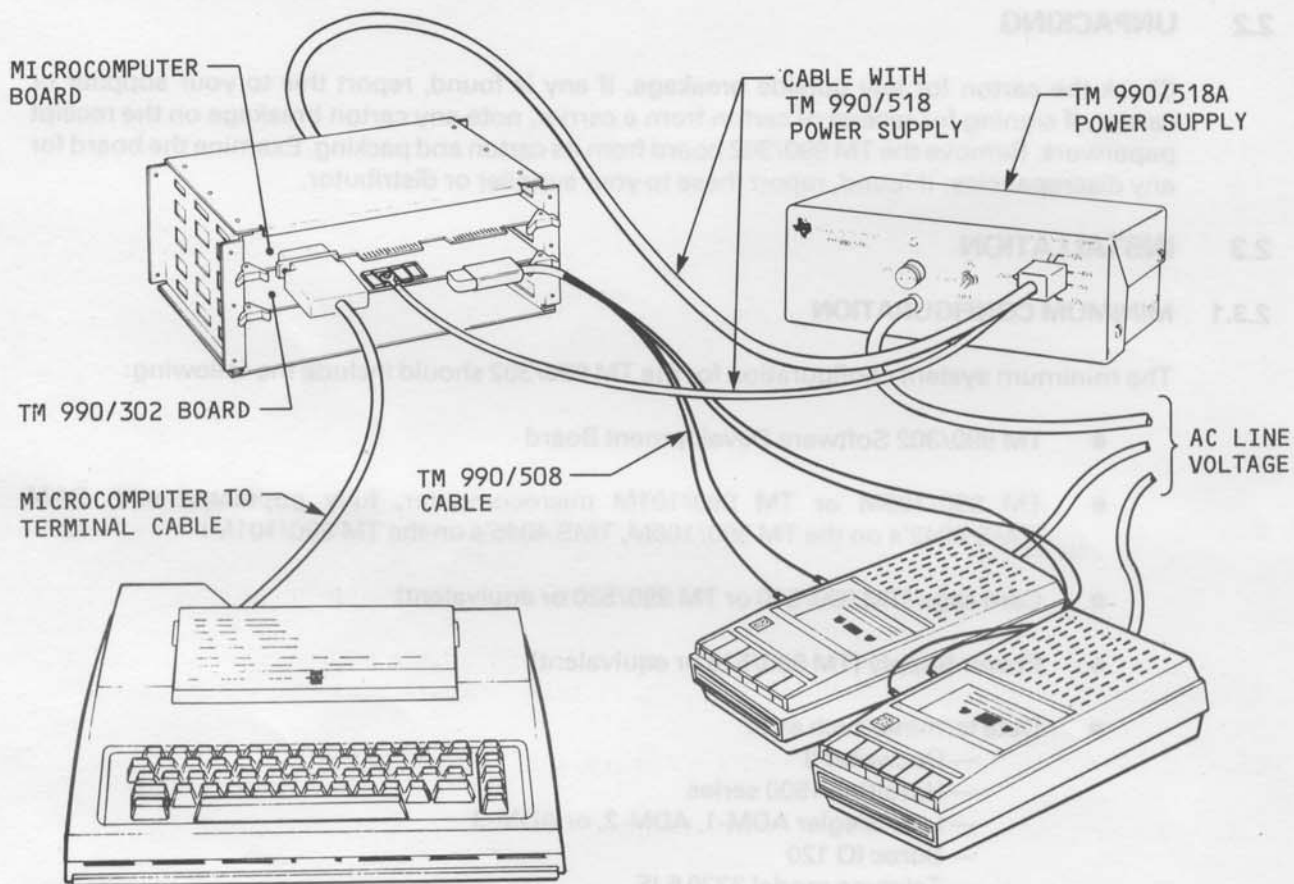
Figure 2-1. Software Development System Configuration

2.3.2 CONNECT POWER SUPPLY

Connect the TM 990/518 power supply to the TM 990/5XX card cage as shown in Figure 2-2(a). Verify correct voltages at the chassis rear panel connections before installing any boards into the chassis.

Do not connect EPROM programming power unless object is in memory, ready to be programmed onto the EPROM. There are two alternate ways to attach EPROM programming power to the TM 990/302 board. Attach the 35V to 55V EPROM power at the power supply terminal [Figure 2-2(b)] or at connector P2 as follows:

- At the power supply terminal on the TM 990/302 board, attach [Figure 2-2(b)]:

  1. TB1-1 to ground

  2. TB1-2 to EPROM programming voltage source

  3. TB1-3 and TB1-4 are unconnected

- or, at connector P2 on the TM 990/302 board, attach [see Figure 2-2(c)]:

  1. Pin 20 to voltage source

  2. Pin 1, 3, 5, or 7 to ground.

NOTE

Disconnect the EPROM power if the EPROM programmer is not going to be used.

CAUTIONS

1. Do not touch the board with EPROM programming voltage applied.
2. Do not read from cassettes while EPROM voltage is connected and an EPROM is in the personality card attached to the TM 990/302.

2.3.3 CHANGE RAM MEMORY MAPPING AND INSTALL SYSTEM PC BOARDS

2.3.3.1 Memory Mapping Change on Microcomputer Boards

The TM 990/302 Software Development Board software does not utilize the standard memory adressing on the TM 990/10X microcomputer boards. On these boards, standard memory configuration has random access memory (RAM) located in the highest memory addresses while erasable read-only memory (EPROM) begins at addresses 0000_{16}. The SDB software requires an opposite configuration with RAM in lower memory and the EPROM on the microcomputer board disabled. The SDB system software will be resident in upper memory on the TM 990/302 board. Figure 2-3 depicts the memory map of the TM 990/302 operating system using both boards. It is assumed that:

- Both microcomputer boards are fully populated with RAM.

- Onboard RAM on the TM 990/100M is mapped from M.A. 0000 to $03FF_{16}$.

- Onboard RAM on the TM 990/101M is mapped from M.A. 0000 to $0FFF_{16}$.
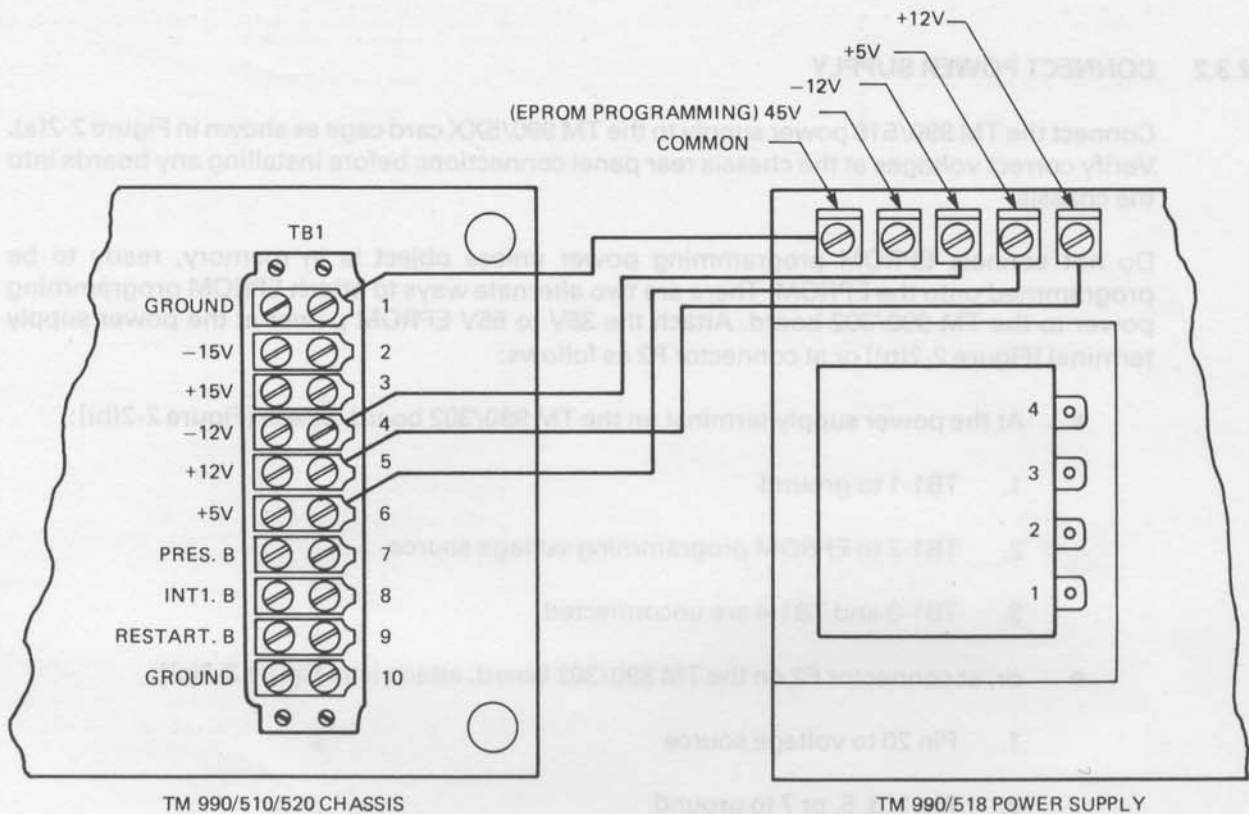
a) Power Connections to TM 990/510/520 Card Cage
from TM 990/518 Power Supply
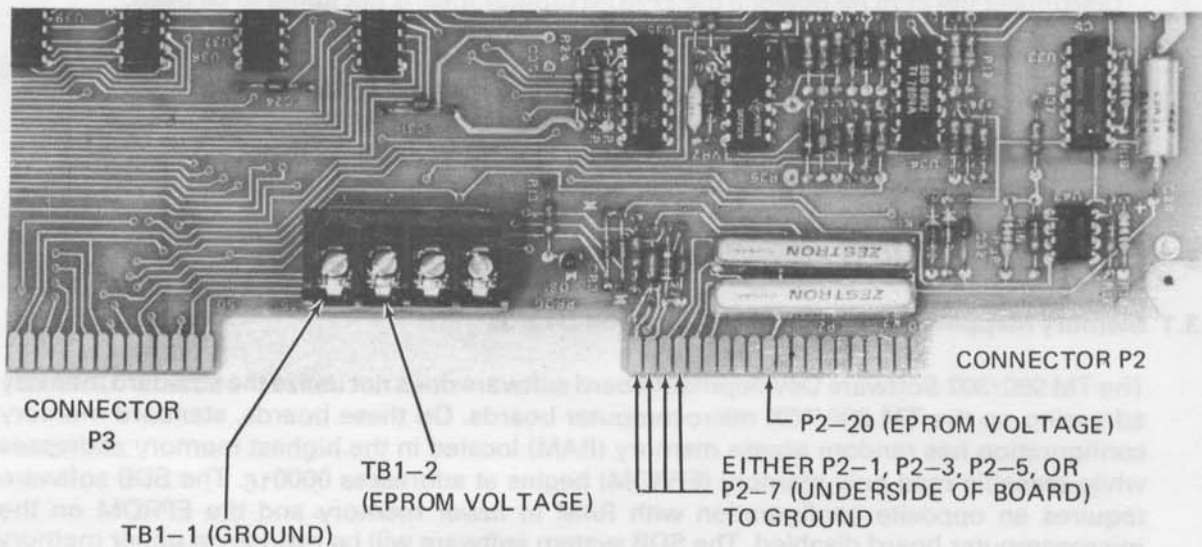
b) EPROM Programming Power Connections to
Terminal TB1 of TM 990/302 Board

c) EPROM Programming Power Connections to
Connector P2 of TM 990/302 Board

NOTE: User can choose which method of attaching EPROM programming voltage (b or c) is preferred.

Figure 2-2. Power Connections To The TM 990/302 Board

Figure 2-3 Memory Configuration Diagram

(a) USING TM 990/101M

- 0_{16} — RAM (TM 990/101M MICRO-COMPUTER BOARD)
- 1000_{16} — RAM
- 2000_{16} — OFFBOARD EXPANSION MEMORY (TM 990/302 BOARD)
- $E000_{16}$ — EPROM
- EPROM
- $FFFF_{16}$

(b) USING TM 990/100M

- 0_{16} — RAM (TM 990/100M MICROCOMPUTER BOARD)
- 40_{16}
- 1000_{16} — RAM
- 2000_{16} — OFFBOARD EXPANSION MEMORY (TM 990/302 BOARD)
- $E000_{16}$ — EPROM
- EPROM
- $FFFF_{16}$
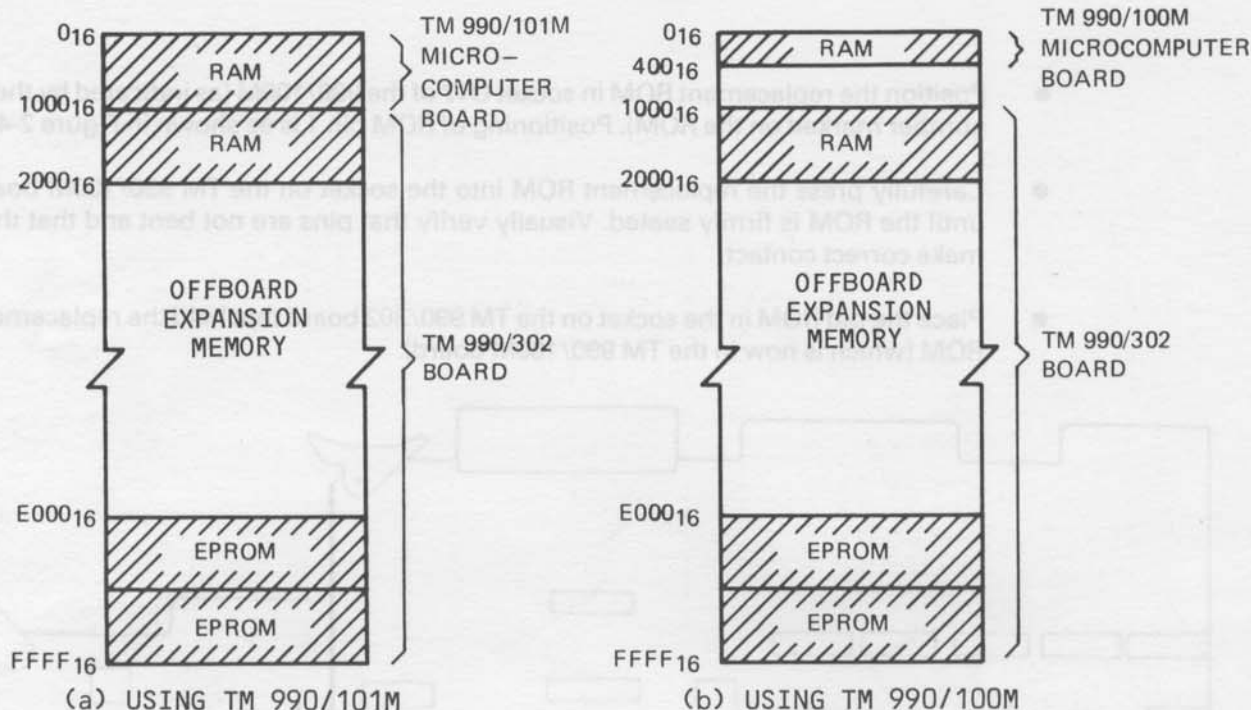
Figure 2-3. Memory Configuration For The Software Development System

Note that the entire onboard EPROM area of both microcomputers is eliminated from the system memory map. The EPROM area on the TM 990/100M board is disabled by a new address decode ROM, and the EPROM on the TM 990/101M board is located in upper memory but not accessed by the system.

2.3.3.2 Advantages of RAM Memory Expansion

An expanded RAM memory will facilitate better use of the Text Editor and Program Debugger as well as other software development programs. Program segments assembled by the Symbolic Assembler will be the same size as without expanded memory; however, a larger symbol table will be allowed. Expanded memory should begin at address 2000_{16}. See your memory expansion board user's guide for correct switch and jumper settings, and proper installation into the system card cage.

2.3.3.3 Preparing the TM 990/100M Microcomputer Board

On the TM 990/100M board, a ROM address decoder must be replaced and jumpers set to the configuration as shipped from the factory except for jumper J11 which is installed for a 20 mA current loop terminal or disconnected (factory ship configuration) for an RS-232-C terminal; this is shown in Table 2-1. After verifying jumper placement, replace the ROM in socket U17 of the TM 990/100M. This new ROM address decoder changes the addressing scheme so that the microcomputer RAM is in lower memory and microcomputer EPROM is disabled. The new decode ROM is shipped (unconnected) on the TM 990/302 board, installed in storage socket XU12. ROM placements are shown in Figure 2-4. Replace the ROM as follows:

- Remove the replacement ROM from socket XU12 on the TM 990/302 board, and remove the ROM supplied with the TM 990/100M in socket U17; temporarily place the latter ROM in a convenient location. Note that the new ROM replacement is marked "2212017 U12" and the ROM to be replaced is marked "991575 U17".

- Position the replacement ROM in socket U17 of the 990/100M (as indicated by the U number marked on the ROM). Positioning of ROM pin 1 is as shown in Figure 2-4.

- Carefully press the replacement ROM into the socket on the TM 990/100M board until the ROM is firmly seated. Visually verify that pins are not bent and that they make correct contact.

- Place the old ROM in the socket on the TM 990/302 board that held the replacement ROM (which is now in the TM 990/100M board).
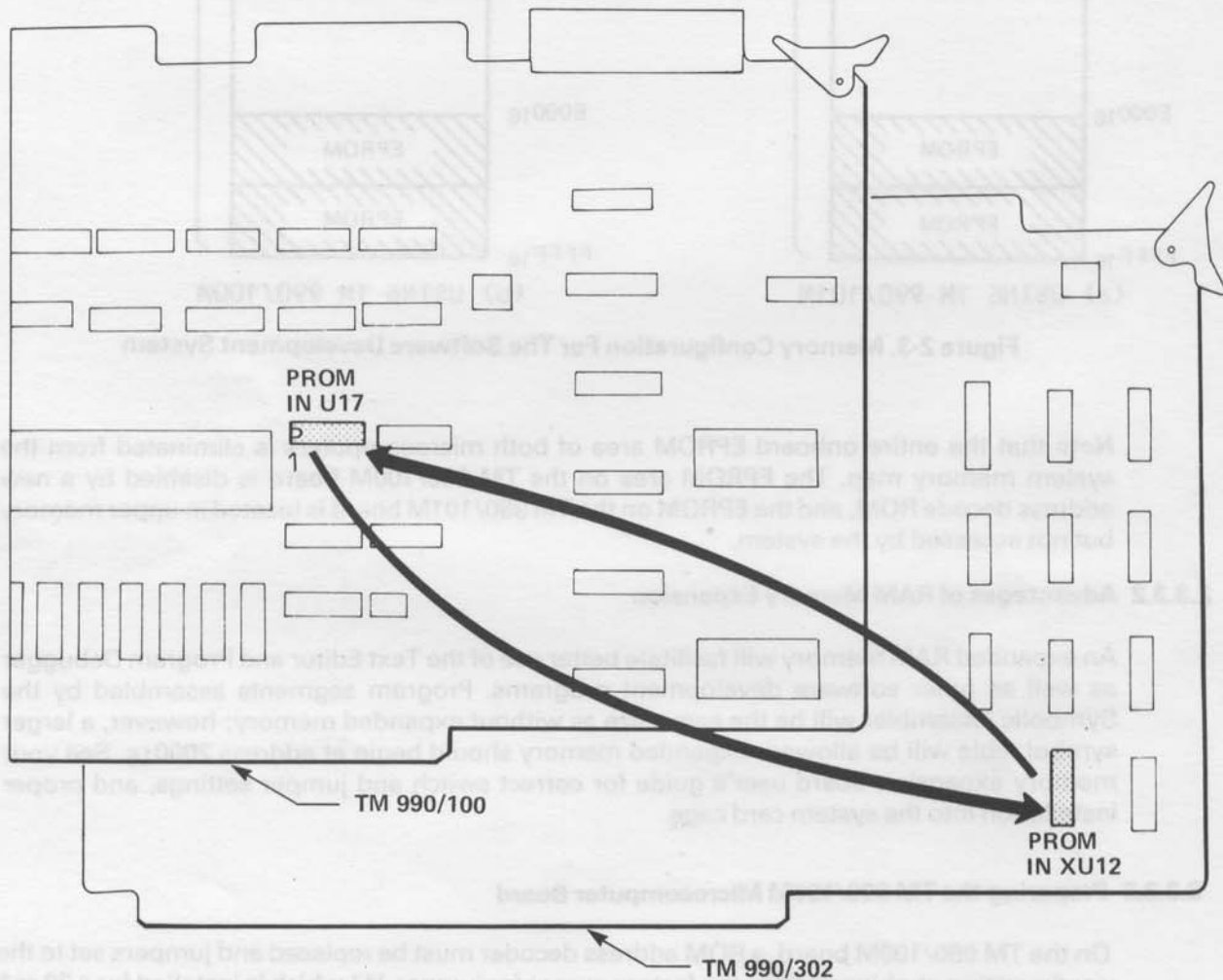
PROM EXCHANGE PROCEDURE:

1. Remove PROM in socket U17 on TM 990/100M microcomputer board.
2. Remove PROM in socket XU12 of TM 990/302 board and insert it into socket U17 of TM 990/100M board.
3. Insert PROM removed from U17 of TM 990/100M board into socket XU12 of TM 990/302 board for safekeeping.

Figure 2-4. Address ROM Changeout At TM 990/100M Board

TABLE 2-1. JUMPER CONNECTIONS ON THE TM 990/100M BOARD

JUMPER	PURPOSE	SET TO POSITION*
J1	TMS 9901 interrupt	P1-18**
J2,J3,J4	EPROM type	DC
J5,J6,J8	Multidrop interface	DC
J9,J10,J12		
J7	EIA/Multidrop select	EIA
J11	EIA/20 mA Current loop select	Install for 20 mA, Disconnect for EIA**
C5	RESTART Delay	Not Installed**

*DC = Don't care; no change required for use in TM 990/302 configuration.
**Position as shipped at factory.

TABLE 2-2. JUMPER CONNECTIONS ON THE TM 990/101M BOARD

JUMPER	PURPOSE	SET TO POSITION*
E1-E2/E2-E3	INT4 From TMS 9902 (Local)	E1-E2
E4-E5/E5-E6	INT5 From TMS 9902 (Remote)	E4-E5
E7/E8/E8-E53	Wait State For Onboard EPROM	DC
E9-E10/E10-E11	TMS 2708/TMS 2716 Memory Mapping	DC
E12-E13/E13-E14	Enable/Disable Onboard EPROM	E12-E13**
E15-E16/E16-E17	RAM/EPROM Mapping	E15-E16**
E18-E19	Pin 1 Of P3 Connected To Ground	DC
E20-E21	Microterminal Power +5V	DC
E22-E23	Microterminal Power +12V	DC
E24-E25	Microterminal Power −12V	DC
E27-E28/E29-E30	EPROM Is TMS 2708	DC
E26-E27/E28-E29	EPROM Is TMS 2716	DC
E32-E33/E34-E35	Expansion EPROM Is TMS 2708	DC
E31-E32/E33-E34	Expansion EPROM Is TMS 2716	DC
E36-E37	Teletype Terminal at P2	(E36-E37 If TTY Required)
E38-E39	Multidrop At Local Port	DC
E39-E40	EIA Or TTY At P2	E39-E40**
E54-E55	Port P3 EIA Compatible	E54-E55**
E55-E56	Port P3 Modem Compatible	Not Installed
C25	RESTART Delay	Not Installed**

*DC = Don't Care; no changes required for use in TM 990/302 configuration.
**Position as shipped at factory.

TABLE 2-3. JUMPER CONNECTIONS ON THE TM 990/302 BOARD

JUMPER	PURPOSE	SET TO POSITION
E1-E2	Causes Wait State In Access Of Slow Memories*	E1-E2**
E2-E3	Does Not Cause Wait State	Not Installed**
E4-E5	Load Function Enabled (Load Vectors In Upper Memory)	E4-E5**
E5-E6	Load Function Disabled	Not Installed**

*Wait state requirement depends on variables listed in Table 2-4.
**Position as shipped at factory.

TABLE 2-4. WAIT STATES REQUIRED FOR MEMORY SPEED AND SYSTEM CLOCK*

MEMORY ACCESS (ns)	3 MHz	4 MHz
450	Wait	Wait
300	No Wait	Wait
200	No Wait	No Wait

*Jumper E2 − E1 = wait, E2 − E3 = no wait

2.3.3.4 Preparing The TM 990/101M Microcomputer Board

On the TM 990/101M board, a new decode ROM is not necessary because the memory decode changes (RAM in lower memory, onboard EPROM disabled) are caused by jumper changes (Table 2-2). On the TM 990/101M, insert jumper E12-E13 to disable onboard EPROM (TIBUG is not used), and insert jumper E15-E16 to reposition RAM/EPROM addressing so that RAM is in lower memory and EPROM is in upper memory. If a teletypewriter is attached to port P2, insert a jumper at E36-E37. All other jumpers are as installed at the factory, indicated in Table 2-2.

2.3.3.5 Install The Microcomputer Board Into the Card Cage

Verify that power is *not* applied to the card cage. Install the microcomputer board into the chassis.

2.3.3.6 Set Up And Install TM 990/302 Board

Memory mapping of the TM 990/302 board can be changed by settings of S1, a four position DIP switch, and a solderable jumper socket on the TM 990/302 board. For most applications, the memory mapping will be for EPROM on the TM 990/302 containing system software, with a memory map as shown in Figure 2-3. For this configuration, switch S1 and the two jumpers should be as shipped from the factory:

- Switch S1: all four switches set to ON.

- Jumper sockets: E1-E2 and E4-E5 as shown in Tables 2-3 and 2-4.

With settings as desired, install the TM 990/302 board into the card cage. Memory map changes are covered in detail in paragraph 2.9.1 of the TM 990/302 Hardware User's Guide.

2.3.4 ATTACH TERMINAL, RUN PRELIMINARY SYSTEM CHECK

Connect a cable from the data terminal to connector P2 of the microcomputer as shown in Figure 2-5. Terminal cable numbers are listed In Table 2-5. Appendices A and B describe cabling for Texas Instruments Model 733/745 and TTY Model 3320/5JE data terminal. At this point, the system configuration can be checked. Apply power to the card cage backplane and data terminal.

CAUTION

Before applying power, check that voltages at the power supply are as specified in paragraph 1.5 and are connected as shown in Figure 2-2.

With power applied, check system operation at this point using the following procedure:

(1) Actuate the RESET switch on the microcomputer (right side facing the card cage).

(2) Press the CR (carriage return) key at the system terminal.

(3) The terminal should respond with a period (.) and a bell indicating at this point the Software Development Board monitor is executing and the system is correctly connected.

If the system monitor does not respond, recheck cabling, jumper connections, and ROM placement (e.g., correct pin positioning), then restart the system by reexecuting steps (1) to (3) above. When the system monitor executes, indicated by a period (.) on the terminal, do not proceed further, but remove power and proceed to the installation of tape recorders, which follows.

TABLE 2-5. AVAILABLE TERMINAL — MICROCOMPUTER CABLES

CABLE NUMBER	CONNECTS
TM 990/501	Connector Kit For Custom Wiring
TM 990/502	RS-232-C Terminal
TM 990/503	Texas Instruments 743/745 Terminal
TM 990/504	Model 33 ASR Teletypewriter Modified For 20 mA Current Loop
TM 990/505	Texas Instruments 733 ASR Terminal

2.3.5 CONNECT THE TAPE RECORDERS TO THE TM 990/302

Figure 2-5 shows the connections between the audio tape recorder/players and the TM 990/302 board. Although the system can be operated with one recorder, a system operates optimally with two tape recorders. Connect the tape recorders as follows:

(1) Tape recorders operate in either a playback or record mode. The TM 990/508 cable joins the TM 990/302 board (connector P2, right side viewed from card cage front) to one or two recorders. At the recorder(s), attach the four labeled leads of the TM 990/508 cable (T1 to T4) as follows:
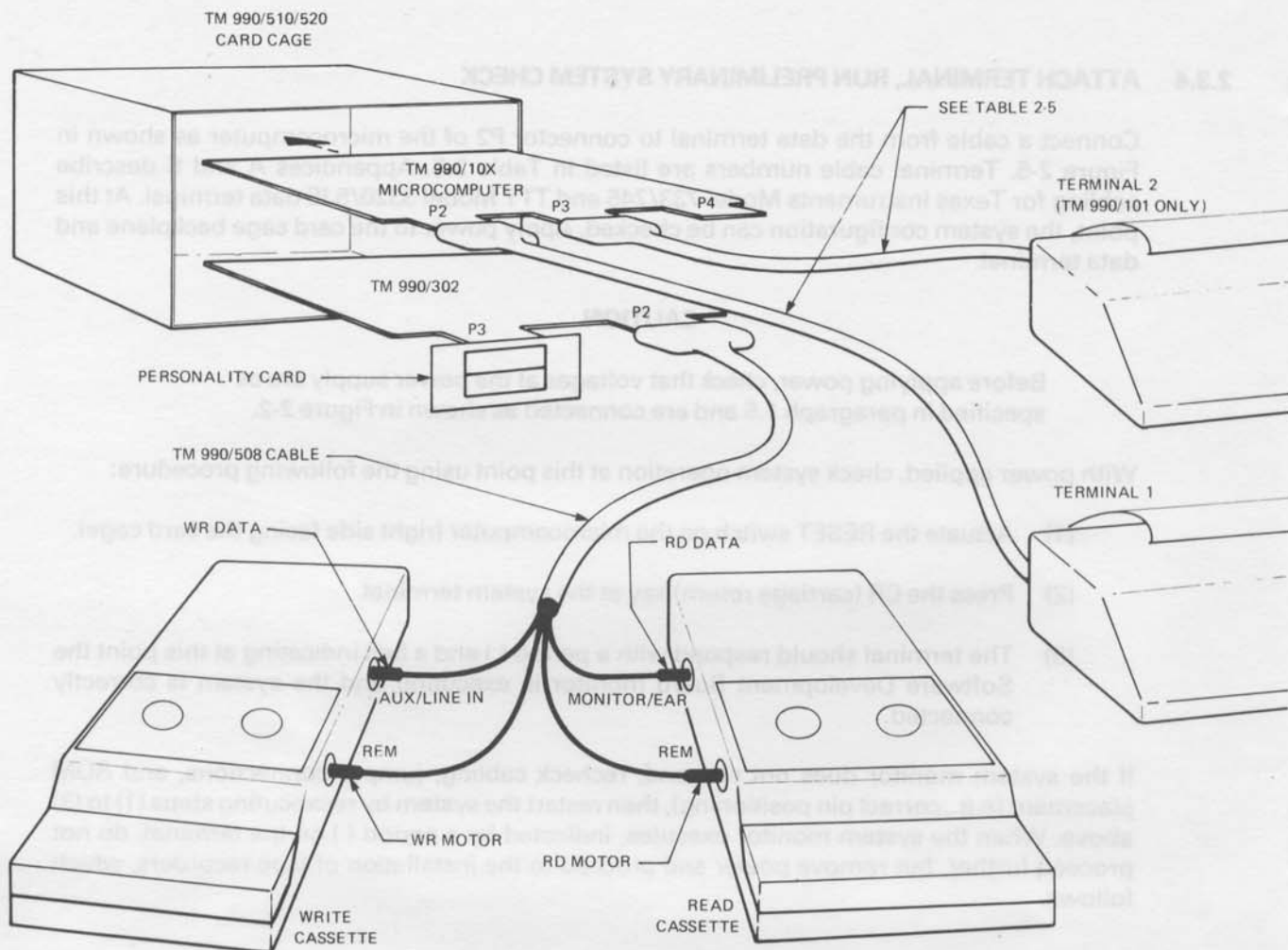
Figure 2-5. System Cabling Between Boards And Peripherals

    a.   At the record cassette unit, attach TM 990/508 leads as follows:

- Lead marked "WR MOTOR" to the motor control jack (REMOTE) of the record unit.

- Lead marked "WR DATA" to the auxiliary input jack (AUX) of the record unit.

    b.   At the playback cassette unit, attach TM 990/508 leads as follows:

- Lead marked "RD MOTOR" to the motor control jack (REMOTE) of the playback unit.

- Lead marked "RD DATA" to the earphone output jack (EAR or MONITOR) of the playback unit.

NOTE

A schematic of the TM 990/508 cable is provided in Appendix I.

(2) Attach the female connector of the TM 990/508 cable to edge connector P2 on the right side (facing card chassis) of the TM 990/302.

CAUTIONS

1. The TM 990/508 cable is keyed to fit only on connector P2 of the TM 990/302 board. If an unkeyed connector is used, verify that the audio cassette cable is connected only to P2 of the TM 990/302 board and not mistakenly to the microcomputer board. Damage can occur if mistakenly connected.

2. Do not operate the tape recorders on battery power as this could cause varying motor speeds in recording and playback, resulting in erroneous data.

3. Do not read data from cassettes while an EPROM is inserted in the personality card attached to the TM 990/302. This could mistakenly program bits on the EPROM.

(3) Insert high-quality tapes into the recorder/players. Rewind the cassettes.

NOTES

1. Texas Instruments cannot guarantee system performance when using substandard tapes. Use only the highest quality tapes to ensure proper performance. Paragraph 2.3.1 contains recommended tapes.

2. If a digital cassette tape is used, avoid recording in the area of tape with the beginning-of-tape hole.

(4) Turn on the tape recorder/players.

(5) Set the tape machine controls as follows:

- volume control to between 50 to 70 percent full volume.

- tone control to between 80 to 100 percent treble.

The correct settings will be indicated by illumination of the sensing LED on some tape machines during data transfer.

CAUTION

Do not change the volume or tone control during a read or write operation; this will probably result in transmission of erroneous data.

2.4 PHYSICAL DEVICE NUMBERS (DEVNOS)

The 302 SDB monitor anticipates that external physical devices are connected to the system at prescribed locations (system connections shown in Figure 2-1). These device locations are shown in Table 2-6.

Physical device numbers (DEVNO's) are used to describe to the system software the following:

● what physical devices (e.g., terminal, cassette recorder/player, etc.) are connected to the system

● the reserved use for each physical device.

Before the system calls up any software development program, it must be given the system configuration. This is accomplished with DEVNO's during the program call. DEVNO's actually indicate if specified physical devices are connected to predetermined I/O conectors on the microcomputer and SDB board. These physical devices are listed in Table 2-6 along with which connector to which these devices must be connected. Table 2-7 lists each DEVNO and its corresponding connector and physical device.

DEVNO's are provided to the system when the particular SDB program is called up as in paragrah 2.6.

TABLE 2-6. DEVICE LOCATIONS/CONNECTIONS IN SYSTEM

DEVICE	CONNECTION	DEVNO
System Terminal	Connected to connector P2 of the microcomputer	1
Cassette Player	Motor controlled through plug marked RD MOTOR of the /508 cable (connected to P3 of the /302 board)	2
Cassette Player	Motor controlled through plug marked WR MOTOR of the /508 cable (connected to the /302 board)	3
Auxiliary Terminal	Connected to P3 of microcomputer board (TM 990/101M only)	4

TABLE 2-7. DEVNO DESCRIPTION

DEVNO	DESCRIPTION
0	Dummy. Used when DEVNO number is required but device is not present or not required.
1	System log (terminal) connected to P2 of microcomputer board.
2	Cassette unit with motor control attached to RD MOTOR plug of /508 cable connected to P2 on /302 board. RD DATA plug is not affected; it *must be inserted* in EAR or MONITOR jack.
3	Cassette unit with motor control attached to WR MOTOR plug of /508 cable connected to P2 on /302 board. WR DATA plug is not affected; it *must be inserted* in AUX or LINE IN jack.
4	Auxiliary terminal optionally connected to P3 of TM 990/101M board (this microcomputer board only).

2.5 SYSTEM INITIALIZATION AND MONITOR CALL

With the system properly installed and connected, the SDB monitor can be entered after power is applied by (1) actuating the RESET switch (right side of microcomputer board when facing chassis) and (2) pressing the CR (carriage return) key on the system terminal. The system should respond with a period (.) and a bell on the system terminal.

NOTE

The EIA interface data format is fixed for 1 start bit, 2 stop bits, even parity and 7 data bits. The baud rate is set up by software detecting the bit width of the ASCII carriage return that user will type in after power up. The available baud rates are: 110, 300, 1200, 2400, 4800, 9600 and 19,200 Hz.

2.6 CALLING SDB PROGRAM

Call the desired SDB program by responding with one of the following mnemonics to the SDB monitor prompt (a period). The call is completed with a carriage return:

Mnemonic	SDS Program
TE	Text Editor (Section 3)
SA	Symbolic Assembler (Section 4)
RL	Relocating Loader (Section 5)
DP	Debugging Package (Section 6)
EP	EPROM Programmer (Section 7)
DM	Dump Memory (Section 8)
SR	2nd EIA Baud Rate (Section 9)
UL	Uplink (Section 11)

Examples:

.TE 0,2   Initial input of source program to Text Editor from keyboard; output source to cassette unit with motor controlled by RD MOTOR plug and data sent via WR DATA plug.

.TE 2,3   Call Text Editor, read source from playback cassette (DEVNO 2); output edited source to record cassette (DEVNO 3)

.SA 2,3,1   Call Symbolic Assembler, read source from playback cassette (DEVNO 2), write object to record cassette (DEVNO 3), print listing on system terminal (DEVNO 1)

.SA 2,3,4   Call Symbolic Assembler, read source from playback cassette; write object to record cassette, and print listing on terminal at auxiliary port of TM 990/101M (DEVNO 4)

.TE 2,4 or   Read source from either cassette and print on the terminal at the
.TE 3,4   auxiliary port of the TM 990/101M (DEVNO 4). DEVNO 4 should operate at 2400 baud or less and be able to understand tab characters. A "Q" command is the only command given under the Text Editor (no editing is attempted).

2.7   RECORDER/PLAYER PROTOCAL

The SDB software does not perform tape rewind operation; thus the user is responsible for rewinding the tape before a read or write operation, and also for loading the tape to the area of magnetic oxide before a write.

In a read operation, the software looks for a header which is an area of tape containing code identifying the start of data. Data following the header code is considered to be valid recorded data.

Before writing on the tapes, the tapes must be rewound to the clear leader, then brought forward over the magnetic oxide area, erasing the magnetic oxide area of any possible header coding from previous write operations. The write operation begins with writing the header for the to-be-recorded data.

Before a read operation, rewind the tape to the clear leader. Software will find the start of valid data by reading the clear leader and magnetic oxide areas (erased) until the header is read; the data following the header is considered valid data.

Note that before tape can be moved using the recorder/player keys, the plug to the motor control jack (REMOTE) must be removed.

2.7.1   SETTING UP FOR A WRITE OPERATION (TWO-CASSETTE OPERATION)

The following assumes a two-cassette operation using the TM 990/508 cable.

(1)   Unplug the WR MOTOR plug from the record unit (REMOTE jack).

(2)   Rewind the cassette to clear leader.

(3)   With the RD DATA plug installed (microphone disabled), press the RECORD key to advance tape into magnetic oxide area; this also erases the tape. Positioning at magnetic oxide area can be ascertained by timing the run of tape or by viewing tape through the clear window of the cassette case until the magnetic oxide area appears.

(4)   Reinsert the WR MOTOR plug (into REMOTE jack) to stop the tape (RECORD key remains depressed).

The recorder is now ready for a software command to write to the recorder.

2.7.2   SETTING UP FOR A READ OPERATION (TWO-CASSETTE OPERATION)

This assumes that clear leader and erased tape precede the data header and that TM 990/508 cable is used.

(1)   Plug the RD DATA plug into the MONITOR/EAR jack.

(2)   Unplug the RD MOTOR plug from the playback unit (REMOTE jack).

(3)   Rewind tape to clear leader.

(4)   Reinsert the RD MOTOR plug into the playback unit (REMOTE jack).

(5)   Depress the PLAYBACK key at the playback unit.

The playback unit is now ready for a software command to read data from its cassette.

2.7.3   ONE-CASSETTE OPERATION

In a one cassette operation, the same DEVNO should be used for the read cassette as well as for the write cassette (with the assembler, DEVNOs must be the same). This allows the use of one motor control plug for both operations (as explained in Tables 2-6 and 2-7, DEVNO's 2 and 3 refer to the motor control plug). For instance, in a text edit call, DEVNO 2 can be specified as both the source input and the edited source output device, meaning that source records will be read in from a recorder/player and edited records will be written to the same device. The following call will be used:

.TE 2,2

In this situation, when changing from a read operation to a write operation (or vice versa), the RD MOTOR plug will be used for both operations (if DEVNO 3 was specified, the WR MOTOR plug would be used for both operations).

Both data plugs can be left plugged into the unit at the same time.

●   To be able to read data, plug the RD DATA plug in the EAR or MONITOR Jack

●   To be able to write data, plug the WR DATA plug into the AUX or LINE IN jack

For example, a text edit operation requires reading source from cassette, then writing edited source back to the same tape. Using DEVNO 2 for both motor controls, set up as follows:

●   Rewind cassette to clear leader.

●   Plug WR DATA into the AUX/LINE IN/MIC jack of cassette player.

●   Plug RD DATA plug into EAR or MONITOR jack of cassette player.

●   Plug RD MOTOR plug into REM jack of cassette player.

●   Press PLAYBACK key on cassette player.

On command, software will read in the source lines. After editing, set up as follows to write out the edited source statements:

●   Press the STOP key on the cassette player.

●   Unplug the RD MOTOR plug from the cassette player.

●   Rewind the cassette tape.

- With the WR DATA plug into the cassette recorder (this disengages any microphone but allows the erase mechanism to function when tape is moved), press the RECORD key(s) at the cassette recorder. Verify that the magnetic oxide area of the tape is brought past the record head. The erase function of the recorder will ensure that previous header data will be erased, preventing mistakes during later read operations.

- When the tape is properly located, insert the RD MOTOR plug into the REM jack of the cassette recorder; this will stop the tape movement leaving the cassette recorder ready to accept data via the text editor software.

After writing the edited source on tape, the user can read in the edited source lines. Set up as follows (note WR DATA and RD DATA plugs are still inserted):

- Press the STOP key at the recorder.

- Unplug the RD MOTOR plug.

- Rewind the tape to clear leader.

- Insert the RD MOTOR into the REM jack.

- Press down the PLAYBACK key on the cassette player.

When commanded by software, the data on cassette will be read.

CAUTION
In one-cassette operation, it is sometimes necessary to change cassettes without rewinding them. Exercise care to prevent any movement of the tape within the cassette while the cassette is outside the recorder/player; movement of the tape can result in loss of data.

3.1 GENERAL

The TM 990/302 Text Editor provides a means for initially generating source-program state-ments, storing them on cassette, and then later editing changes into these source statements. Interactive communication from the terminal allows the user to (1) generate a new source-statement file or (2) edit a previously generated source-statement file contained on cassette. In either mode, the source statements can be edited as required, then written to cassette for assembly by the assembler.

As an aid in using this manual, a single program that can be loaded and executed by the reader on the TM 990/302 will be used to explain the system software. This program will cause the LED on the TM 990/302 board to blink (shown in Figure 1-1). The writing of this program begins in this section, in its inception as source statements. Later we will follow along this theme in assembling, loading, and executing the program.

Text Editor comands are as follows:

- G command: Get source lines from source DEVNO device (paragraph 3.5.1)

- P Command: Print source line(s) on system terminal (paragraph 3.5.2)

- Insert source line(s) within text command (paragraph 3.5.3)

- Delete source line(s) from text command (paragraph 3.5.4)

- R command: Resequence line numbers using specified increment (default incre-ments are 10: e.g., 010, 020, 030, etc.; paragraph 3.5.5)

- K Command: Keep line(s) in memory; write these to the destination DEVNO device and input source lines from the source device (paragraph 3.5.6)

- Q command: Quit Text Editor mode, write remaining source lines in memory and exit from Text Editor and reenter monitor (paragraph 3.5.7).

3.2 SYSTEM CONFIGURATION

Minimum configuration includes a system terminal, the TM 990/302 board, a microcomputer board, and one cassette player/recorder. One cassette can be used in a dual role: container of the source statements to be input and edited, and storage for the edited source statements. The optimum system configuration includes two audio cassettes, one for the source cassette (containing source statements to be edited) and one for the destination cassette (to receive editd source code). See Section 2 for detailed data on system configuration.

3.3 CONSIDERATIONS

- The maximum program size which can be loaded into memory is limited by memory space. Program segment size is determined by the number of characters written to the destination DEVNO device during a K (Keep) or Q (Quit) command.

- IDT directive <u>must begin</u> all programs (see Figure 2.10).

- Memory space can be economized by judicious use of (1) comments following source statements and (2) space between source statement fields. The latter is facilitated by the horizontal tab feature of the Text Editor (paragraph 3.4.2).

- The <u>only</u> exit back to the monitor is through the Quit command. The ESC key will <u>not</u> exit to the monitor.

3.4   TEXT EDITOR CALL AND SPECIAL KEY FUNCTIONS

3.4.1   TEXT EDITOR CALL

To call the Text Editor, respond to the monitor prompt with the following (see paragraph 1.7):

$$.TEΛ<\underline{DEVNO}> \begin{Bmatrix} Λ \\ , \end{Bmatrix} <\underline{DEVNO}> <(CR)>$$

DEVNO containing source to be edited ⟍   ⟋ DEVNO that receives edited source

When source lines will not be brought in from cassette but will be initially input from the keyboard (i.e., a Get command is not heeded), the input DEVNO is 0 (zero for dummy).

Examples:

.TE 2,3	Input source lines from RD MOTOR cassette; write edited source to WR MOTOR cassette.
.TE 0,2	For input of source from keyboard; note that DEVNO 0 is used for keyboard input which also disables get command; output edited source to RD MOTOR cassette.
.TE 2,2	Use one cassette (RD MOTOR cassette); input entire program segments from this cassette; output edited segments back to this cassette.
.TE 2,4	Input from RD MOTOR cassette; write output to terminal connected to P3 of TM 990/101M.

When initialized, the Text Editor will issue a question mark (?) prompt asking for user input of Text Editor commands, explained in subsection 3.5.

.TE 2,3

.? ◄————————— Prompt for Text Editor command

3.4.2   SPECIAL KEY FUNCTIONS

3.4.2.1 Use Tab Function To Space Between Source Fields (Control I)

To tab between the label, opcode mnemonic, operand, and comment fields of the source statements, press the I key while the control (CTRL) key is pressed. This will automatically place the required spacing between source fields when later printed as a source listing or by the print command; however, initially the tab will create a single space. This function economizes memory space.

3.4.2.2 Use CONTROL H and F Commands To Correct Characters

To correct a character already input on a line, press the H key while the CONTROL (CTRL) key is pressed. For each CONTROL H entered, the printing mechanism will backspace one character. When at the character to be changed, insert the correct character. To return to the end of the line, use the CONTROL F command (press the F key while the CONTROL key is pressed). For each CONTROL F entered, the printing mechanism will move forward one character. Instead of using the CONTROL F function, the user can retype the characters that were backed over.

3.5 COMMANDS

All commands are comprised of one character with no, one, or two arguments. All are terminated by a carriage return.

3.5.1 GET SOURCE LINES COMMAND (G)

This command brings one program segment of previously edited source lines into memory from the input device designated in the Text Editor call and then prints the number of the last line brought into memory. Once in memory, the source lines can be edited using the other Text Editor commands. The bounds of one program segment are set by the Text Editor Keep (K) or Quit (Q) command; each command writes one program segment to the source line output device. Before reading in source lines in a one-cassette operation, it issues a prompt to the user asking if the cassette is ready to be read from. In a two-cassette operation, prompts are not issued and the command executes after the carriage return. Format (see paragraph 1.7):

?G <(CR)>

Example:

?G

**SWAP TAPES

**HIT 'CR'TO GO — } Prompts issued in single-cassette operation only

900 ◄──────── Number of last statement brought into memory

? ◄──────── When program segment read, prompt issued

NOTE

Source lines can be inserted from the keyboard into the memory buffer before bringing in source lines from the cassette with the Get command. When this is the case, the first line number on the cassette must be a higher number than the highest line number inserted from the keyboard. Keyboard insertion is explained in paragraph 3.5.3. Errors are explained in paragraph 3.6.

3.5.2 PRINT SOURCE LINES COMMAND (P)

This command causes designated source lines to be printed on the system terminal. A four-digit source line number, printed to the left of each line is used to uniquely reference each line. The command consists of the mnemonic P followed by either no, one, or two numbers (line numbers in decimal) and a carriage return. A no-number command will print out the first line in the buffer. A one-number command will print out the line specified. A two-number command will print the lines from the first line number to the second line number; the first line number must be a smaller number than the second line number. If the first line number is not specified (i.e., a space entered), its default value will be the first line number in the buffer. Printing of lines can be stopped/started by depressing the terminal space bar.

?P [Source Line] $\left\{ {\scriptstyle \Lambda \atop ,} \right\}$ [Last Source Line] <(CR)>

———————————— optional last line of multiline print

First (or only) line number ————————————>

Examples:

(1) ?P ◄——————————— Finish with carriage return

    100   IDT   'BLINK' ◄— First line in memory buffer

    ? ◄——————————————— Return to command scanner

(2) ?P100   Print source line 100

    100   IDT   'BLINK'

    ?

(3) ?P100,140   Print source lines 100 to 140

    100   IDT   'BLINK'

    110 BEGN   LWPI >1000   WP ADDR

    120   LI   R12,>1706 LED CRU ADDR

    130 STRT   MOV   R1,R1   R1 ALL ZEROES?

    140   JEQ   ON   YES, LIGHT LED

    ? ◄—————————————— Prompt for next command

3.5.3 INSERT SOURCE LINES COMMAND

This command is used to input source lines from the terminal:

- as initial source-line inputs (first input of program source code)

- between or following existing source lines

● in place of existing source lines (replace existing lines)

Lines are inserted by specifying the line number then following this with the source statement to be inserted. In this manner, the user can designate his own line numbering system. It is suggested that lines be in increments, with the increment value sufficient to allow later insertion of lines without renumbering the existing lines. For example, an increment of ten (lines 100, 110, 120, etc.) would allow later inserting lines 101 to 109, 111 to 119, etc. Note that line numbers will also be the source statement number in the assembly listing.

Source line format is the same as will be displayed in the source listing (from left to right: label field, opcode field, operand field, comment field), with each field separated by at least one space. When inserting source lines, the line number precedes the source statement. The label field begins following this line number or following this line number and a single space. The mnemonic field begins two spaces following the line number (if no label) or one space following the label.

NOTE

It is recommended that all spaces between fields
be inserted with the CONTROL I tab function with
the label field beginning immediately after the
line number. The CONTROL I tab function en-
sures that all fields be aligned in a source listing
or line printing. When inserted, tabs appear
merely as a single space. See paragraph 3.4.2.1.

Figure 3-1 shows the format for inserting a source line. It follows the general source listing format shown in Figure 4-4 (Section 4)
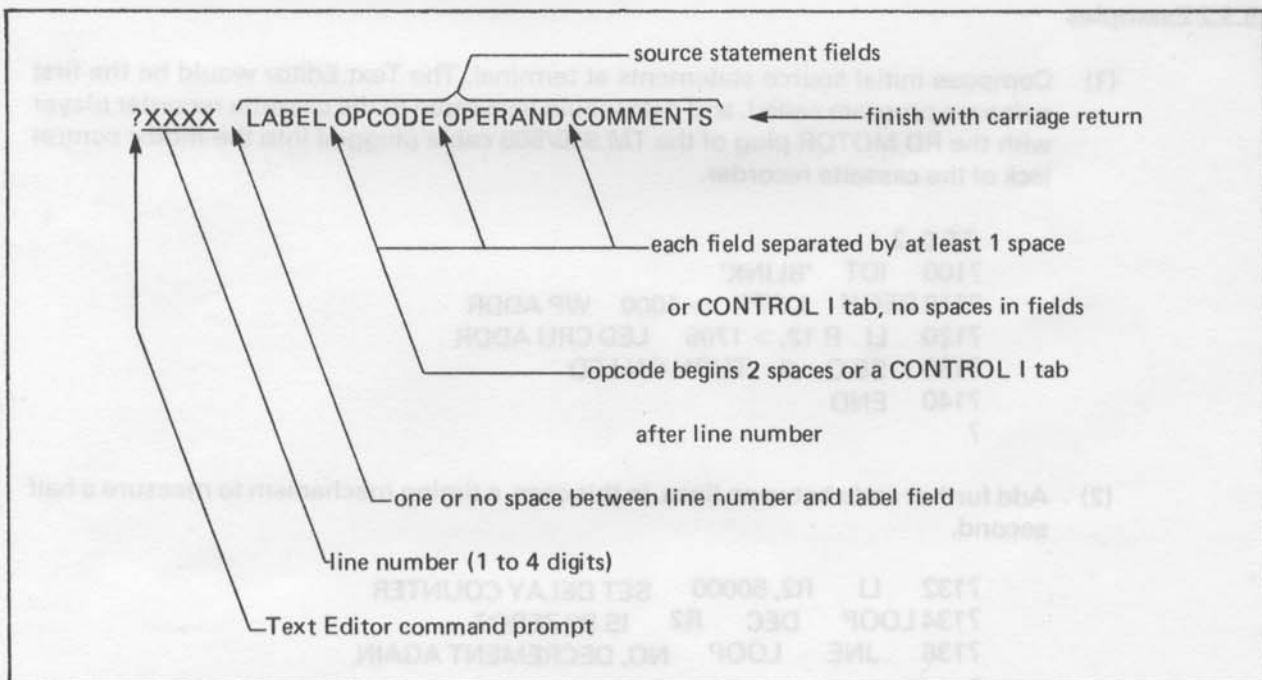
Figure 3-1. Source Statement Fields

3-5

3.5.3.1 Initial Source Input

When used for initial input of source statements (generated from keyboard, not read in from cassette), the Text Editor is the first software development program called. In this case, merely call the Text Editor, specifying DEVNO 0 (zero) for source input and DEVNO 2 or 3 for source output, then use the insert line process.

CAUTION

If DEVNO 0 is not used for source input, an end-of-file will not be written by the Quit command and inserted source lines will be lost.

To begin lines at line number 100, enter the number 100 followed by the source statement. For example; enter an IDT directive which is <u>mandatory</u> for all programs:

```
                  ┌──two spaces or one CONTROL I indicate first entry is opcode
                  │
?100    ╱   IDT  'BLINK'
       ╱
?110  ╲BEGN   LWPI  > 1000   WP ADDR
       ╲
?        ╲──── one or no space indicates first entry is label
```

Note that the question mark prompt from the Text Editor command scanner appears after each line insertion. At this command level, either additional lines can be inserted or any of the other Text Editor commands can be executed.

3.5.3.2 Examples

(1) Compose initial source statements at terminal. The Text Editor would be the first software program called, and data would be written to the cassette recorder player with the RD MOTOR plug of the TM 990/508 cable plugged into the motor control jack of the cassette recorder.

```
.TE 0, 2
?100   IDT   'BLINK'
?110 BEGN   LWPI  > 1000   WP ADDR
?120   LI  R 12, > 1706   LED CRU ADDR
?130   SB O   0   TURN ON LED
?140   END
?
```

(2) Add further code between lines. In this case, a timing mechanism to measure a half second.

```
?132   LI   R2, 50000   SET DELAY COUNTER
?134 LOOP   DEC   R2   IS R2 ZERO?
?136   JNE   LOOP   NO, DECREMENT AGAIN
?
```

(3)   A printing of the program so far would show the following:

```
?P100,140
100                 IDT    'BLINK'
110      BEGN       LWPI   >1000        WP ADDR
120                 LI     R12,>1706    LED CRU ADDR
130                 SBO    0            TURN ON LED
132                 LI     R2,50000     SET DELAY COUNTER
134      LOOP       DEC    R2           IS R2 ZERO?
136                 JNE    LOOP         NO, DECREMENT AGAIN
140                 END
?
```

(4)   Now it is desired to add some control statements for turning off and LED as well as for turning it on. Insert the following after line 120:

```
?121STRT  MOV  R1,R1   R1 ALL ZEROES?
?122 JEQ  ON   YES, LIGHT LED
?123 JMP  OFF  NO, TURN OFF LED
?124ON SBO  0   TURN ON LED
?125 SETO  R1   SET ON/OFF FLAG
?126 JMP  RUN   GO TO TIMER
?127OFF SBZ  0   TURN OFF LED
?128 CLR  R1   RESET FLAG
```

(5)   A printout of the program would now look like:

```
?P100,140
100                 IDT    'BLINK'
110      BEGN       LWPI   >1000        WP ADDR
120                 LI     R12,>1706    LED CRU ADDRESS
121      STRT       MOV    R1,R1        R1 ALL ZEROES?
122                 JEQ    ON           YES, LIGHT LED
123                 JMP    OFF          NO, TURN OFF LED
124      ON         SBO    0            TURN ON LED
125                 SETO   R1           SET ON/OFF FLAG
126                 JMP    RUN          GO TO TIMER
127      OFF        SBZ    0            TURN OFF LED
128                 CLR    R1           RESET FLAG
130                 SBO    0            TURN ON LED
132                 LI     R2,50000     SET DELAY COUNTER
134      LOOP       DEC    R2           IS R2 ZERO?
136                 JNE    LOOP         NO, DECREMENT AGAIN
140                 END
```

(6)   The timing required for the decrementing timer can now be checked for a correct value. Since the two instructions at lines 0134 and 0136 require a total of 24 clock and memory cycles (at 333 ns each), a half-second blink would take approximately 62,500 loops through those two instructions. This will require a change in the

loop-count value at line 0132. To change, merely insert over the old line 0132.

```
?132    LI    R2,62500    SET DELAY COUNTER
?
```

(7) A printout of line 0132 shows the change was made.

```
?P132
132    LI    R2,63500    SET DELAY COUNTER
?
```

(8) A check of the printout shows that register numbers were preceded by the letter R; however, this has not been set up so that the assembler will recognize these alphanumeric register symbols (it automatically recognizes the number only). Thus, the EQU assembler directive has to be used (assembler directives are explained in Appendix F). This should be inserted before the first use of predefined register numbers, as shown next:

```
?105    R1    EQU    1
?106    R2    EQU    2
?107    R12   EQU    12
```

(9) An AORG assembler directive is necessary to designate program location when loading object into memory (Symbolic Assembler object is absolute, not relocatable). This directive should be added at the beginning of the source code.

```
?101    AORG    >1040
?
```

(10) At the end of the decrement loop at lines 134 and 136, there is a need for some means to repeat the entire sequence with the LED in the opposite mode (lit or extinguished). Thus, there is a need to jump back to the start of the program, reentering it where the R1 flag is checked.

```
?138    JMP    STRT    RESTART PROGRAM
?
```

(11) To have the program ready to execute as soon as it is loaded by the Relocating Loader, a program entry label can be specified in the operand field of the END assembler directive. The program entry label identifies the start of program execution. This address is placed in the Program Counter by the Relocating Loader so that the program can be executed by the EX command of the Program Debugger. Note that by inserting over an existing line, the new line replaces the old.

```
?140    END    BEGN
?
```

(12) Finally, a printout of the program as it is constructed so far:

```
?P100,140
100              IDT     'BLINK'
101              AORG    >1040
```

3-8

```
105        R1      EQU    1
106        R2      EQU    2
107        R12     EQU    12
110  BEGN        LWPI   >1000         WP ADDR
120                LI     R12,>1706     LED CRU ADDR
121  STRT        MOV    R1,R1         R1 ALL ZEROES?
122                JEQ    ON            YES, LIGHT LED
123                JMP    OFF           NO, TURN OFF LED
124  ON           SBO    0             TURN ON LED
125                SETO   R1            SET ON/OFF FLAG
126                JMP    RUN           GO TO TIMER
127  OFF          SBZ    0             TURN OFF LED
128                CLR    R1            RESET FLAG
130                SBO    0             TURN ON LED
132                LI     R2,62500      SET DELAY COUNTER
134  LOOP        DEC    R2            IS R2 ZERO?
136                JNE    LOOP          NO, DECREMENT AGAIN
138                JMP    STRT          .RESTART PROGRAM
140                END    BEGN
?
```

3.5.4 DELETE SOURCE LINES COMMAND

This command deletes source lines one line at a time. When the line(s) are deleted, source line numbers following the deleted lines remain the same until renumbered by the resequence (R) command used with the Keep and Quit commands. A carriage return completes the command.

The format is similar to the insert lines process. In response to the question mark of the Text Editor command scanner, specify the line number to be deleted, and follow this with a carriage return. An erroneous line number can be deleted by retyping the line preceding the line to be deleted and inserting a plus (+) sign after the number of the retyped line (e.g., 220+).

Format (see paragraph 1.7):

? <line number> <(CR)>

Examples:

(1) In our BLINK programming example, notice that line 130 should be deleted and that the next line, 132, needs to be changed in order to show the beginning of the counter; thus lines 130 and 132 can be deleted and substituted with a correct line 132. (Note that this can also be caused by deleting one line and inserting over the other.)

```
?130        ←──────── Delete line 130, finish with a CR
?132        ←──────── Delete line 132, finish with a CR
?P128,134   ←──────── Print from line 128 to 134
128              CLR    R1       RESET FLAG    } Lines 130 and
134  LOOP   DEC    R2       IS R2 ZERO?   } 132 deleted
?130  RUN   LI     R2,62500 SET DELAY COUNTER new line
?    ←──────── control back to command scanner
```

3-9

(2) Lines 0122 and 0123 of the program can be replaced with a single instruction.

?122   JNE OFF   NO TURN OFF LED ◄── Replace line 122 with new line
?123 ◄───────────────────────────── Delete line 123
? ◄─────────────────────────────── Return to command scanner

(3) Finally, a printout of the program as it is constructed so far:

```
?P100,140
  100                IDT    'BLINK'
  101                AORG   >1040
  105        R1      EQU    1
  106        R2      EQU    2
  107        R12     EQU    12
  110        BEGN    LWPI   >1000      WP ADDR
  120                LI     R12,>1706  LED CRU ADDR
  121        STRT    MOV    R1,R1      R1 ALL ZEROES?
  122                JNE    OFF        NO, TURN OFF LED
  124        ON      SBO    0          TURN ON LED
  125                SETO   R1         SET ON/OFF FLAG
  126                JMP    RUN        GO TO TIMER
  127        OFF     SBZ    0          TURN OFF LED
  128                CLR    R1         RESET FLAG
  130        RUN     LI     R2,62500   SET DELAY COUNTER
  134        LOOP    DEC    R2         IS R2 ZERO?
  136                JNE    LOOP       NO, DECREMENT AGAIN
  138                JMP    STRT       RESTART PROGRAM
  140                END    BEGN
?
```

3.5.5 RESEQUENCE LINES BEGINNING AT NEW VALUE (R)

This command begins line numbering at a specified decimal value and renumbers all lines beginning at that value, incrementing each line by ten. This command has a practical application where a series of line inserts and deletes has widely changed line numbering or has filled up the numbers between lines so that insertion by line number is not possible. Renumbering of lines takes place during a Keep or Quit command when lines are written to the ouput buffer. The increment value of ten cannot be changed. A line number can be from 1 to 9999 with numbers from 1 to 8999 incremented by ten and numbers from 9000 to 9999 incremented by one. If a program is large enough to have a line number greater than 9999, the transfer to tape operation is halted, an end-of-file is written to the output cassette, and the Text Editor is exited with an ERROR 32 message output (errors are explained in paragraph 3.6). Data following line 9999 will be lost.

Format (see paragraph 1.7):

?R<first line number in decimal>
└────────── must be less than 9000$_{10}$

For example, if a program has line numbers 112, 113, 115, 119, 120, 121, 127, and 130, and the following is executed:

?R255

the program (after a Keep or Quit) will have line numbers of 255, 265, 275, 285, 295, 305, 315, and 325.

CAUTION

Do not follow the Resequence command with a
Quit command, else only the first program seg-
ment will be written to cassette. Instead, use the
Keep command to record each segment, then use
the Q command to write the EOF.

3.5.6   KEEP (SAVE) EDITED SOURCE LINES COMMAND (K)

This command writes the source lines in the memory edit buffer to the output DEVNO device
(specified in the call for the Text Editor). The Text Editor retains program control (unlike the
Quit command which transfers control to the monitor). The block of lines will be stored on the
output device as a source segment, the entity that will be brought in later as a block by the Get
source lines command (paragraph 3.5.1). The Keep command is terminated with a carriage
return; after which a prompt is printed reminding the user that the correct tape should be on
the write cassette. The software will wait until a carriage return is entered, indicating that the
cassette recorder is ready. If a number is returned after Keep command execution, this is the
first line now in the memory edit buffer. If no number is printed, the memory edit buffer is
empty.

Format (prompt issued only for one-cassette operation):

    ?K<(CR)>

Example:

    ?K
    **SWAP TAPES   ⎫ Prompt: when tape is ready,
    **HIT 'CAR' TO GO—   ⎬ answer with carriage return
    1010 ◄─────────────────── First line in buffer following Keep
    ? ◄───────────────────── Return to Text Editor command scanner

NOTE

Make sure that the cassette unit is correctly
connected, powered up, and ready for operation
before executing the Keep command.

3.5.7   QUIT TEXT EDITING COMMAND (Q)

This command is the only exit from the Text Editor program to the monitor. The Quit
command is similar to the Keep command (paragraph 3.5.6); however, the Quit command
performs the following additional functions:

- if initial input of source lines (DEVNO 0 designated at editor call as input device;
  thus no input from cassette), the Quit command writes an end-of-file mark to
  cassette after the memory edit buffer contents are written to the output cassette
  (does not attempt to read input device).

- if input is brought in from cassette (not initial input), the Quit command reads in the
  remaining input-device program contents up to the end-of-file mark (written there
  by a previous Quit command) and writes these to the output device so that all
  program contents are on the output device.

- the Quit command returns control to the monitor.

3-11

After the Quit command is executed, it should be assumed that the memory buffer is empty of useable data.

Format (see paragraph 1.7):

?Q <(CR)>

. ◄—————— control returns to SDB monitor (period printed)

CAUTION

The end-of-file mark is used by the Text Editor to terminate its casette read operation and return control to the Text Editor command scanner. If an end-of-file mark is not found, control will not be passed to the Text Editor editing commands; thus, it is imperative to use the Quit command to terminate program editing.

3.6   ERROR CODES FOR TEXT EDITOR

3.6.1   **ERROR 21

- Commands: G, K, Q

- Meaning: Error in transmission of source statements from cassette to memory buffer; could be a checksum error or error in data communications format.

- Recovery Procedure: The program segment with the error is lost and will not be written to cassette with a Keep or Quit command. A Quit command will write the remaining program segments to the output cassette; then, the truncated program can be read in again and the lost source records re-entered. Or, after the Quit command, read in the original source cassette (with error) to see if the error reoccurs (any editing in the first session will not be on this cassette). Or, the program segment can be considered lost and the user can continue editing the remaining source segments, finishing the session with a Quit command, without the program segment in error.

3.6.2   **ERROR 22

- Command: G

- Meaning: There is a source line in the memory buffer with a line number larger than the first line number of a segment brought in from cassette. Incoming source lines should be ascendingly greater than line numbers in the buffer.

- Recovery Procedure: Further input of a program segment is prohibited (i.e., Get command is prohibited). Commands other than Get can be executed; however, editing out of the line(s) causing this error will not permit Get command to execute. A quit command causes a return to the monitor without reading in remaining program segments from input cassette. User can re-edit original source on the input cassette or edit the truncated source on the output cassette.

3.6.3 **ERROR 23

- Command: G

- Meaning: Get command cannot be performed because end-of-file (EOF) has been detected; thus complete program file has been read.

- Recovery Procedure: Execute Quit command to write memory buffer contents to cassette.

3.6.4 **ERROR 24

- Command: G

- Meaning: Insufficient space in memory buffer to receive new source records.

- Recovery Procedure: Execute Keep command to write buffer contents to cassette. Re-execute Get command.

3.6.5 **ERROR 25

- Command: All

- Meaning: Text Editor command input is not valid.

- Recovery Procedure: Insert valid Text Editor command.

3.6.6 **ERROR 26

- Command: K, R

- Meaning: Resequencing of line numbers requested after Keep command executed. Entire program through EOF marker must have the same line sequence numbering.

- Recovery Procedures:
    (1) Continue using present sequencing value

    (2) or execute Quit command, reenter Text Editor, and specify new re-sequencing value before first Keep command. Last Resequence command will apply.

3.6.7 **ERROR 27

- Command: P

- Meaning: In Print command, first (beginning) line number to be printed was larger than second (ending) line number.

- Recovery Procedure: Reenter command with first line number smaller than second line number.

3.6.8 **ERROR 28

- Command: P, I

- Meaning: Line number to be printed is too small; a line number of higher value has been sent to the output cassette.

- Recovery Procedure: Reenter command; use a source line number with a value higher than the highest source line number in the output cassette.

3.6.9 **ERROR 29

- Commands: Insert or Change Commands

- Meaning: Command issued to insert or change a line having a higher value than the highest-numbered line in the memory buffer.

- Recovery Procedure: If line to be inserted or changed has a higher number than the highest line number memory buffer, use Get command to read in the program segment containing line numbers in the range desired.

3.6.10 **ERROR 30

- Command: Insertion Command

- Meaning: Not enough memory buffer space for line to be inserted or line to replace existing line.

- Recovery Procedure: Decrease memory buffer contents using a Keep command or the line deletion command. Then attempt the line insertion again. If insertion command also causes a line to be deleted (replace existing line), then line to be replaced was deleted but new line was not inserted in its place.

3.6.11 **ERROR 31

- Command: R

- Meaning: Starting line in the Resequence command is larger than 9000.

- Recovery Procedure: Reissue the Resequence command with a line-start value less than 9000.

3.6.12 **ERROR 32

- Command: R

- Meaning: Resequencing command attempted to generate a line number larger than 9999 during a Keep or Quit function. Maximum line number allowed is 9999.

- Recovery Procedure: When the line number reaches 9999, an end-of-file is written to the cassette, and data following line 9999 is lost. Re-edit source as required and begin resequencing lines with a lower beginning number. The resequencing command increments line numbers from 1 to 8999 by ten, and increments line numbers from 9000 to 9999 by one.

3.7   DATA BACKUP ON CASSETTE

Sometimes it is desireable to make a backup tape of source programs in order to share known source or to have a copy in the case of inadvertant destruction of the master source. To make a backup tape, use the Text Editor with the following commands:

```
.TE 2,3
?Q
.
```

The tape will be copied from the cassette unit controlled by DEVNO 2 to the tape unit controlled by DEVNO 3.

3.8   MULTIFILE CASSETTES

The use of multifile cassettes is discouraged. All files are serial and may be difficult to find via random positioning. With several files per cassette, greater protection must be taken for the cassette.

SECTION 4

SYMBOLIC ASSEMBLER

4.1 GENERAL

The TM 990/302 Symbolic Assembler is a two-pass assembler that interprets the source statements created using the Text Editor. These source statements are assembled into absolute (not relocatable) machine code executable by the TMS 9900 series of micro-computers (See Figure 4-1).

The symbolic Assembler performs the following functions:

- translates the assembly language mnemonics used by the TMS 9900 series of microcomputers

- assembles the two pseudo instructions NOP and RT (see paragraph 4.5); however, memory mapping instructions LMF, LDD, and LDS are not recognized by this asembler

- allows use of at least 136 one- to four-character labels for symbolic memory addressing

- allows comments to follow as part of a source statement

- interprets limited add and subtract expressions used with symbolic addresses (LABL-2, LABL+10, etc.); see paragraph 4.7

- interprets eleven assembler directives to facilitate coding (assembler directives are explained in Appendix F)

- provides assembler listings containing source statement number in decimal, location counter value in hexadecimal, assembler object in hexadecimal, source statement, comments, and error messages (See Figure 4-1)

- assembled object output contains loader tags as well as a checksum code for each object record. Object is in absolute form, not relocatable.

The TM 990/302 Symbolic Assembler is a two-pass assembler. This means that the assembler reads the source code twice. Before each read operation, the source code cassette tape must first be positioned at program beginning before the assembler is executed. During the first pass, the assembler builds a table containing:

- symbols used in operands for addressing

- the address of these symbols.

During the second pass, the assembler resolves the relative addresses and displacements in instructions using these symbols, and generates an object on cassette and hard-copy assembly listing on the system terminal. Figure 4-2 depicts the passes of the two-pass operation.
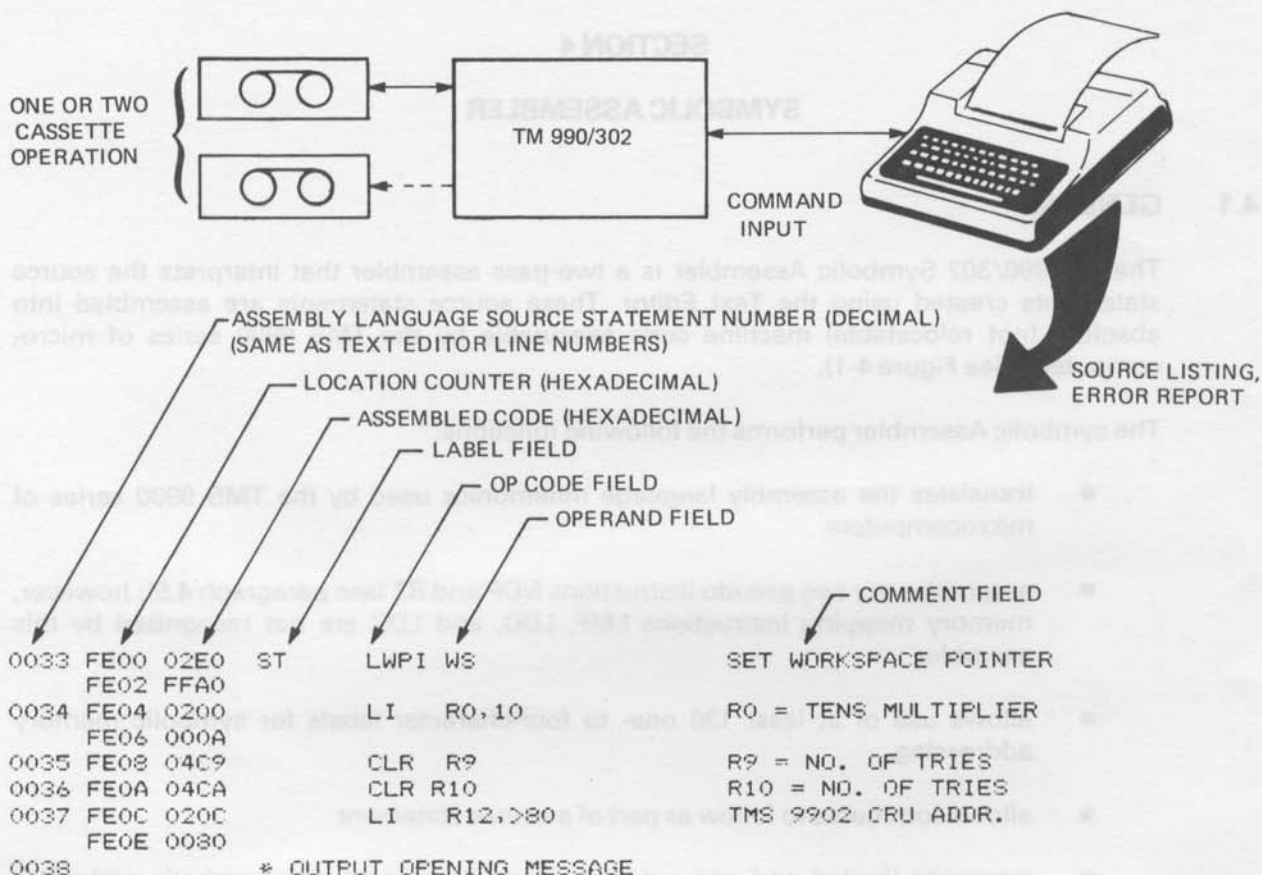
Figure 4-1. Symbolic Assembler

The following labels point to the source listing in the figure:

- ASSEMBLY LANGUAGE SOURCE STATEMENT NUMBER (DECIMAL) (SAME AS TEXT EDITOR LINE NUMBERS)
- LOCATION COUNTER (HEXADECIMAL)
- ASSEMBLED CODE (HEXADECIMAL)
- LABEL FIELD
- OP CODE FIELD
- OPERAND FIELD
- COMMENT FIELD

```
0033 FE00 02E0    ST     LWPI WS        SET WORKSPACE POINTER
     FE02 FFA0
0034 FE04 0200           LI   R0,10     R0 = TENS MULTIPLIER
     FE06 000A
0035 FE08 04C9           CLR  R9        R9 = NO. OF TRIES
0036 FE0A 04CA           CLR  R10       R10 = NO. OF TRIES
0037 FE0C 020C           LI   R12,>80   TMS 9902 CRU ADDR.
     FE0E 0030
0038             * OUTPUT OPENING MESSAGE
```

ONE OR TWO CASSETTE OPERATION

TM 990/302

COMMAND INPUT

SOURCE LISTING, ERROR REPORT

4.2   CONSIDERATIONS

- Optimum system configuration would include two cassette recorder/players. A one-recorder/player operation could require changing source and object tapes as many times as necessary during the second pass; this would not be necessary in a two recorder/player operation. In a one-recorder/player operation, take care to avoid writing over the source code cassette with assembled object. Cassette operation is explained in paragraph 2.7.

- LMF, LDD, and LDS instructions (memory mapping) are not interpreted.

- Assembler directives are restricted to the 11 defined in Appendix F. External symbol references and definitions are not recognized; thus, assembler programs cannot be linked by a linking loader.

- Object will be in non-relocatable absolute code; thus, the user must specify the program load address with the AORG assembler directive (explained in Appendix F).

- Macro instructions are not recognized by the assembler.

(1) FIRST PASS: BUILD TABLE OF SYMBOLS

SYMBOLIC ASSEMBLER

LABEL	LOC. CNTR.
STRT	0000
LOOP	001C
LOP1	002C
LOP2	00A4
SUM1	00FA
SUM2	00FC
SUM3	00FE
SUM 4	0100
•	•
•	•
•	•

SYMBOL TABLE
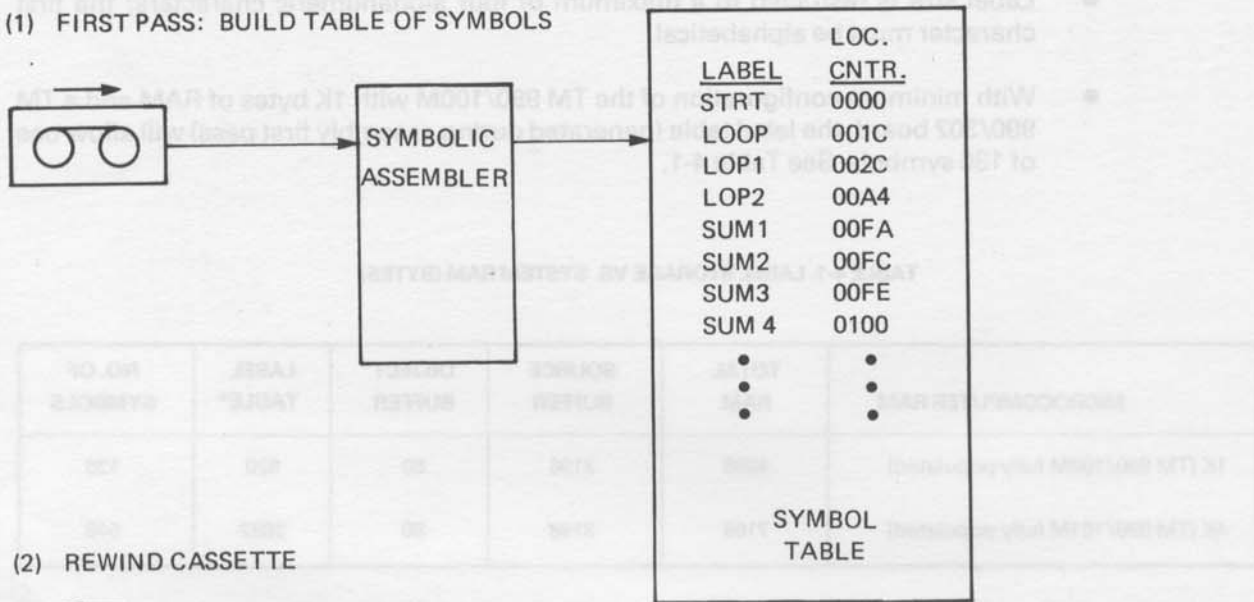
(2) REWIND CASSETTE
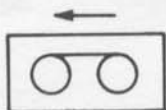
(3) SECOND PASS: RESOLVE ADDRESSES,
GENERATE SOURCE LISTING,
ERROR REPORT

SYMBOLIC ASSEMBLER

SYMBOL TABLE

LOC. CNTR.	OPCODE (HEX)	SOURCE STMT.
0000	02E0	STRT LWPI > 1000
0002	1000	
0004	04E0	CLR @SUM1
0006	00FA	
0008	04E0	CLR @SUM2
000A	00FC	
000C	04E0	CLR @SUM3
000E	00FE	
•	•	•
•	•	•
•	•	•
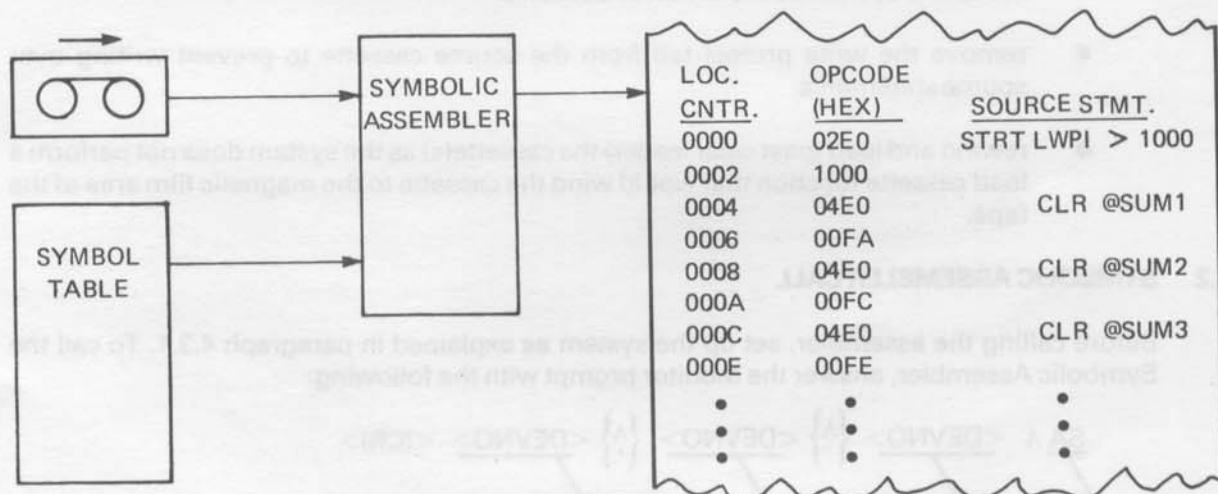
Figure 4-2. Two-Pass Assembler Operation

4-3

- Label size is restricted to a maximum of four alphanumeric characters; the first character must be alphabetical.

- With minimum configuration of the TM 990/100M with 1K bytes of RAM and a TM 990/302 board, the label table (generated during assembly first pass) will allow use of 136 symbols. See Table 4-1.

TABLE 4-1. LABEL STORAGE VS. SYSTEM RAM (BYTES)

MICROCOMPUTER RAM	TOTAL RAM	SOURCE BUFFER	OBJECT BUFFER	LABEL TABLE*	NO. OF SYMBOLS
1K (TM 990/100M fully populated)	4096	3196	80	820	136
4K (TM 990/101M fully populated)	7168	3196	80	3892	648

*Table size in bytes; six bytes required per symbol.

4.3 SYMBOLIC ASSEMBLER CALL AND OPERATION

4.3.1 SYSTEM SETUP

Prior to calling the Symbolic Assembler, perform the following:

- set up the system as explained in Section 2

- remove the write protect tab from the source cassette to prevent writing over source statements

- rewind and load (past clear leader) the cassette(s) as the system does not perform a load cassette function that would wind the cassette to the magnetic film area of the tape.

4.3.2 SYMBOLIC ASSEMBLER CALL

Before calling the assembler, set up the system as explained in paragraph 4.3.1. To call the Symbolic Assembler, answer the monitor prompt with the following:

.SA Λ <DEVNO> {Λ,} <DEVNO> {Λ,} <DEVNO> <(CR)>

- DEVNO of device to receive listing
- DEVNO of device that will receive assembled object
- DEVNO of device containing source statements
- SA command calls assembler

Examples:

(1)   .SA 2,3,1 ◄─────────── Finish with a carriage return

       └─────── Listing to system terminal

      └───────── Object to cassette 2

     └─────────── Source statements from cassette 1

(2)   .SA 2,2,0 ◄─────────── Finish with a carriage return

       └─────── No listing (dummy specified, object only desired)

      └───────── Object to cassette 1 (same as source)

     └─────────── Source statements from cassette 1

(3)   .SA 2,0,1 ◄─────────── Finish with a carriage return

       └─────── Listing on system terminal

      └───────── No object (listing only desired)

     └─────────── Source statements from cassette 1

(4)   Maximum equipment configuration

    .SA 2,3,4 ◄─────────── Finish with a carriage return

       └─────── Listing to second EIA connector (printer or hard-copy device)
                   control will be through the system terminal

      └───────── Object code to cassette 2

     └─────────── Source code from cassette 1

NOTE

Prior to using DEVNO 4 in the assembler its baud
rate should be initialized with the SR command.

4.3.3   ONE- AND TWO-CASSETTE OPERATION

Two-cassette operation is recommended because it uses less steps than a single-cassette
operation. Assembler procedures are slightly different for each configuration. Single or dual
cassette operation is automatically determined by the assembler by comparing source-input
and object-output DEVNO values (same DEVNO indicates single-cassette operation).

NOTE

For either operation, position the tape as specified in paragraph 2.7 prior to beginning assembly.

4.3.3.1 Two-Cassette Operation

Before execution of the assembler call (paragraph 4.3.2), set up the system as explained in Section 2, and have the source cassette rewound to the clear leader. When the assembler call is executed, the assembler will begin reading source statements and continue until the END assembler directive is read or an end-of-file is read (the end-of-file is placed on the tape as one of the final operations of the Text Editor Quit command). After all source code is read in, the code must be read again in a second pass in which the assembler resolves all relocatable symbolic addresses, prints an assembled source code listing (if designated), and writes assembled object to the designated cassette. Before this second pass, the assembler outputs the two-line message:

```
** REWIND TAPE
**HIT 'CR' TO GO —
```

Rewind the source-statement cassette and ready it for the second pass of the assembly. When ready, press the carriage return to begin pass two. The source statements will be read in again. During this pass symbolic addressing will be resolved using addresses from the symbol table generated during the first pass. If the source program is too large to fit into the source statement buffer, repeated reads of the source statement cassette will be made until the END assembler directive or end-of-file is read, with the assembler performing operations on each portion read. As the object buffer is filled up, the object code will be written to the object cassette. When assembly is complete, control reverts back to the monitor and the monitor period prompt will be issued. An example of assembler interaction at the system terminal is as follows (this example designates dummy for source listing in order to show basic interaction between Symbolic Assembler and user without a listing):

```
.SA 2,3,0
    ** REWIND TAPE          ⎫   Set up for second pass
    ** HIT 'CR' TO GO —      ⎭
.  ◄————————————————— End of assembly, period prompt from monitor
```

4.3.3.2 One-Cassette Operation

One-cassette operation requires more steps than a dual-cassette operation; thus, the latter configuration is recommended. Operation is similar to the two-cassette operation as explained in paragraph 4.3.3.1, and it is recommended that the user read that paragraph first. A flow chart of a one cassette operation is shown in Figure 4-3.

Single-cassette operation is sensed by the monitor from the assembler call command (same cassette DEVNO for source-input and object-output indicates single cassette), and the assembler uses interactive message and procedures accordingly. Prior to calling the

START

INSERT SOURCE CASSETTE

ENTER ASSEMBLER

READ IN SOURCE STATEMENTS, 1ST PASS

REWIND SOURCE CASSETTE

READ IN SOURCE STATEMENTS, 2ND PASS

IS OBJECT CODE BUFFER FULL ?

HAS END STATEMENT BEEN PROCESSED ?

HAS END OF FILE BEEN PROCESSED ?

NO — YES — NO — YES — NO — YES

INSERT OBJECT CODE CASSETTE

WRITE OBJECT CODE

INSERT SOURCE CASSETTE

INSERT OBJECT CODE CASSETTE

WRITE OBJECT CODE

END

Figure 4-3. Flow Diagram Of One-Cassette Operation

assembler, the source cassette must be rewound with the tape before the program beginning. When the assembler call is executed, the assembler begins reading in source statements from cassette.

<div align="center">

NOTE

</div>

Considerations for cassette operation in a one-recorder/player configuration are provided in paragraph 2.7. When "rewind" is stated herein, it does not mention the need to unplug the motor control plug. Remember to wind tape past clear leader before recording on it.

During the first pass, the assembler will build the symbol table of symbol characters and their addresses. The first pass is completed after the read of and END directive or an end-of-file code. At this time, the assembler sends a message asking that the source cassette be rewound to the beginning of the program so that the second pass can be initiated.

If the program is small enough, it will be necessary to change cassettes only once (unload source cassette, load object cassette). In this small-program example, the entire program can be contained in the source and object buffers without additional reads or writes (see Table 4-2). This operation will require:

(1) Ready source cassette for first pass (rewind); execute assembler call. First pass begins when carriage return is pressed. Entire source tape is read in. Message prompt indicates end of first pass of source tape:

     **REWIND TAPE
     **HIT 'CR' TO GO —

(2) Set up for second pass of reading source. Rewind the source cassette, and enter a carriage return to start second pass. After the second pass is complete, the following message requests that the source cassette be removed and the object cassette inserted:

     **SWAP TAPES
     **HIT 'CR' TO GO —

(3) Replace source cassette with a rewound object cassette (do not rewind the source cassette as further source lines for this program may be on the cassette). Position object tape past clear leader under printhead. Press the carriage return to start writing of object to cassette.

(4) When the monitor prints a period (.) on the terminal, program assembly is complete and the user can call up another SDB program. Assembled object is on the cassette in the unit.

If the source program is so large that the object buffer in memory fills up, successive readings of source and writings of object are required in the second pass:

(1) Ready source cassette for first pass (rewind); execute assembler call. First pass begins when carriage return is pressed. Entire source tape is read in. Message prompt indicates end of first pass of source tape:

      **REWIND TAPE
      **HIT 'CR' TO GO —

(2) Rewind source tape for second pass of reading source. Enter a carriage return when ready. Source is read in and object sent to object buffer. When the object buffer is full, the following message requests a change from the source cassette to the object casette:

      **SWAP TAPES
      **HIT 'CR' TO GO —

(3) Replace source cassette with object cassette (do not rewind either but have tape past clear leader if initial input). Depress the carriage return; object will be written to the object cassette. When complete, the next prompt asks to reinsert the source cassette:

      **SWAP TAPES
      **HIT 'CR' TO GO —

(4) Replace the object cassette with the source cassette (do not rewind either). Depress the carriage return; source will be read again from the source cassette. When the assembler finishes with this segment of source, the message prompt appears:

      **SWAP TAPES
      **HIT 'CR' TO GO —

Repeat steps (3) and (4) until assembly of the program is complete. Completion will be indicated by a period (.) prompt at the terminal meaning that any SDB command can be entered. Complete program object will be in cassette in unit.

NOTE

1. Because the assembler does not designate which cassette to load next, it is recommended to mark each cassette as "SOURCE" or "OBJECT" to identify the tape contents.

2. When cassettes are removed unwound, take care to prevent movement of the tape within the cassette while removed. Such movement could result in loss of data.

An example of assembler/user interaction is as follows:

```
.SA 2.2.0  ◄────────────────────── Assembler call (DEVNO 2 for source and object)
** REWIND TAPE        ⎫   Set up source for second pass
**HIT 'CR' TO GO —    ⎭        depress carriage return
** SWAP TAPES         ⎫   Insert object cassette,
**HIT 'CR' TO GO —    ⎭        depress carriage return
        •
        •           ·  Successive source/object cassette changes
        •
** SWAP TAPES         ⎫   Insert source cassette,
**HIT 'CR' TO GO —    ⎭        depress carriage return
** SWAP TAPES         ⎫   Insert object cassette,
**HIT 'CR' TO GO —    ⎭        depress carriage return
                      ──── End of assembly, control returns to monitor
```

4.4   ASSEMBLY LISTING FORMAT

If designated in the assembler call, a source statement listing will be printed during the second pass of the assembler. The listing format is shown in Figure 4-3. Paragraphs 4.4.1 to 4.4.7 explain the different columns and fields of the listing.

```
                    ASSEMBLY LANGUAGE SOURCE STATEMENT NUMBER (DECIMAL)
                          (SAME AS TEXT EDITOR LINE NUMBER)

                    LOCATION COUNTER (HEXADECIMAL)

                    ASSEMBLED OBJECT CODE (HEXADECIMAL)

                    LABEL FIELD

                    INSTRUCTION MNEMONIC

                    OPERAND FIELD

                                                COMMENT FIELD

    0033 FE00 02E0    STAR   LWPI  WSP          SET WORKSPACE POINTER
         FE02 FFA0
    0034 FE04 0200           LI    R0,10        R0 = TENS MULTIPLIER
         FE06 000A
    0035 FE08 04C9           CLR   R9           R9 = NO. OF TRIES
    0036 FE0A 04CA           CLR R10            R10 = NO. OF TRIES
    0037 FE0C 020C           LI    R12,>80      TMS 9902 CRU ADDR.
         FE0E 0080
    0038                   * OUTPUT OPENING MESSAGE
    0039 FE10 2FA0           XOP   @MESS,14     OPENING MESSAGE
         FE12 FEB0
```

Figure 4-4. Assembly Listing Format

4.4.1 ASSEMBLY LANGUAGE SOURCE STATEMENT NUMBER

This is the decimal line number of the source statement. This line number is the same specified when using the Text Editor. This number shows the sequence in which the statements were processed by the assembler.

4.4.2 LOCATION COUNTER

This is the hexadecimal number showing the memory address of the assembled code. Location counter values are given only to locations containing executable code or data values. This value is also used by the assembler to generate the symbol table locations.

The location counter value is absolute from the beginning of the program; thus, it normally begins with location 0000_{16} unless an AORG (absolute origin) assembler directive designates an alternate address. If used, the AORG value will bias the location counter to the specified memory address. The object code generated following an AORG directive is not relocatable, and will force load to the specified address. The location counter also indicates the memory address at which the object resides when loaded into memory without a loader bias. In Figure 4-4, the object code is $02E0_{16}$ at memory address (M.A.) $FE00_{16}$, $FFA0_{16}$ at M.A. $FE02_{16}$, etc. The location counter increments by 0002_{16}, indicating the number of bytes from the beginning of the program.

4.4.3 ASSEMBLED OBJECT CODE

The hexadecimal object code, resulting from the assembly process, is placed in the third column. The corresponding source statement is shown in the next two columns to the right.

4.4.4 LABEL FIELD

This field contains an alphanumeric (A to Z, 0 to 9) symbol which may be used to reference the instruction or data on the same line. The symbol will be given the value in the location counter. Symbols can contain a maximum of four characters, the first of which must begin with an alphabetic character (A to Z), and optional other characters must be alphanumeric. Labels must begin at the first column of the source statement. This field is separated from the mnemonic field by at least one space. A comment can begin in the first column of the label field as explained in paragraph 4.4.7.

4.4.5 INSTRUCTION MNEMONIC FIELD

This four-character field contains assembly-language and assembler-directive instruction mnemonics. It is separated from adjacent fields by at least one space.

4.4.6 OPERAND FIELD

This field contains the operands of the instructions and directives. It is separated from adjacent fields by at least one space and contains no spaces.

4.4.7 COMMENT FIELD

This field contains comments that explain source-statement operation or its role in the program. A comment can take up a full line by beginning the line with an asterisk (*) in the first character position of the label field.

4.5 INSTRUCTION SET

The 72 instructions used by the microcomputers are assembled by the Symbolic Assembler. This includes the DCA, DCS, and LIIM instructions used by the TMS 9940. Instructions are listed in Appendix H. The Symbolic Assembler also recognizes two pseudo instructions:

- The NOP instruction that can be used in place of a JMP $+2 instruction which essentially is a no-op (no operation). NOP can be used to replace code to be deleted in memory or it can be used to force additional execution time. Both NOP and JMP $+2 assemble to the machine code 1000_{16}. The pseudo instruction uses no operand.

- The RT instruction can be used in place of a B *R11 instruction (the normal return from a branch and link subroutine). Both RT and B *R11 assemble to the machine code $045B_{16}$. The pseudo instruction uses no operand.

4.6 LABELS

Labels are explained in paragraph 4.4.4.

4.7 MATHEMATICAL EXPRESSIONS

Mathematical expressions can be used in a limited basis. They must follow several rules:

- can be used only with valid symbols

- restricted to use with instructions of formats 1, 3, 4, 6, and 9; note that these formats contain T fields that designate the type of addressing on the source or destination operand (formats are explained in Appendix H, Instruction Set); for example: MOV @TABL+2,@SUM-16

- *cannot* be used with immediate operands (*no* LI R2,VALU+18, or LWPI WP+32)

- total value of the expression cannot exceed 65,535 ($FFFF_{16}$)

- are limited to only addition and subtraction expressions; multiplication and division will not be interpreted.

- may not be parenthesized.

- cannot be used with the EQU assembler directive.

4.8 OBJECT CODE

Object format is generated in 16-bit hexadecimal segments, each segment preceded by a loader tag. Loader tags and object format are explained in Appendix G.

All object records will contain a checksum field. This checksum is the two's complement of the hexadecimal value of the ASCII code of all generated object codes and the loader tags in one object record. The Relocating Loader will use this value to check the validity of each object record loaded.

Object loader tags utilized by the Symbolic Assembler are 0, 1, 7, 9, B, and F.

4.9 ERRORS

As the assembly listing is being printed during the second assembler pass, errors found during assembly will be noted on the assembly listing by an error number message following the line in question. The error number will identify the error according to the codes in Table 4-2.

For object code corresponding to an invalid source line, the assembler will replace that instruction with "NOP" (no operation) assembler pseudo-instructions which have an object value equal to 1000_{16}. The NOP instruction merely tells the processor to execute the next instruction; this allows the user to run his program during debugging, substituting other object code for the NOP's. One NOP will be substituted for each word in the invalid instruction.

An error total will be printed at the end of the listing. Error numbers are explained in Table 4-2 and Appendix E.

TABLE 4-2. ASSEMBLER ERROR CODES

ERROR NO.	EXPLANATION
1	Invalid Symbol: Symbols must be alphanumerical with the first character alphabetical. Use correct format.
2	Multiply Defined Symbol: Symbol is used to define location of a previous source line. Use other symbol.
3	Symbol Table Overflow: Symbol table cannot accept any further entries. Restrict quantity of symbols to amount shown in Table 4-1.
4	Mnemonic Size Too Large: Op-code mnemonic is more than four characters.
5	Undefined Mnemonic: Op-code mnemonic is not one of the valid TM 990 mnemonics nor a defined XOP mnemonic. Valid TM 990 mnemonics are listed in Appendix H.
6	Illigal Register Number: Register number is greater than 15.
7	CRU Instruction Displacement: Displacement of TB, SBZ, or SBO instructions is outside the range of −128 to +127 words.
8	Jump Instruction Displacement: Displacement of jump instructions is outside the range of −128 to +127 words.
9	Invlaid Shift count: Shift count must be from zero to 15.
10	Non-increasing Location Counter: Memory address in AORG assembler directive is a smaller value than current value of location counter. This will occur when second AORG value is less than the value in a prior AORG directive. Location Contents must be in ascending order.
11	Byte Value Too Large: Operand of BYTE assembler directive is a value larger thana 255.
12	No start of Text: Character string following a TEXT assembler directive did not start with a single quote. Such character strings are delimited with single quotes. A single quote within a TEXT directive is specified by two single quotes.
13	IDT Length Error: Character string of IDT assembler directive (at beginning of program) has character string of more than eight characters.
14	Invlaid IDT: Character string of IDT assembler directive did not begin with a single quote.
15	Illegal TEXT Statement: TEXT statement contained character that cannot be interpreted.
16	Illegal XOP Number: XOP number above 15 specified. XOP numbers are from 0 to 15.
17	Undefined Symbol: Symbol was used in an instruction which had not been defined in the symbol field of the source statement.
18	Input I/O Error: Error in reading source from cassette; probably a checksum error or improper cassette connection.
19	No END Directive: Must have an END directive as last statement in program (see paragraph F.2.7 in Appendix F). Use Text Editor to input this statement.
20	Illegal mathematical expression used. Rewrite without mathematical expression. See paragraph 4.7.

As the assembly listing is being printed during the second assembler pass, errors found during assembly will be noted on this assembly listing by an error number message following the line in question. The error number will identify the error according to the codes in Table 4-2.

4.10 ASSEMBLY OF EXAMPLE PROGRAM

Figure 4-5 shows the listing generated by the assembly of the example program written in the Text Editor section of this manual. In the assembler call, the DEVNO's for both source and object were the system terminal; thus, the object records are also printed out.

```
0100                      IDT   'BLINK'
0101    1040              AORG  >1040
0110    1040 02E0 BEGN    LWPI  >1000        WP ADDR
        1042 1000
0120    1044 020C         LI    R12,>1706    LED CRU ADDR
        1046 1706
0121    1048 C041 STRT    MOV   R1,R1        R1 ALL ZEROS?
0122    104A 1603         JNE   OFF          NO, TURN OFF LED
0124    104C 1D00 ON      SBO   0            TURN ON LED
0125    104E 0701         SETO  R1           SET ON/OFF FLAG
0126    1050 1002         JMP   RUN          GO TO TIMER
0127    1052 1E00 OFF     SBZ   0            TURN OFF LED
0128    1054 04C1         CLR   R1           RESET FLAG
0130    1056 0202 RUN     LI    R2,62500     SET UP DELAY COUNTER
        1058 F424
0134    105A 0602 LOOP    DEC   R2           IS R2 ZERO ?
0136    105C 16FE         JNE   LOOP         NO, DECREMENT AGAIN
0138    105E 10F4         JMP   STRT         RESTART PROGRAM
0140         1040         END   BEGN
ERRORS=0
```

——— Object Records

```
00000BLINK    91040B02E0B1000B020CB1706BC041B1603B1D007F48BF
9104EB0701B1002B1E00B04C1B0202BF424B0602B16FEB10F4110407F3FCF
:104EB0701B1002B1F00B04C1B0202BF424B0602B16FEB10F4110407F3FCF
```

——— Colon indicates end-of-program; loader ignores any following object

Figure 4-5. Assembly Listing Of Example Program

5.1   GENERAL

This section explains the operation of the Relocating Loader program which can be used to load object code into memory for execution or for programming onto PROM's. Features of the Relocating Loader program allow the user to designate the following:

- Load Address: The address where the program is to be loaded into memory. This address is used to determine the program start address when using the Debugger or where the code is located when using the PROM Programmer.

- Load Bias: A bias to be added to the values of relocatable addresses in the program being loaded. Because a program can be assembled as if it begins at relative location 0000_{16} and is to be executed beginning at a different address, then the relocation must be resolved before execution. For example, if a program is assembled as if it began at memory address 0000_{16} but is loaded into memory for execution beginning at address $E000_{16}$, it should be biased by the value $E000_{16}$, and a symbolic address with an assembled address value of 0080_{16} will have an address value of $E080_{16}$ after being loaded. If a segment of a program is being loaded the bias will be to the beginning of program, not beginning of the segment.

NOTE
The TM 990/302 Symbolic Assembler does not output relocatable object; its object is in absolute form.

- Load Length: Length in bytes of object code to be input from the object cassette; this allows loading only part of a program.

- Start Address: For an absolute object file, it is the absolute address of the first byte to be loaded into memory. For a relocatable object file, it is equal to the value of the second parameter (load bias) plus the relocatable address of the first byte to be loaded into memory. This parameter is ineffective with the Relocating Loader if an object file on cassette was generated from a source file without an AORG directive.

5.2   SYSTEM CONFIGURATION

A typical configuration is shown in Figure 5-1.

Communication to the system will be through the system console. The cassette recorder/player will contain the object tape, wound to the clear leader.

5.3   CONSIDERATIONS

Memory addresses 0000_{16} to $03FF_{16}$ are reserved for use by the operating system. Do not attempt to load programs in this memory area.

Remove any EPROM connected to the TM 990/302 board or remove EPROM programming power before reading data from cassette; otherwise, the EPROM could be mistakenly programmed.

Be aware of the address at which the program will begin when (1) in the target system and (2) in debugging using the Program Debugger (Section 6). The Relocating Loader will resolve any address conflicts in relocatable code for either of these functions. With the assistance of the Load Length and Start Address, a large program can be divided into segments and loaded into memory. Usually a Load Length value greater than the length of the program will not affect the normal loading of the program because the loading will stop at the end of the object program. However, the sum of the Load Length and the Start Address should always be less than or equal to FFFF$_{16}$.

5.4 LOADER PROGRAM CALL AND OPERATION

The call to the Relocating Loader uses two lines. The first line requests the Relocating Loader and specifies the DEVNO of the recorder/player containing the object cassette. The second line issued to describe the four load criteria: Load Address, Load Bias, Load Length, and Start Address (further defined in paragraph 5.1 above). All four criteria are assumed to be hexadecimal values.

The first line is a response to the period (.) prompt by the monitor. The second line is a response to a question mark (?) prompt by the Relocating Loader. Both lines are completed with a carriage return.

NOTE

If the second line parameters cannot be interpreted by the Relocating Loader, the question mark prompt will be reissued.
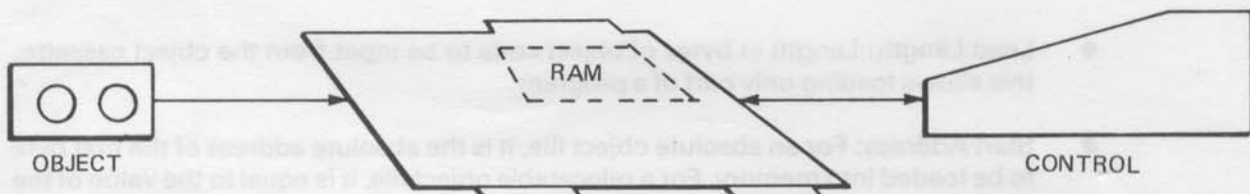
Figure 5-1. Relocating Loader Block Diagram

The format is as follows (see paragraph 1.7):

    .RL<DEVNO> <(CR)>
    ?<Load Address><$\wedge$><Load Bias< $\wedge$ > <Load Length><$\wedge$><Start Address> <(CR)>

5.5   EXAMPLES

(1)   This first example can be used to load into memory the example program that was developed in the sections on the Text Editor and Assembler. Load a 20_{16} byte program at memory address 1040_{16}. It does not need to be relocated because it was assembled as if it began at memory address 1040_{16} by using the AORG > 1040 directive (also, the Symbolic Assembler outputs only absolute object). This will set up the program for being run on the Program Debugger with the Program Counter set to 1040_{16}. Begin loading at the start of the program.

```
                                  ┌──────────── Input object program from cassette 1
                          ┌───────┘
                    .RL 2
                    ?1040,0,20,1040
                                             Start to load at the beginning of object program
                                         Load 20₁₆ bytes maximum
                                      No bias needed, AORG >1040 used
                                   Load code into memory starting at memory address 1040₁₆
```

.RL 2
?1040,0,20,1040

— Start to load at the beginning of object program

— Load 20_{16} bytes maximum

— No bias needed, AORG >1040 used

— Load code into memory starting at memory address 1040_{16}

NOTE

The relocating Loader sets the Program Counter to the entry address specified in the END assembler directive of the program. Because the example program uses this feature, the program can be executed after being loaded by using the EX (execute program) command of the Program Debugger (paragraph 6.5.1)

(2)   Load entire 1000_{16} byte program into memory starting at memory address 40_{16}, bias to starting at memory address $E000_{16}$ (code is relocatable). The maximum load length is 1000_{16} bytes, and start the loading at the beginning of the object program. (The bias of $E000_{16}$ allows this program to be programmed onto an EPROM which will be inserted into the system and executed with a beginning program address of $E000_{16}$).

— Input program from cassette 2

.RL 3
?400,E000,1000,E000

— Start to load at the beginning of object program

— Load 1000_{16} bytes maximum

— Bias to memory address $E000_{16}$ (code is relocatable)

— Load code into memory starting at memory address 400_{16}
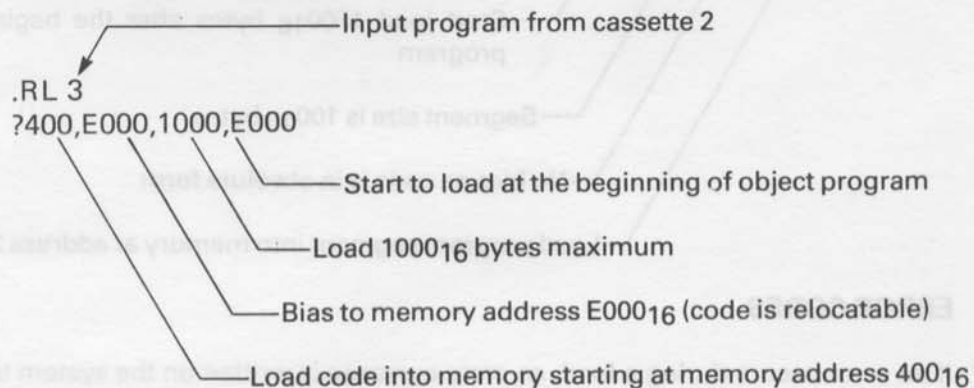
(3) Load entire 2048-byte program starting at memory address 1000_{16} with no bias (code is written in absolute form with an AORG >1000 directive specifying the beginning load address). Note that spaces are used for field delimiters as well as the comma. Object program is on cassette 1.

.RL 2
?1000,0,800,1000

—————— Start load at the beginning of object program

—————— Maximum length is 2048 bytes (800_{16})

—————— No bias (code is in absolute form)

—————— Load code into memory at address 100016

(4) Load the second 1000_{16} byte segment of a 2000_{16} byte relocatable program into memory starting at memory address 2000_{16}. Bias as if the entire program began at memory address $D000_{16}$. Note that the bias for the beginning of the program is given (not the beginning of the program segment). Object program is on cassette 2. There is no provision for placing relocatable code on audio cassette.

.RL 3
?2000,D000,1000,E000

—————— Start load 1000_{16} bytes after the beginning of object program

—————— Segment size is 1000_{16} bytes

—————— Bias for program start address is $D000_{16}$ (code is relocatable)

—————— Load program segment into memory at address 2000_{16}

(5) Load the last 100_{16} bytes of a 2000_{16} byte program into memory starting at memory address 2000_{16}. No bias is wanted as code is in absolute form. Object program is on cassette 1. Program is AORGed to 0.

.RL 2
?2000,0,100,1F00

—————— Start load $1F00_{16}$ bytes after the beginning of object program

—————— Segment size is 100_{16} bytes

—————— No bias as code is in absolute form

—————— Load program segment into memory at address 2000_{16}

5.6   ERROR CODES

If an error occurs during a load, an error message is written on the system terminal and the load is aborted. Error messages contain a code which is defined as follows.

5.6.1 **ERROR 51

- Meaning: An invalid load tag was found. The loader interprets several load tags including 9 (absolute load address), A (relocatable load address), B (absolute data), and C (relocatable data). Load tags used by the 990 family of computers are explained in Appendix G.

- Recovery: Rewind cassette and re-execute the load operation. If the error reoccurs, reassemble the program to obtain a new object on cassette.

5.6.2 **ERROR 52

- Meaning: Checksum error occurred. When the object code is formatted into object records by the assembler, the last load tag field of each record is a checksum value which is the two's complement of the sum of all ASCII character values representing the object code; this includes all characters beginning with the first tag character in the object record up to and including the "7" tag of the checksum field. The loader makes a similar computation when loading the object and compares the results to the checksum value. If the comparison is a match, the loaded data is considered valid.

- Recovery: Rewind cassette and re-execute the load operation. If the error reoccurs, reassemble the program to obtain a new object on cassette.

NOTE

For both errors 51 and 52, clean the tape heads of the recorders (if possible) before re-executing the load operation.

SECTION 6
PROGRAM DEBUGGER

6.1 GENERAL

The TM 990/302 Program Debugger allows the user to monitor the execution of his program, check for problems and (if necessary) patch code in question, then resume program execution using the new values. The Debugger features include: (1) execution of one or more instructions starting at a designated Program Counter value, (2) single-step execution, (3) execute program to a specified instruction location (breakpoint), (4) trace program execution with hardware register printout after each instruction executed, (5) record conditional-jump caused path changes, and (6) inspect and change contents of memory, the hardware registers, the workspace registers, or Communication Register Unit (CRU) values. The Relocating Loader program (Section 5) is needed in order to load into memory the object to be debugged.

Commands under the Debugger include (applicable paragraph number is in parentheses):

- EX Command: Execute program unconditionally (6.5.1)

- IC Command: Inspect/change CRU values (6.5.2)

- IM Command: Inspect/change memory contents (6.5.3)

- IR Command: Inspect/change hardware register values (6.5.4)

- IW Command: Inspect workspace register (6.5.5)

- RU Command: Conditional jump run, record path changes (6.5.6)

- SB Command: Set breakpoint for execution limit (6.5.7)

- ST Command: Software trace where the three hardware register contents are printed out during each instruction execution (6.5.8).

6.2 SYSTEM CONFIGURATION

Minimum system configuration would include one cassette and a terminal as peripherals. Object would be loaded into memory with the Relocating Loader, and the debugged object would be written back to the same cassette using the Dump Memory program (Section 8).

- connect the cassette and terminal as explained in Section 2.

- load an object program at the desired beginning address using the Relocating Loader as explained in Section 5.

- insert the tape which will receive the debugged object in the recorder/player, and rewind it to clear leader. Paragraph 2.7 describes correct tape handling procedures.

6.3 CONSIDERATIONS

All numerical inputs to commands must be hexadecimal values without the preceding greater-than (>) sign. For example, to inspect fifteen bits at CRU address 100_{16}, the value F (*not* 15) must be specified:

    ? IC 100,F

Program execution begins at the address values in the hardware registers. These registers are first set by the IR (inspect registers) command described in paragraph 6.5.4.

6.4 PROGRAM DEBUGGER CALL

Answer the monitor period prompt with the following:

    .DP<(CR)>   Finish with a carriage return

The Program Debugger will respond with a question mark (?) prompt asking for one of the Debugger commands. For example:

    .DP ◄─────── Call Debugger
    ? ◄─────────────── Debugger prompt asking for command

6.5 DEBUG PROGRAM COMMANDS

6.5.1 EXECUTE MEMORY COMMAND (EX)

The EX command causes program execution to begin at the Workspace Pointer (WP) and Program Counter (PC) address set using the IR (inspect hardware registers, paragraph 6.5.4) command or at the PC value set by the Relocating Loader when a program entry address is specifed in the END assembler directive of the program.

Format:

    ? EX <(CR)>

NOTE

> The example program developed in the section on the Text Editor and Symbolic Assembler, and loaded by the Relocating Loader can be executed with the EX command if properly assembled and loaded. When executed, the LED on the TM 990/302 board (next to edge connector P2) should blink.

6.5.2 INSPECT/CHANGE CRU COMMAND (IC)

This command causes a pattern of 1 to 16 bits to be displayed at a CRU software base address and allows the user to write a 1-to-16 bit pattern to the CRU software base address displayed. The address entered is the unshifted register 12 contents, not the CRU bit address (i.e., the CRU bit address is the register 12 16-bit address shifted one bit right). For a further explanation of the CRU and the instructions that read and write to it, see the programming section of your microcomputer user's guide.

The command displays a 16-bit value. After displaying the value read at the CRU, the printhead spaces to the right and awaits one of the following:

- a space causing the same CRU address to be read again and the value displayed

- a carriage return causing a return to the Debug Program scanner

- a new hexadecimal number to be written to the address displayed followed by a space or carriage return causing the action specified in the above two descriptions for space and carriage return.

Format:

?IC[Λ]<CRU Address> $\left\{ {\Lambda \atop ;} \right\}$ <[No. of Bits]> <(CR)>

If 0 used or no entry, 16 bits displayed

Examples:

?IC 120	Show value at CRU bits 0090_{16} to $009F_{16}$ (TMS 9901)
0120=FFFF	Carriage return, return to command scanner.
?IC 120,8	Show value at CRU bits 0090_{16} to 0097_{16}
0120=00FF 0	Enter a value of zero, terminated with a space
0120=0000	Carriage return, return to command scanner
?IC 120	Show value at CRU bit 0090_{16} to $009F_{16}$
0120=FF00	Carriage return, return to command scanner
?	

If the number of bits specified is 0, all 16 bits will be read. The valid bit numbers are $0-F_{16}$.

The CRU value displayed will show the CRU bit values beginning at the address specified. If one bit address is requested, its value (a one or zero) will be shown in the rightmost bit (least-significant bit or LSB) of the displayed value and the remainder of the displayed value is zero filled. If 2 to 16 address bits are to be displayed, succeeding addresss bits will be shown right to left as depicted below:

?IC 80,6
0080=003A
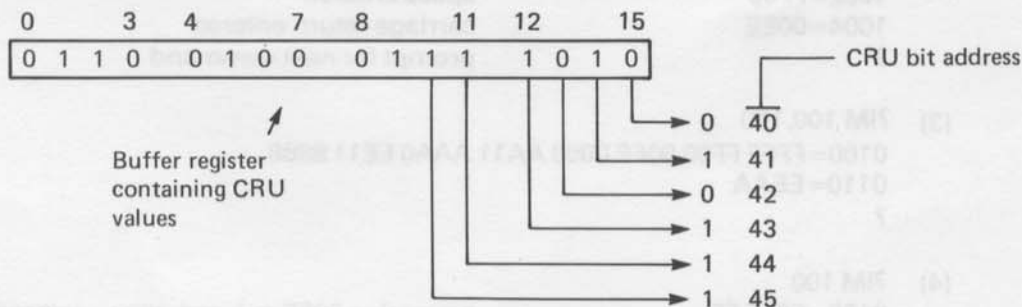
6.5.3 INSPECT/CHANGE MEMORY COMMAND (IM)

This command has two modes. In the first mode, the contents of one 16-bit memory address is displayed, and the user can select further action by the following:

- A space-bar input causes the next word to be displayed.

- A minus-key input causes the previous word to be displayed.

- A carriage-return input causes a return to the Program Debugger command scanner.

In the second mode, a block of such 16-bit memory contents are displayed, and the user can select further action by the following:

- A space-bar input causes a pause during the display process. Another space bar entry will continue it.

- An ESC-key input causes the early termination of the display process.

Formats:

?IM[Λ]<Memory Address> <(CR)>

causes contents of one memory address to be displayed.

?IM[Λ] <Beginning Address> {Λ,} [Ending Address>] <(CR)>

causes contents from "beginning address" to "ending address" to be displayed

Examples:

(1) ?IM 1000 ──────────── finish with carriage return
     1000=FFFF ◄──────────── carriage return entered
     ? ◄──────────────────── prompt for next command

(2) ?IM 1000
     1000=FFFF   space entered
     1002=FF00   space entered
     1004=00EE   carriage return entered
     ?   prompt for next command

(3) ?IM 100,110
     0100=FFFF FF00 00EE 0000 AA11 AAA0 EE11 8888
     0110=EEAA
     ?

(4) ?IM 100
     0100=FFFF EE   new value 00EE entered, space entered
     0102=FF00 0   new value 0000 entered, space entered
     0104=00EE —   minus entered
     0102=0000   prints changed contents of M.A. 0102
     ?   CR returns to command scanner

6.5.4 INSPECT/CHANGE HARDWARE REGISTERS (IR)

This command is used to set up the hardware registers before executing the execute (EX) command, the software trace (ST) command, the conditional jump run command (RU), or the set breakpoint (SB) command. To change the displayed register, enter a new value next to the dsplayed value. To repeat the displayed register's value, enter a minus (−) sign.

Format:

```
?IR   <(CR)>
```

Examples:

(1) ?IR   finish with a carriage return
     W=0700   Workspace Register displayed, space entered
     P=0122   Program Counter displayed, space entered
     S=1000   Status Register displayed, space or CR entered
     ?   prompt for next command

(2) ?IR
     W=0700 6A0—   new value, minus entered
     W=06A0   new register value displayed, CR entered
     ?   return to command scanner

(3) ?IR
     W=06A0   new value shown as in above command, CR
     ?   entered

6.5.5 INSPECT/CHANGE WORKSPACE (SOFTWARE) REGISTERS COMMAND (IW)

This command allows the user to inspect and change one or all 16 of the Workspace Registers. These software registers are located in memory beginning at the address specified by the current Workspace Pointer value or as defined using the inspect/change hardware registers debug command (IR). Contents will be displayed as contents of memory addresses instead of a specific register. Specify either one register number after the IW command or specify only the IW command to inspect all register contents. The command is completed with a carriage return. Options are available when a register is displayed.

- space bar entry will display contents of next register

- minus key (−) will display contents of previous register

- hexadecimal value entered will change contents of displayed register to the entered value

Formats (see paragraph 1.7):

```
? IW[Λ] [Register No.] <(CR)>
```

Examples:

(1) Inspect one register with WP = 2000$_{16}$

    ?IW6   finish with carriage return
    200C=2314   finish with carriage return

(2) Inspect and change register values

    ?IW6
    200C=2314   R6 displayed, press space bar
    200E=9887 BBBB   R7 displayed, enter new R7 value, press space bar
    2010=0EEE CCCC   R8 displayed, enter new R8 value, press space bar
    2012=5555 —   R9 displayed, enter minus sign to display R8
    2010=CCCC —   R8 displayed showing change
    200E=BBBB   R7 contents, finish with carriage return
    ?   return to command scanner

(3) Print out entire Workspace Register values with WP at M.A. 2000$_{16}$

    ?IW ◄——————— finish with carriage return, print all registers (default)
    2000=0113 EEDD F435   0000   AAAA 2314   9887   0EEE
    2010=5555 4444 3476   5FFA   1111   2222   2333   FFF1

6.5.6 RUN PROGRAM, MONITOR CONDITIONAL JUMPS (RU)

This command causes controlled program execution beginning at the value in the Program Counter, which can be set using the IR (inspect hardware register) command. The run command operand is the number (in hexadecimal) of instructions to be executed, with no default value; a number must be specified. This conditional jump run command provides the ability ·to record the data path changes caused by conditional jump instructions (excludes unconditional jump or JMP instruction). During the program execution each instruction is checked to see if it is a conditional jump instruction and also to see if that conditional jump instruction causes a path change (jump condition supported by Status Register contents). If so, the address of that jump instruction will be recorded on a 16 word buffer. After the specified number of execution steps is completed, the contents of the current hardware registers (WP, PC, and ST) and the buffer is printed out. The buffer contents are as follows: the first contains the most recently occurred conditional jump instruction, the second word contains the second most recent one, and so on. The maximum number of path changes that can be recorded is 16. If more than 16 path changes have occurred, the most recent 16 changes are recorded. If there are less than 16 path changes, the unused buffer spaces are filled with zeros.

Format (see paragraph 1.7):

    ? RU[Λ] <N><(CR)>
              \
               \————————— Number (in hex) of steps of execution.

After executing the specified number of instructions, control will return to the Debug Package Command Scanner, which issues its question mark prompt.

Examples:

```
?IR                                    finish with a carriage return
W=0710                                 space entered
P=0604 510 ◄────────────────────────── set Program Counter to the execution-start
                                       address, terminated with carriage return


? RU 46                                Run for 46₁₆ instructions
     0710 0554 1000 ◄───────────────── WP, PC, & ST contents at run completion
  0290=054E 0532 0524  0000  0000  0000  0000  0000  ⎫ 3 conditional
  02A0=0000 0000 0000  0000  0000  0000  0000  0000  ⎭ jumps executed
  ?
```

The above example showed that 46_{16} instructions have been executed and three path changes recorded. The most recent path change is in memory address $054E_{16}$.

6.5.7   SET BREAKPOINT COMMAND (SB)

This command allows the user to designate one or two memory addresses at which program execution will halt and the contents of the hardware registers will be printed out. When a breakpoint is executed, all breakpoints are cleared, and new breakpoints must be re-entered, if desired.

The breakpoint command substitutes the machine code for XOP 15 at the address of the breakpoint(s). When the breakpoint is executed, the hardware registers contents are printed out (WP, PC, and ST register contents), the original contents at the breakpoint address are restored, and command returns to the Debugger Package command scanner. Execution of this command follows this sequence:

- Enter the IR command to set up the hardware registers to the beginning of program execution.

- Enter the SB command specifying the hexadecimal memory addresses where the breakpoints are desired. When the command is terminated with a carriage return, program execution starts.

- When any breakpoint occurs, the hardware-register contents will be printed out, control will return to the command scanner with the question-mark prompt issued, and all breakpoints will be cleared.

Format:

?SB[Λ] <Breakpoint Address> {Λ, } [Breakpoint Address] <(CR)>

                                                       hexadecimal values

Examples:

```
(1)  ?IR                              finish with carriage return
     W=0710 ◄────────────────────────── set up workspace pointer and the start
                                        address of the program to be executed

     P=0342 05A0
     ?SB 05F4 ◄──────────────────────── set address of last instruction to be executed
         0710 05F4 1000 ◄─────────────── breakpoint printout of register contents
     ? ◄─────────────────────────────── return to command scanner, breakpoints cleared
```

(2) ?SB 610   finish with carriage return
         0710 0610 1000 ◄────── breakpoint printout of register contents
     ? ◄──────────────────────── return to command scanner, breakpoint cleared

6.5.8 SOFTWARE TRACE COMMAND (ST)

This command causes the contents of the Workspace Pointer, Program Counter, and Status Register to be printed out after execution of each instruction. Controlled program execution begins at the value in the program counter, which can be set using the IR command. The ST command operand is the number (in hexadecimal) of instructions to be executed, with a default value of one instruction if no operand is specified. The trace function is provided by printing out the hardware register contents after each instruction execution. To use this command:

● Initialize the hardware registers to the start of program execution.

● Enter the Software Trace Command.

At each instruction execution, the WP, PC, and ST contents will be printed.

Formats (see paragraph 1.7):

    ? ST [Λ] [N] <(CR)>
                  └─Number of steps of execution (hexadecimal, default = 1)

Example:

Execute 4 instructions under software trace.

    ?IR   Finish with a carriage return
  / W=0710   Space entered
    P=0322 5EE   Change program counter value, CR exit
    ?ST 3 ◄─────────────────────────────── Trace 3 instructions
       0710   05EE   1000
       0710   05F0   4000
       0710   05F2   2000
    ? ◄──────────────────────────────────── Return to command scanner

CAUTIONS

1. Execute the software trace on user programs only. Do not execute this program on one of the SDB utilities.
2. This command cannot be stopped before completion of all instructions executed (cannot be stopped by ESC key).

6.5.9 EXIT USING ESC KEY

To exit a debugger command, press the ESC key and control reverts to the debugger command scanner. A second ESC key entry hands over control to the monitor.

SECTION 7

EPROM PROGRAMMER

7.1 GENERAL

This program allows the user to program the following EPROMs:

- TMS 2708 EPROM

- TMS 2716 EPROM

- TMS 2508 EPROM

- TMS 2516 EPROM

- TMS 2532 EPROM

Prior to programming these EPROMs, the user object program must be loaded into memory using the Relocating Loader (Section 5). This program will have gone through the software development cycle of generation by text editing, assembly into object, load object into memory and program debug, then repetition of this cycle (edit-assemble-load-debug) until final approval of the object program. The second consideration in paragraph 7.3 explains a restriction on reading in data to be programmed.

Two hardware selections must be made before the erased EPROM can be inserted and the EPROM programming software called up. These include:

- selection of the correct personality card for the EPROM type

- insertion of jumpers on the personality card corresponding to the specific EPROM to be programmed.

It is assumed that the system memory will have been correctly configured according to the memory mapping switch on the TM 990/302 board (see paragraph 2.3.3.1).

When the system is configured properly, the EPROM Programmer can be called up. It has four basic features available to the user:

- program the EPROM with the contents of the designated memory addresses in either inline mode (consecutive byte addresses) or parallel mode (all even bytes or all odd bytes). The values programmed into EPROM are automatically compared to the designated memory address upon completion.

- compare EPROM contents to memory contents; this is especially useful in checking a just-programmed EPROM for correct data transfer and is an automatic feature of the EPROM programmer

- read EPROM contents into memory (no copy verification is made)

- verify that the EPROM is erased (all ones)

7.2 SYSTEM CONFIGURATION

System configuration should consist of the following:

- TM 990/302 and microcomputer boards with program to be placed into the EPROM already loaded into RAM using the Relocating Loader (Section 5)

- +35 to +55 V power supply

- terminal for interactive control

- personality card for the EPROM to be programmed

- erased EPROM

This configuration is shown graphically in Figure 7-1.

7.3 CONSIDERATIONS

- The EPROM Programmer requires a 35 to 55 volt power supply. The TM 990/518 power supply provides this power requirement.

- Bits on the EPROM can be mistakenly programmed if the EPROM is inserted in the personality card attached to the TM 990/302 board while EPROM programming voltage is applied to the board and data is being read from a cassette. This necessitates that the program to be programmed on the EPROM be read from cassette while the EPROM is not connected to the TM 990/302 board or while EPROM programming power is not connected to the board.

7.4 EPROM ERASURE PROCEDURE

The EPROM can be erased by exposing the chip to ultraviolet light (wavelength of 2537 angstroms) through the transparent window on the chip. Recommended exposure is ten watt-seconds per centimeter which is equivalent to approximately 30 minutes exposure to a filterless model S52 short wave ultraviolet lamp approximately 2.5 centimeters above the EPROM. EPROM erasure state is all ones.

7.5 SYSTEM SETUP

7.5.1 EPROM PERSONALITY CARD

An EPROM personality card attaches to the TM 990/302 SDB board at connector P3 as shown in Figure 7-2. This card provides the socket to hold the EPROM as well as interface circuitry between the SDB and the EPROM.

There are two personality cards available for the TM 990/302 SDB. As shown in Table 7-1, each card is used for programming more than one EPROM type.

TABLE 7-1. PERSONALITY CARD CHARACTERISTICS

FOR PROGRAMMING	PART NUMBER
TMS 2708 and TMS 2716	TM 990/514
TMS 2508, TMS 2516, and TMS 2532	TM 990/515

To insert the personality card, press the female connector on the back of the personality card on to connector P3 (left side of the TM 990/302 Board). Note the top and bottom of card as shown in Figure 7-2. LED DS1 will glow when the personality card is properly attached.

7.5.2 INSERT PROM INTO PERSONALITY CARD, DESIGNATE PROM MODEL

Insert the EPROM to be programmed into the personality card as shown in Figure 7-2. Align the pins with pin one of the EPROM in the top right (facing the card) of the socket on the personality card. Take care to prevent pins from being bent.

Two personality cards, as shown in Table 7-1, are used to hold two different EPROM models. To designate which model is being programmed, position the three jumpers on the right side of the card onto the pins corresponding to the PROM model as shown in Table 7-2.

TABLE 7-2. JUMPER PLACEMENT ON PERSONALITY CARD

EPROM MODEL	CARD NUMBER	JUMPER PLACEMENT ON CARD
TMS 2708	TM 990/514	2708
TMS 2716	TM 990/514	2716
TMS 2508	TM 990/515	2508
TMS 2516	TM 990/515	2516
TMS 2532	TM 990/515	2532
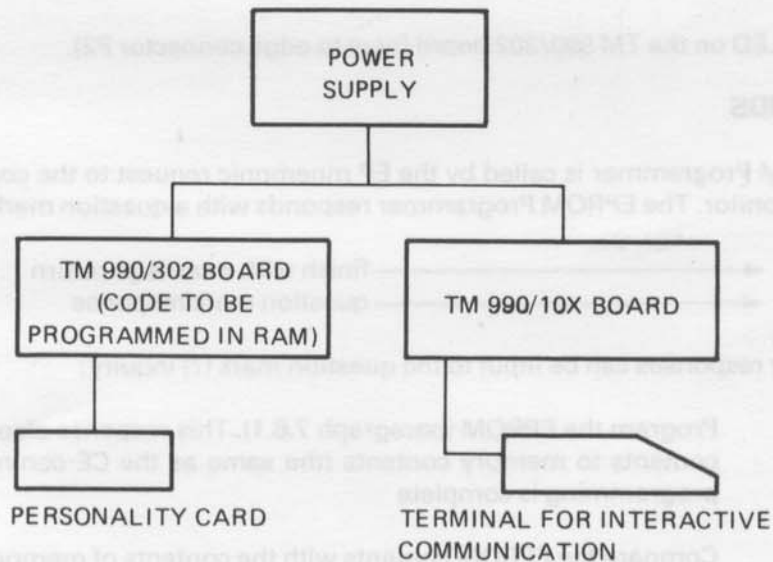
Figure 7-1. Typical EPROM Programming Configuration

POWER SUPPLY

TM 990/302 BOARD (CODE TO BE PROGRAMMED IN RAM)

TM 990/10X BOARD

PERSONALITY CARD

TERMINAL FOR INTERACTIVE COMMUNICATION
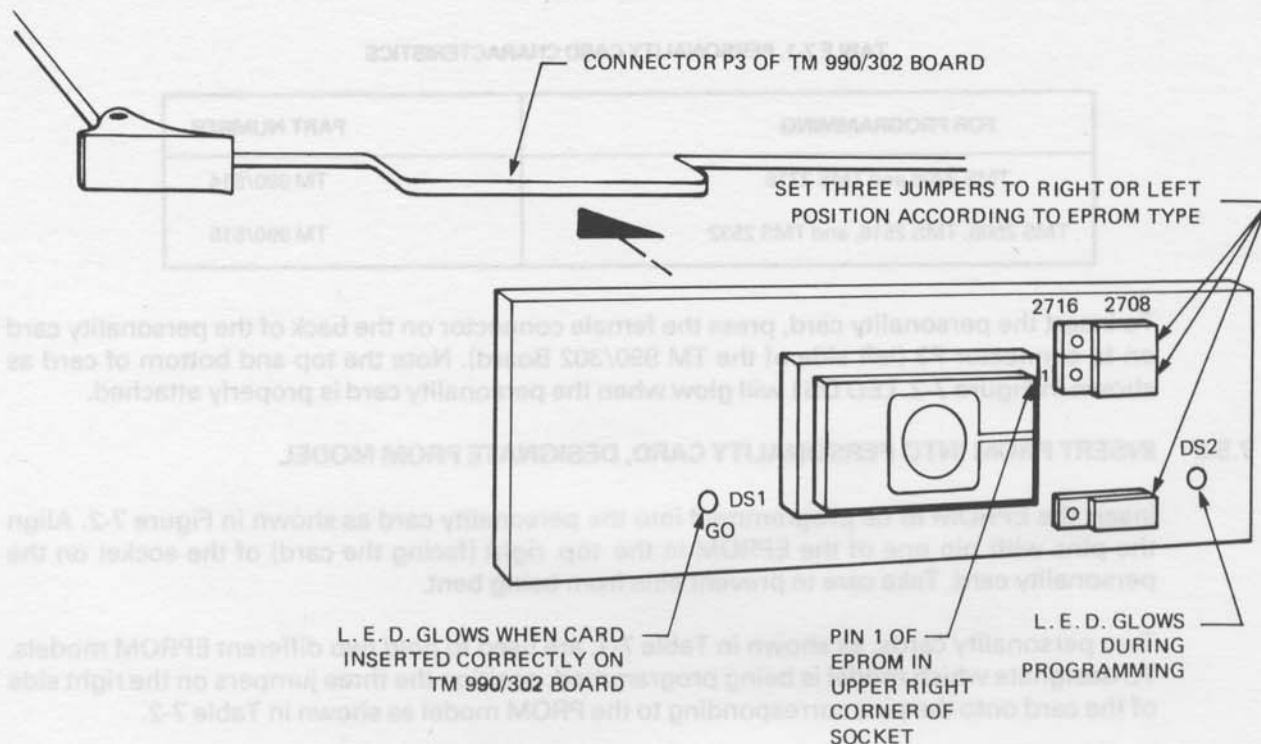
Figure 7-2. Personality Card

7.5.3 PERSONALITY CARD LED'S

If the personality card is inserted incorrectly (e.g., upside down), the LED marked GO on the card will be extinguished. When inserted correctly, this light will glow.

While the EPROM is being programmed, the following LED's will illuminate:

- LED on the right side of the personality card

- LED on the TM 990/302 board (next to edge connector P2).

7.6 COMMANDS

The EPROM Programmer is called by the EP mnemonic request to the command scanner of the SDB monitor. The EPROM Programmer responds with a question mark:

```
.EP  ◄──────────────────── finish with a carriage return
?    ◄──────────────────── question mark response
```

One of four responses can be input to the question mark (?) inquiry:

- PP   Program the EPROM (paragraph 7.6.1). This response also compares EPROM contents to memory contents (the same as the CE command below) when programming is complete

- CE   Compare the EPROM contents with the contents of memory; this can be used to verify data in a just-programmed PROM (paragraph 7.6.2)

● RE   Read the contents of the EPROM into memory; this can also be used for PROM data verification (paragraph 7.6.3)

● VE   Verify that the EPROM is erased (all ones) (paragraph 7.6.4)

Commands PP, CE, and RE use the same five- to six-field format as explained in paragraph 7.6.1 for the PP command. Parameters in the second line are completed with a carriage return.

NOTE

If the second line parameters cannot be inter-preted by the EPROM Programmer, the question mark prompt will be reissued.

7.6.1   PROGRAM THE EPROM COMMAND (PP)

The EPROM programming software is called by the mnemonic PP followed by five or six fields of description data in answer to the question mark inquiry on the terminal:

Format (see paragraph 1.7):

.EP ◄────────────────────── finish with a carriage return

?PP <EPROM type>$\left\{{\Lambda \atop ,}\right\}$<mem start>$\left\{{\Lambda \atop ,}\right\}$mem stop$\left\{{\Lambda \atop ,}\right\}$<EPROM start>$\left\{{\Lambda \atop ,}\right\}$<P/I>$\left\{{\Lambda \atop ,}\right\}$[byte start]<(CR)>

──────── no comma or space

The fields in the second line of the EPROM programming call require specific data before the EPROM programming will begin. When programming is complete, the EPROM Programmer will compare EPROM contents to memory contents, and give a comparison result similar to the CE command (paragraph 7.6.2). The fields in the second line are explained below in their order of appearance. After each explanation is an example using the command fields so far covered.

?PP EPROM type This field specifies the four-digit number of the EPROM to be programmed (2708, 2716, 2508, 2516, or 2532). The software will then verify that the correct personality card is being used and the correct jumpers are in place on the personality card for the EPROM being programmed (see paragraphs 7.5.1 and 7.5.2 for personality card setup). For example:

.EP
?PP2708,

NOTE

If the card or jumpers are not correct, the question mark prompt will be reissued.

Mem Start   This field specifies the start address in memory that contains the program to be programmed in the EPROM. This is a hexadecimal number *not* preceded by a greater-than sign. For example:

.EP
?PP2708,1000,

Mem Stop	This field specifies the last address in memory that contains the program to be programmed in the EPROM. This is a hexadecimal number *not* preceded by a greater-than sign. For example:

.EP
?PP2708,1000,102C,

program the EPROM with the 45 bytes in
M.A. 1000_{16} to $102C_{16}$

EPROM Start	This field specifies the EPROM address of the first byte in the EPROM to be programmed.

This allows partial programming of the EPROM (thus allowing programming of the EPROM in stages), if desired, since only the program area specified from the Mem Start to the Mem Stop fields will be programmed at any one time. The number must be hexadecimal, *not* preceded by a greater-than sign. If the address is too large, programming will be repeated. There is no provision to determine if any of the EPROM area has already been programmed. There is no need to specify an EPROM ending address, since this will be determined from the memory area containing the code (determined from the Mem Stop-Mem Start value). For example:

.EP
?PP2708,1000,102C,4C,

program the EPROM with the contents of memory
addresses 1000_{16} to $102C_{16}$ beginning at EPROM address
$004C_{16}$

P/I	In this field, place an I for in-line mode or a P for parallel mode. These two modes are shown graphically in Figure 7-3. In the in-line mode, the EPROM is programmed with continuous contents of the memory addresses specified. In parallel mode, the EPROM is programmed with either the even or odd memory address contents (this allows 16-bit machine code to be contained in two 8-bit even- and odd-byte EPROM sections, the addressing configuration of TM 990 microcomputer boards).

If the I mode is given, EPROM programming begins in the in-line mode immediately. If the P mode is given, another inquiry follows (this inquiry). For example:

.EP
?PP2708,1000,102C,4C,I ◄──────────── finish with CR

NOTE

In-line mode with programming execution begins when carriage return is executed; if P (parallel) mode given, one more inquiry is issued (explained below, page 7-8).

RAM MEMORY START BYTE

MEMORY EVEN BYTE

MEMORY ODD BYTE

EPROM BYTES

EPROM START

MEMORY LIMIT
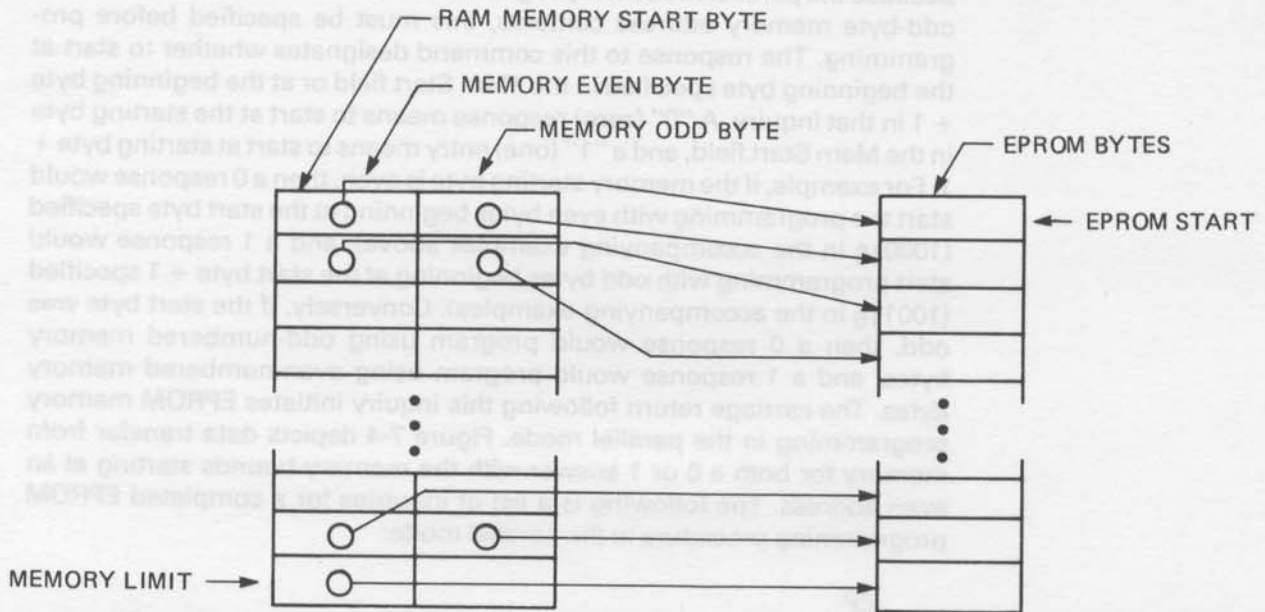
(a) In-Line EPROM Programming

RAM MEMORY START BYTE

EPROM BYTES

EPROM START

MEMORY LIMIT
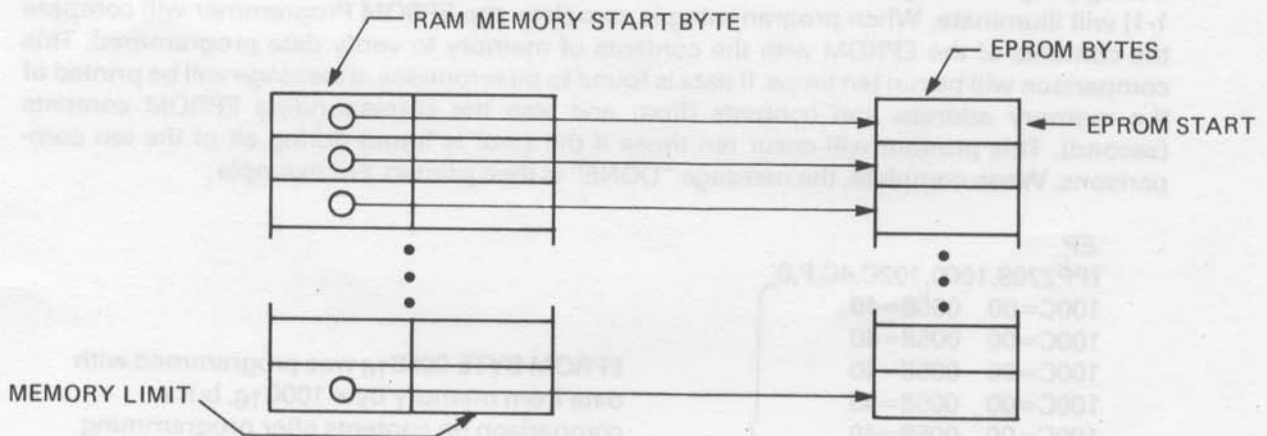
(b) Parallel EPROM Programming
(Either Even or Odd Memory Bytes
Programmed on EPROM)

Figure 7-3. In-Line And Parallel EPROM Programming

Byte Start   This field is necessary only if a "P" response is entered in the P/I field. Because the parallel mode will program the EPROM with either the even- or odd-byte memory address contents, this must be specified before programming. The response to this command designates whether to start at the beginning byte specified in the Mem Start field or at the beginning byte + 1 in that inquiry. A "0" (zero) response means to start at the starting byte in the Mem Start field, and a "1" (one) entry means to start at starting byte + 1. For example, if the memory starting byte is even, then a 0 response would start the programming with even bytes beginning at the start byte specified (1000_{16} in the accompanying examples above), and a 1 response would start programming with odd bytes beginning at the start byte + 1 specified (1001_{16} in the accompanying examples). Conversely, if the start byte was odd, then a 0 response would program using odd-numbered memory bytes, and a 1 response would program using even-numbered memory bytes. The carriage return following this inquiry initiates EPROM memory programming in the parallel mode. Figure 7-4 depicts data transfer from memory for both a 0 or 1 answer with the memory bounds starting at an even address. The following is a list of inquiries for a completed EPROM programming procedure in the parallel mode:

.EP
?PP2708,1000,102C,4C,P,0 ◄─────────────── followed by a CR

└─beginning at EPROM address $004C_{16}$, program in the contents of the even memory addresses from 1000_{16} to $102C_{16}$

During programming, the LED's on the personality (Figure 7-2) and TM 990/302 card (Figure 1-1) will illuminate. When programming is complete, the EPROM Programmer will compare the contents of the EPROM with the contents of memory to verify data programmed. This comparison will be run ten times. If data is found to be erroneous, a message will be printed of the memory address and contents (first) and also the corresponding EPROM contents (second). This printout will occur ten times if the error is found during all of the ten comparisons. When complete, the message "DONE" is then printed. For example:

.EP
?PP2708,1000,102C,4C,P,0
100C=00 0058=40
100C=00 0058=40
100C=00 0058=40   EPROM BYTE 0058_{16} was programmed with
100C=00 0058=40   data from memory byte $100C_{16}$, but a
100C=00 0058=40   comparison on contents after programming
100C=00 0058=40   shows the data differs. Error found
100C=00 0058=40   during all ten comparisons.
100C=00 0058=40
100C=00 0058=40
100C=00 0058=40
DONE
?

7-8
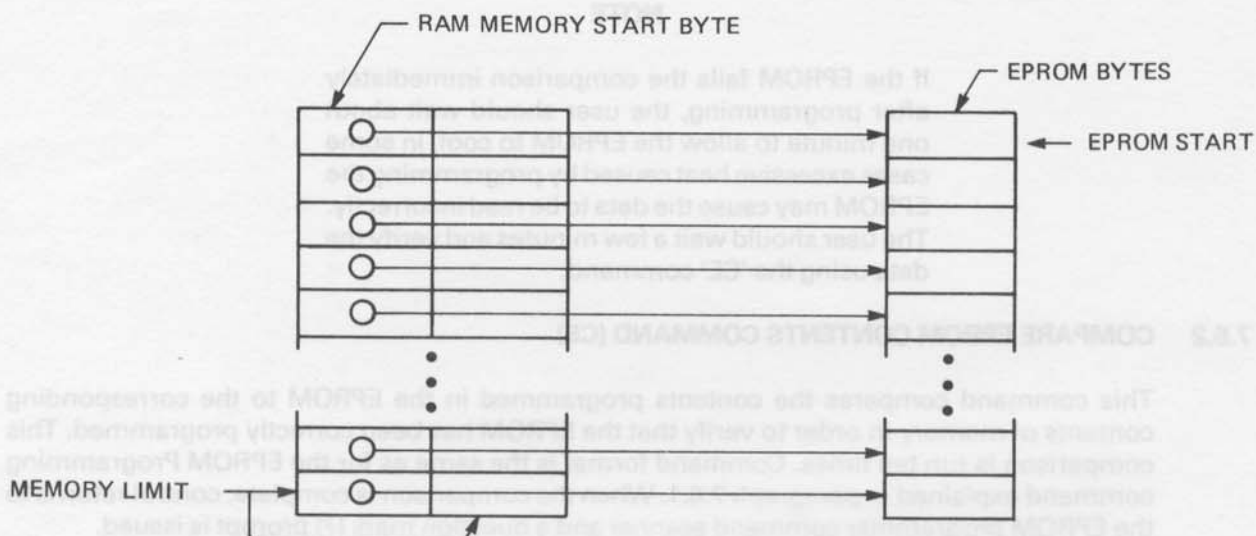
(a) Program Even Memory Bytes into EPROM
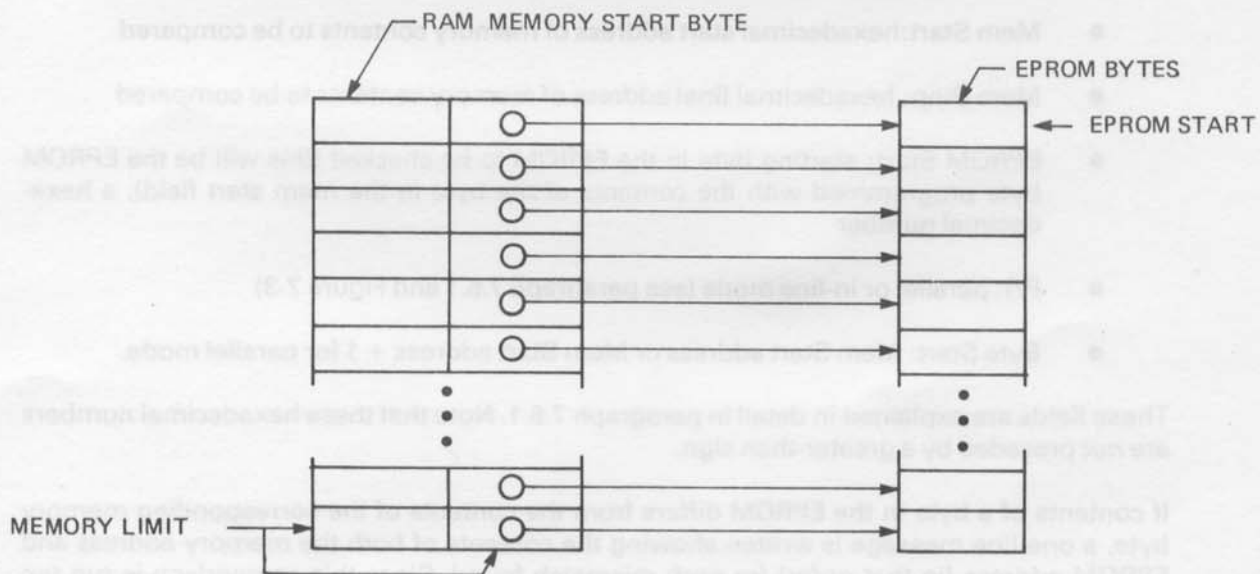
(b) Program Odd-Numbered Memory Bytes into EPROM

Figure 7-4. Data Transfer In Parallel Programming Mode

NOTE

If the EPROM fails the comparison immediately after programming, the user should wait about one minute to allow the EPROM to cool. In some cases excessive heat caused by programming the EPROM may cause the data to be read incorrectly. The user should wait a few minutes and verify the data using the 'CE' command.

7.6.2   COMPARE EPROM CONTENTS COMMAND (CE)

This command compares the contents programmed in the EPROM to the corresponding contents of memory in order to verify that the EPROM has been correctly programmed. This comparison is run ten times. Command format is the same as for the EPROM Programming command explained in paragraph 7.6.1. When the comparison is complete, control returns to the EPROM programmer command scanner and a question mark (?) prompt is issued.

7.6.2.1 Format

?CE<EPROM type> $\left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\}$ <mem start> $\left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\}$ <mem stop> $\left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\}$ <EPROM start> $\left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\}$ <P/I> $\left\{ \begin{smallmatrix} \wedge \\ , \end{smallmatrix} \right\}$ [byte start]<(CR)>

— no comma or space

* EPROM type: 2708, 2716, 2508, 2516, or 2532

* Mem Start: hexadecimal start address of memory contents to be compared

* Mem Stop: hexadecimal final address of memory contents to be compared

* EPROM Start: starting byte in the EPROM to be checked (this will be the EPROM byte programmed with the contents of the byte in the mem start field), a hexadecimal number

* P/I: parallel or in-line mode (see paragraph 7.6.1 and Figure 7-3)

* Byte Start: Mem Start address or Mem Start address + 1 for parallel mode.

These fields are explained in detail in paragraph 7.6.1. Note that these hexadecimal numbers are *not* preceded by a greater-than sign.

If contents of a byte in the EPROM differs from the contents of the corresponding memory byte, a one-line message is written showing the contents of both the memory address and EPROM address (in that order) for each mismatch found. Since this comparison is run ten times, the error message can be written that many times. When the entire compare process is complete, the message "DONE" is printed and control returns to the SDB command scanner.

7.6.2.2 Examples

```
(1) .EP                              finish with carriage returns
    ?CE2708,1000,102C,4C,P,0
     100B=40   0057=00               comparison discrepancies found during
     100B=40   0057=00                 all ten comparisons.
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
     100B=40   0057=00
    DONE
    ?         ─memory──── ── EPROM
               addr. and      addr. and
               contents       contents
```

```
(2) .EP ◄────────────────────────── finish with carriage return
    ?CE2716,2000,2400,0,I
    DONE ◄──────────────────────── EPROM/memory comparison okay
    ?                               return to programmer command scanner
```

7.6.3 READ EPROM CONTENTS INTO MEMORY COMMAND (RE)

This command reads the contents in the EPROM into a designated memory area. This can be
used with the Dump Memory command (Section 8) in order to store object on cassette for later
debugging, EPROM programming, etc. Command format is the same as for the EPROM
Programming command explained in paragraph 7.6.1. No check is made to verify correct data
transfer. When data transfer is complete, the word "DONE" is printed out and control returns
to the SDB monitor and a period (.) prompt is issued.

7.6.3.1 Format

?RE<EPROM type>$\left\{{\Lambda \atop ,}\right\}$<mem start>$\left\{{\Lambda \atop ,}\right\}$<mem stop>$\left\{{\Lambda \atop ,}\right\}$ <EPROM start> $\left\{{\Lambda \atop ,}\right\}$<P/I>$\left\{{\Lambda \atop ,}\right\}$[byte start]<(CR)>

   ── no comma or space

- EPROM Type: 2708, 2716, 2508, 2516, 2532, or 9940

- Mem Start: hexadecimal start address of memory area to be written to

- Mem Stop: hexadecimal final address of memory area to be written to

- EPROM Start: starting byte in the EPROM to be read into memory (hexadecimal value)

- P/I: parallel or in-line mode (see paragraph 7.6.1 and Figure 7-3)

- Byte Start: mem start address or mem start address + 1 for parallel mode.

These fields are explained in detail in paragraph 7.6.1.

Only the memory area specified will be written to with data from the EPROM beginning at the EPROM address specified. If the memory area is larger than the EPROM contents to be stored, memory bytes not written to will not be altered. If the P mode (parallel) is requested, data transfer will be from the EPROM bytes to:

- every other byte beginning at the EPROM start address if a 0 (zero) is specified in the "Byte Start" field

- every other byte beginning at the EPROM start address + 1 if a 1 (one) is specified in the "Byte Start" field.

7.6.3.2 Examples

(1) .EP
   ?RE2516,2800,29FF,0,I   finish with a carriage return

   DONE
   ?   transfer the EPROM contents, in inline mode, beginning at EPROM address 0, to memory addresses 2800_{16} to $29FF_{16}$

(2) In this example, transfer the contents of two TMS 2708 EPROM's (1K by 8 each) into memory. The EPROM's have been programmed in the parallel mode; thus, one will be read into the even-number bytes and the other read into the odd-number bytes, beginning at memory address 1000.

   .EP
   ?RE2708,1000,17FE,0,P,0

   DONE   load at M.A. 1000_{16}, 1002_{16}, 1004_{16},...,$17FE_{16}$
   .EP
   ?RE2708,1000,17FF,0,P,1

   DONE   load at M.A. 1001_{16}, 1003_{16}, 1005_{16},...,$17FF_{16}$
   ?   return to command scanner
Note: the second EPROM reading sequence could also have been:
   ?RE2708,1001,17FF,0,P,0

7.6.4 VERIFY EPROM AREA IS ERASED COMMAND (VE)

This command checks designated memory addresses in the EPROM for an erased condition (all ones). The format consists of the mnemonic VE followed by the EPROM type and the hexadecimal beginning and ending addresses of the EPROM area to be verified (*no* greater-than sign precedes the hexadecimal number). The command is completed with a carriage return. As the case with the CE command (paragraph 7.6.2), the specified condition is checked for ten times, and discrepancies are printed out as many times as found. If non-erased bytes are found, the addresses and their contents are printed out; no printout indicates all addresses checked were erased. When the verification process is complete, the word "DONE" is printed out and control is returned to the SDB command scanner with a period (.) printed on the terminal.

7.6.4.1 Format

?VE<EPROM Type> {$^\wedge_,$} <Start Address> {$^\wedge_,$} <Ending Address> <(CR)>

└─ no comma or space

7.6.4.2 Examples

(1) .EP
    ?VE2716,0,7FF ◄─────────── finish with a carriage return
    0710=EF
    074B=FB
    0710=EF
    074B=FB   bytes not fully erased
    0710=EF   (found during all ten checks)
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    0710=EF
    074B=FB
    DONE
    ? ◄─────────────── return to command scanner

(2) .EP
    ?VE2708,400,7FF
    DONE   all bytes found erased
    ?

SECTION 8

DUMP MEMORY COMMAND

8.1 GENERAL

This program dumps object programs in memory to a cassette. The object programs are dumped in absolute format which is not relocatable in a later load by the Relocatable Loader (i.e., the object code will not have relocatable code tags; thus, to later reload and execute the code, it must be reloaded at the absolute address at which it can properly execute).

8.2 FORMAT

The format is on two lines. The first line is in response to the period (.) prompt by the monitor. This line specifies the Dump Memory command and the DEVNO of the cassette to which the object will be dumped; it is executed by a carriage return. The second line is in response to the question mark (?) prompt by the Dump Memory command. This line contains the memory bounds of the object to be dumped: the start and stop address of the object, both addresses should be even values. Both lines are completed by carriage returns, and delimiters between fields can be commas or spaces. Addresses must be hexadecimal without the greater-than sign (>).

.DM $\left\{ {}^{\wedge}_{,} \right\}$ <DEVNO> <(CR)>

? <start address> $\left\{ {}^{\wedge}_{,} \right\}$ <stop address> <(CR)>

8.3 EXAMPLE

.DM 2   ⎱ Both lines completed
?1000,14FE   ⎰ with carriage returns

— Dump to cassette 1 the object program contained in memory addresses 1000_{16} to $14FE_{16}$

8-1

SECTION 9

SETTING BAUD RATE AT SECOND EIA PORT

9.1 GENERAL

There are two kinds of CPU boards (TM 990/100M and TM 990/101M) that can be used in the TM 990/302 system. One major difference between these two CPU boards is that two EIA connectors are provided on the TM 990/101M while only one EIA connector is provided on the TM 990/100M board. If the TM 990/101M board is used, the set rate (SR) command is needed to set up the proper baud rate for the second EIA port (port P3, the middle port on the outside edge when inserted in the card cage). In order to use this port, the user must specify the port baud rate using the SR (set rate) command.

9.2 CONSIDERATIONS

On the second EIA port, the data format is fixed for one start bit, two stop bits, even parity, and seven data bits. The only thing the user needs to set up is the baud rate. The available baud rates are: 110, 300, 1200, 2400, 4800, 9600, and 19,200 Hz.

9.3 FORMAT

.SR <(CR)
? <BAUD RATE> <(CR)>

9.4 EXAMPLES

(1) .SR
     ?19200   Set to 19.2K baud

(2) .SR
     ?1200   Set to 1200 baud

9.5 ERROR CODE

If a nonsupported rate is specified in the SR command, the following message is output:

   **ERROR 91

To recover, re-execute the SR command with one of the baud rates specified in paragraph 9.2.

10.1   GENERAL

There are several user callable utilities in the TM 990/302 system software. These utilities can provide very important assistance to the user in debugging and be of use when coding an exercise program. The entries to these utilities are through BLWP instructions. A table of BLWP vectors is located in lower system software ROM. The utilities use their own workspaces.

10.2   CONSIDERATIONS

Since the entry of the utilities require BLWP vectors, any attempt to get into the utilities with other than the required BLWP instruction will cause an unknown result. These utilities are located at memory addresses 80_{16} through $3FF_{16}$; thus these locations are reserved.

10.3   UTILITIES

10.3.1 TM 990/302 RETURN TO SYSTEM SOFTWARE

This is not a utility as much as it is an entry point to the TM 990/302 system software. It provides a convenient way of returning from user's program to TM 990/302 system program.

Calling sequence:

BLWP @>E000

10.3.2 DECIMAL ASCII TO BINARY CONVERSION

This routine converts the signed decimal ASCII number to a binary value in two's complement form. The conversion range is from −32768 to +32767. The conversion stops when a character other than a decimal ASCII character (hexadecimal 30 to 39 following the sign character) is detected in the number string. Before it is called, register R1 should contain the ASCII-number buffer address. After completion, register R0 contains the binary result and R1 points to one byte after the last ASCII digit.

Calling sequence:

BLWP @>E004

Example:

	LI	R1,BUFF	R1 ponts to the ASCII buffer
	BLWP	@>E004	Do a conversion
	MOV	R0,R10	Put result into R10
BUFF	TEXT	'−12366'	Decimal ASCII character string
	BYTE	0	Terminate string with a non-decimal ASCII

10.3.3 HEXADECIMAL ASCII TO BINARY CONVERSION

This routine converts a hexadecimal number to a binary value. The conversion range is from 0 to $FFFF_{16}$. The conversion stops when a character other than a hexadecimal ASCII character (hexadecimal 30 to 39 and 41 to 46) is detected in the number string. Before it is called, register R1 should contain the buffer address of the hex ASCII number. After it is returned, R0 contains the binary result and R1 points to one byte after the last ASCII digit.

Calling sequence:

    BLWP @>E008

Example:

	LI	R1,BUFF	R1 points to ASCII buffer
	BLWP	@>E008	Do a conversion
	MOV	R0,R10	Put result into R10
BUFF	TEXT	'1BA'	Hexadecimal ASCII character string
	BYTE	0	Terminate string with a non-hexadecimal ASCII

10.3.4 BINARY TO DECIMAL ASCII CONVERSION

This routine converts a binary value in two's complement form to a signed decimal ASCII number. The ASCII result is terminated with a 'space' (hexadecimal 20). Before it's called, R0 should contain the binary value and R1 the ASCII buffer address. After it's returned R1 points to one byte after last digit.

Calling sequence:

    BLWP @>E00C

Example:

	MOV	@NUM,R0	Place binary number in R0
	LI	R1,BUFF	R1 points to buffer
	BLWP	@>E00C	Do a conversion, decimal ASCII in BUFF
NUM	DATA	1234	

10.3.5 BINARY TO HEXADECIMAL ASCII CONVERSION

This routine converts a binary value to a hexadecimal ASCII number. The ASCII result is terminated with a 'space' (hexadecimal 20) which is the string delimiter for the print ASCII routine. Before the conversion is called, R0 should contain the binary number and R1 should contain the hex ASCII buffer address. After it's returned, R1 points to one byte after the last digit.

Calling sequence:

    BLWP @>E010

Example:

```
        MOV    @NUM,R0      R0 contains hex number
        LI     R1,BUFF      R1 points to buffer
        BLWP   @>E010       Do a conversion, decimal ASCII in BUFF

NUM     DATA   >FFB0
```

10.3.6 ECHO CHARACTER ON THE PRIMARY EIA PORT

This routine inputs a character from the EIA terminal and echoes it back to the primary EIA port (port P2 on the microcomputer). The character is stored in the high order byte of R0.

Calling sequence:

```
BLWP   @>E014
```

10.3.7 OUTPUT A CHARACTER TO THE PRIMARY EIA PORT

This routine outputs an ASCII character which is stored in the high order byte of R0. The character is output at EIA port P2 on the microcomputer board.

Calling sequence:

```
BLWP   @>E018
```

Example:

```
LI     R0,>3000     R0 contains an ASCII0 in high order byte
BLWP   @>E018       Output it
```

10.3.8 OUTPUT A MESSAGE TO THE PRIMARY EIA PORT

This routine outputs a string of ASCII characters to the primary EIA port (port P2 on the microcomputer). The output is terminated by a byte of zeroes (null character). Before execution, R0 must contain the address of the message.

Calling sequence:

```
BLWP        @>E01C
```

Example:

```
        LI          R0,MSSG          Address of message in R0
        BLWP        @>E01C

MSSG    TEXT 'IT SURE IS EASY TO USE'    Message
             BYTE 0                      Null delimiter
```

10.3.9 INPUT UP TO 80 CHARACTERS FROM PRIMARY EIA PORT

This routine inputs up to 80 characters from the P2 port of the TM 990/10X microcomputer. The character string ends with a carriage return. Characters are stored beginning at M.A. 0110_{16}.

Calling sequence:

```
BLWP        @>E020
```

SECTION 11

UPLINK BETWEEN TM 990/302 AND HOST COMPUTER

11.1 GENERAL

The TM 990/302 Uplink program allows the Software Development System to be used as an ASR 733 emulator to communicate with a host computer such as the TI 990/10 or TI 990/4. This feature allows the use of a more powerful computer when doing program development such as text editing, assembly, link editing, etc. Then the object can be written directly to the TM 990/302 for final debugging in the test bed environment in which the program will operate. Object program transfer is through a host computer utility such as copy/concatenate. The Uplink program operates in two basic modes:

TM 990/506 CABLE:
CONNECTOR P1 TO P3 OF TM 990/101M MICROCOMPUTER
CONNECTOR P3 TO HOST COMPUTER EIA BOARD
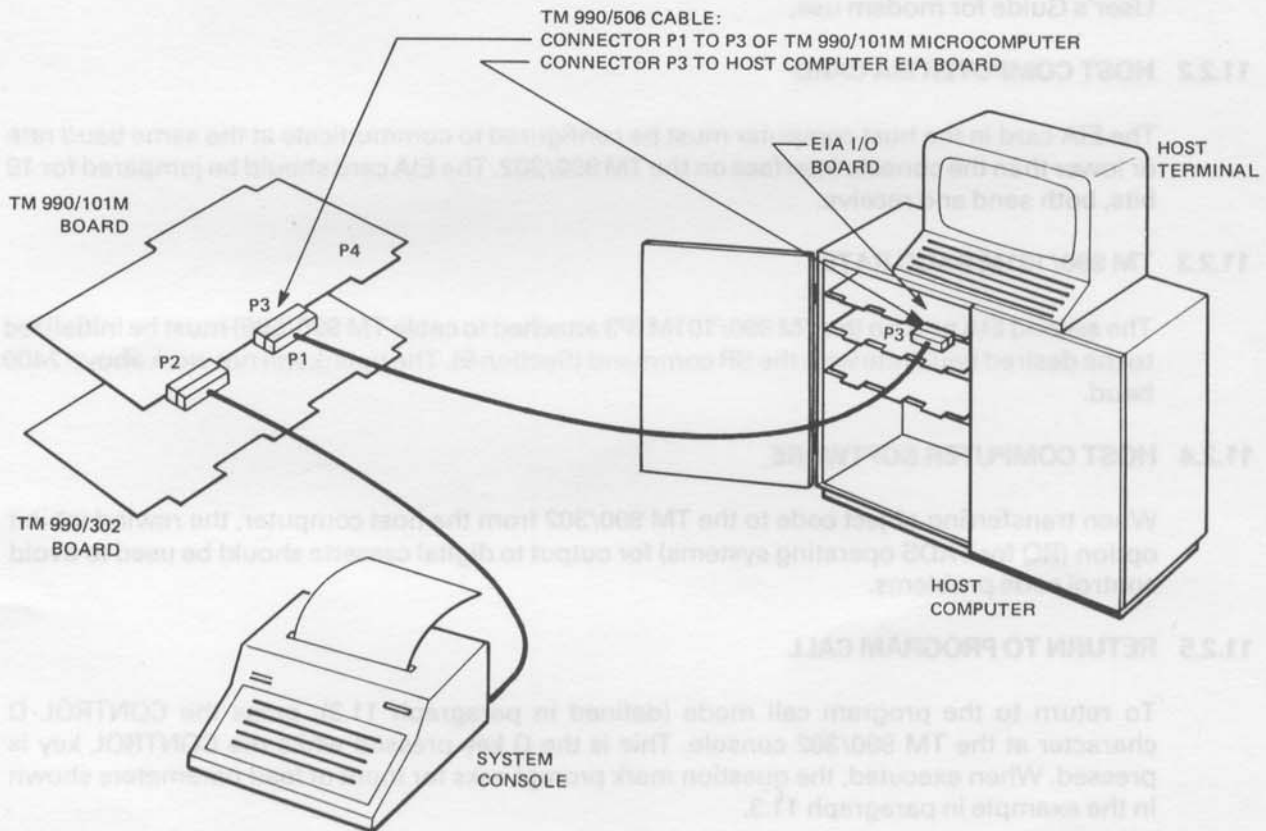
Figure 11-1. System Configuration For Uplink Program

- Terminal Mode in which the host computer is accessed and controlled through the Software Development System keyboard (paragraph 11.4).

- Load Mode in which the TM 990/302 acts as a passive 733 ASR terminal, accessed (such as for storage on a cassette) through the keyboard of the host computer (paragraph 11.5).

11.2 SYSTEM CONFIGURATION AND EXECUTION CONSIDERATIONS

Figure 11-1 shows the system configuration. The TM 990/302 and TM 990/101M boards are interconnected through the card cage backplane. The host computer's EIA interface board is connected to the secondary EIA port on the TM 990/101M via the TM 990/506 cable (this secondary microcomputer port is necessary for system operation with the Uplink program). Note that only the TM 990/101M microcomputer can be used.

11.2.1 TM 990/506 CABLE

The TM 990/506 cable should be hooked up with cable connector P1 connected to connector P3 of the TM 990/101M microcomputer, and with cable connector P3 connected to the 25-pin EIA port on the host computer. This cabling is a reverse of the configuration in the TM 990/506 User's Guide for modem use.

11.2.2 HOST COMPUTER EIA CARD

The EIA card in the host computer must be configured to communicate at the same baud rate or lower than the console interface on the TM 990/302. The EIA card should be jumpered for 10 bits, both send and receive.

11.2.3 TM 990/101M BAUD RATE

The second EIA port on the TM 990/101M (P3 attached to cable TM 990/506) must be initialized to the desired baud rate with the SR command (Section 9). The uplink will not work above 2400 baud.

11.2.4 HOST COMPUTER SOFTWARE

When transferring object code to the TM 990/302 from the host computer, the rewind inhibit option (RO for TXDS operating systems) for output to digital cassette should be used to avoid control code problems.

11.2.5 RETURN TO PROGRAM CALL

To return to the program call mode (defined in paragraph 11.3), press the CONTROL D character at the TM 990/302 console. This is the D key pressed while the CONTROL key is pressed. When executed, the question mark prompt asks for input of load parameters shown in the example in paragraph 11.3.

11.2.6 RETURN TO MONITOR

To return to the Software Development System monitor press the CONTROL C character at the TM 990/302 console. This is the C key pressed while the CONTROL key is pressed. When control is passed to the monitor, the host computer and the TM 990/302 are no longer communicating with each other.

11.3 UPLINK PROGRAM CALL

When the UL command is entered at the TM 990/302 console, the command mode is entered and a question mark (?) prompt is issued asking for entry of load parameters, the same as required by the Relocating Loader (Section 5).

Format:

.UL ◄─────────────────────────────── complete with carriage return

?<load address> <,> <load bias> <,> <load length> <,> <start address> <(CR)>

                       └── address within the program to start load

                └── number of bytes to load (maximum)

         └── bias to be added to relocatable code

  └── beginning address where to store program on TM 990/302 board

The second-line parameters are the same as for the Relocating Loader program, explained in paragraph 5.1.

When the second line parameters are concluded with a carriage return, the TM 990/302 goes into the Terminal Mode (paragraph 11.4).

Example:

```
.SR
?1200        }  set baud rate with
.UL              SR command (Section 9)
?1000,0,600,0 ◄─────── loading parameters
```

NOTE

Control may revert back to the program call by entering a CONTROL D (EOT, press the D key while CONTROL key is pressed); the question mark prompt will appear asking for insertion of new load parameters.

11.4 TERMINAL MODE

In the Terminal Mode, system control is maintained at the Software Development Board console attached to the TM 990/101M board. In this configuration, the TM 990/302 should appear like a 733 ASR in that all data sent to the host computer will be echoed back to the TM 990/302 and printed on the terminal connected to the TM 990/101M. In effect, the TM 990/302 becomes transparent to the terminal/host communications. All functions at the host computer (assemblies, file manipulations, etc.) can be done at the 733 ASR through the TM 990/302. All entries would be the same as if they were entered at the host terminal.

Figure 11-2 and 11-3 are examples of executing the copy/concatenate program on the TI 990/10 and TI 990/4 host computers respectively. Both load the same object program to the TM 990/302.

```
 ┌ .SR:                                                         ── Set baud rate
302│ ?1200
Terminal│ .UL      ◄─────────────────────────────────────────── Call Uplink
 └ ?950,950,↑000,0  ◄──────────────────────────────┐
   ┌ SYSTEM COMMAND INTERPRETER - PLEASE LOG IN     └── Load parameters
   │      USER  ID: ■■■■■■
   │      PASSCODE: ■■■■■■
   │
   │ [] CC
Host │ COPY/CONCATENATE                               990/10 operating
Terminal│  INPUT ACCESS NAME (S): LBD4MAIN4OBJ          system executes,
   │  OUTPUT ACCESS NAME:     CS03                    object copied thru
   │  REPLACE?: NO                                    EIA port
   │ 5MAXIMUM RECORD LENGTH:
   │
   │
   │ COPY COMPLETED
   └ []  ◄──────────────────────────────────────── CONTROL C pressed at 302
 ┌ .DP ◄───────────────────────────────┐            terminal
302│ ?IR
Terminal│ W=0080                                         Call Debugger
 │ P=E04C    950                                    at TM 990/302
 │ S=D000
 └ ?EX
```

Figure 11-2. Uplink Program Execution, TI 990/10 As Host Computer

11-4

In this mode, the Uplink program executes as if the TM 990/302 is a 733 ASR with digital cassettes. The Uplink program is called as shown in paragraph 11.2, then control is maintained through the host computer console. If a transfer is initiated from the host computer to one of the cassette drives of the TM 990/302 EIA port, the TM 990/302 will collect one line of tagged object code, then use the Relocating Loader to place the object code into the memory of the TM 990/302.

Figure 11-4 is an example of loading object down from the TI 990/4 host computer to the TM 990/302 console. Note that if the RD option is not used, the object code will be printed on the TM 990/302 console. When using the operating system on the TI 990/10 computer, the RU option is not necessary.

```
              ┌ .SR
  302      │   ?1200          ◄───────────────────── Set baud
Terminal   │   .UL            ◄───────────────────── rate
           └   ?1000,0,1000,0 ◄─────────────        ┌── Call Uplink
                                                        Set load parameters

         ┌ TX990   SYSTEM        RELEASE 2.2 ◄────── Boot TX990 at host computer
         │  MEMORY SIZE(WORDS): 24576   AVAILABLE: 16361
         │  !                              ◄──────── No OCP, execute TX990
         │  TXDS   936215 ◄A      1/ 0    00:00
         │
Host     │  PROGRAM: :TXCCAT/SYS
Terminal │    INPUT: DSC2:MAIN4/OBJ                 ┐
         │   OUTPUT: CS1                            │ Execute object copy
         │  OPTIONS: RO                             │ through EIA port
         │  TXCCAT 937543 ◄◄                        ┘
         │
         │  TXDS   936215 ◄A      1/ 0   00:02
         │                                          CONTROL C at 302 terminal
         └ PROGRAM: ◄───────────────────────────    to revert to
           ┌ .DP                                    TM 990/302 monitor,
  302      │  ?IR                                    call Debugger
Terminal   │  W=0080
           │  P=E03A   1000
           │  S=D400
           └  ?EX
```

NOTE: Both host and TM 990/302 can use the same terminal.

Figure 11-3. Uplink Program Execution Using TX990 on Host Computer

After the TM 990/302 Uplink has received the specified amount of data, it will return to the monitor as shown in Figure 11-4 with the period (.) prompt. The host computer will return to its monitor or utility menu. After the object code has been loaded to the TM 990/302 board, the object can be debugged using the Program Debugger (explained in Section 8).

11.5 LOAD MODE

In this mode, the Uplink program executes as if the TM 990/302 is a 733 ASR with digital cassettes. The Uplink program is called as shown in paragraph 11.3, then control is maintained through the host computer console. If a transfer is initiated from the host computer to one of the cassette drives of the TM 990/302 EIA port, the TM 990/302 will collect one line of tagged object code, then use the Relocating Loader to place the object code into the memory of the TM 990/302.

Figure 11-4 is an example of loading object down from the TI 990/4 host computer to the TM 990/302. Note in this program that if the RO option is not used, the object code will be printed on the TM 990/302 console. When using the operating system on the TI 990/10 computer, the R0 option is not necessary.

```
At TM 990/302 Console:

.SR
?1200
.UL
?1000,0,1000,0


    At TI 990/4 Console:


    PROGRAM:  :TXCCAT/SYS
      INPUT:  DSC2:MAIN4/SYS
     OUTPUT:  CS1
    OPTIONS:  RO


    TXCCAT  937543  **


    TXDS  936215  *A      1/ 0    00:02

    PROGRAM:
```

Figure 11-4. Transferring Object In The Load Mode On TI 9904

After the TM 990/302 Uplink has received the specified amount of data, it will return to the monitor as shown in Figure 11-4 with the period (.) prompt. The host computer will return to its monitor or utility menu. After the object code has been down loaded to the TM 990/302 board, the object can be debugged using the Program Debugger (explained in Section 6)

APPENDIX A

WIRING TELETYPE MODEL 3320/5JE FOR TM 990/10XM

A-1 GENERAL

Figure A-1 shows the wiring configuration required to connect a 3320/5JE Teletype in a 20 mA current loop with a TM 990/10XM. Other teletypewriter models may require different connections; therefore, consult the manufacturer for correct wiring of other models. Teletypewriters can be used with Assembly No. 999211-0001 only.

CAUTION

Note the 117 Vac connection at pins 1 and 2. Be sure that this voltage is not accidently wired to the TM 990/10XM board.

A-2 CONNECTIONS

The following assumes that the teletypewriter is wired as it came from the factory.

(1) Locate the 151411 terminal block at the left rear (viewed from the rear) of the machine (Figure A-1).

(2) Move the white/blue wire from terminal 4 to terminal 5 on the terminal block.

(3) Move the brown/yellow wire from terminal 3 to terminal 5 on the terminal block.

(4) Move the purple wire from terminal 8 to terminal 9 on the terminal block (for 20 mA neutral signaling).

(5) Locate the power resistor behind the teletype power supply. Remove the blue wire from the 750 ohm tap and connect it to the 1450 ohm tap, as shown in Figure A-2.

(6) Check pins 3, 4, 6, and 7 at terminal strip 151411. Voltage to ground must be zero with power applied. If not, do not connect to the TM 990/10XM.

NOTE

For teletypewriter operation jumper J11 must be installed and J7 must be in the EIA position.

A-3   TROUBLESHOOTING

If the printer continues to chatter after the RESET switch on the TM 990/10XM has been activated, reverse connections 6 and 7 at the terminal strip.
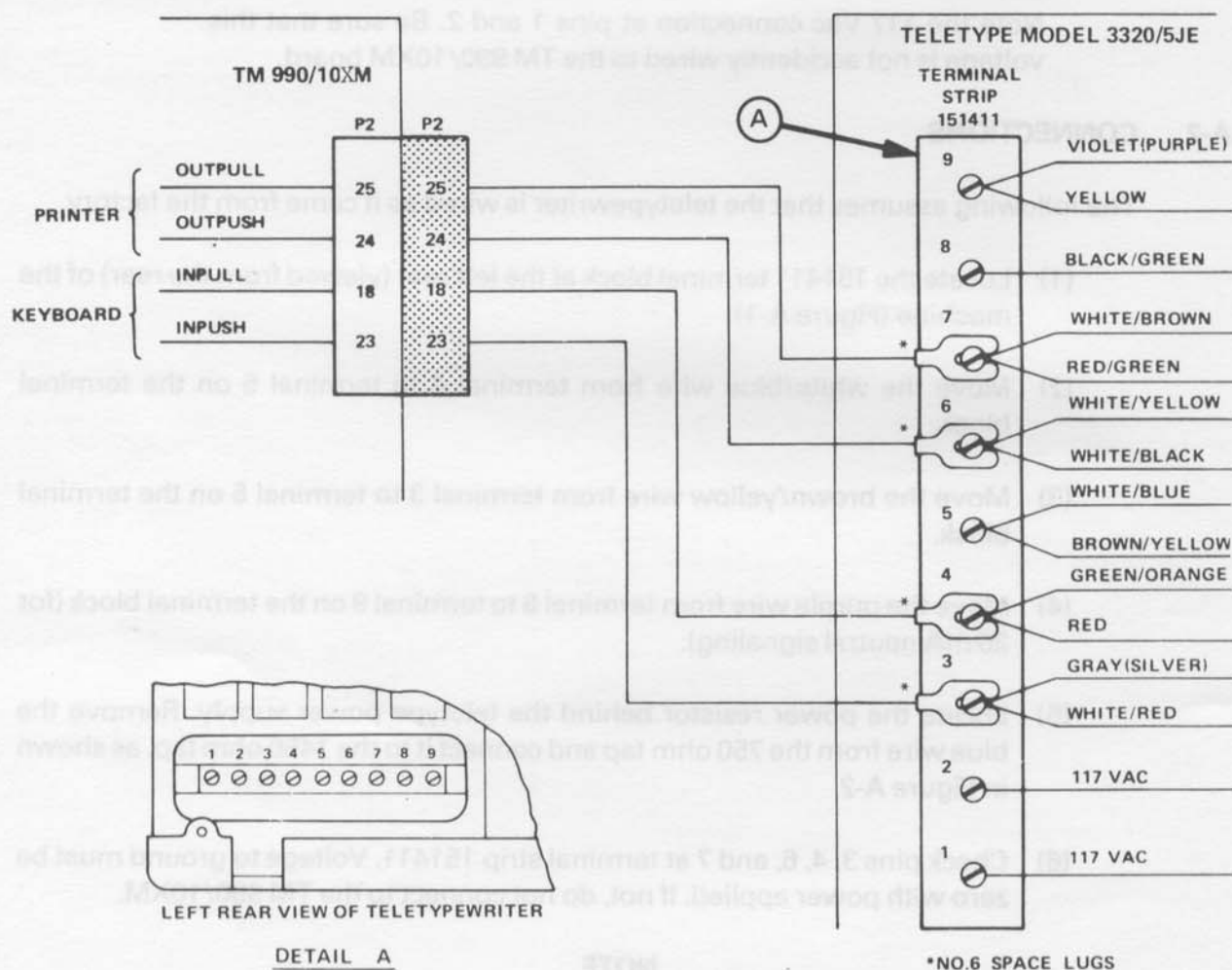
Figure A-1. Teletypewriter Terminal Strip Connections

FRONT

1450 OHM TAP
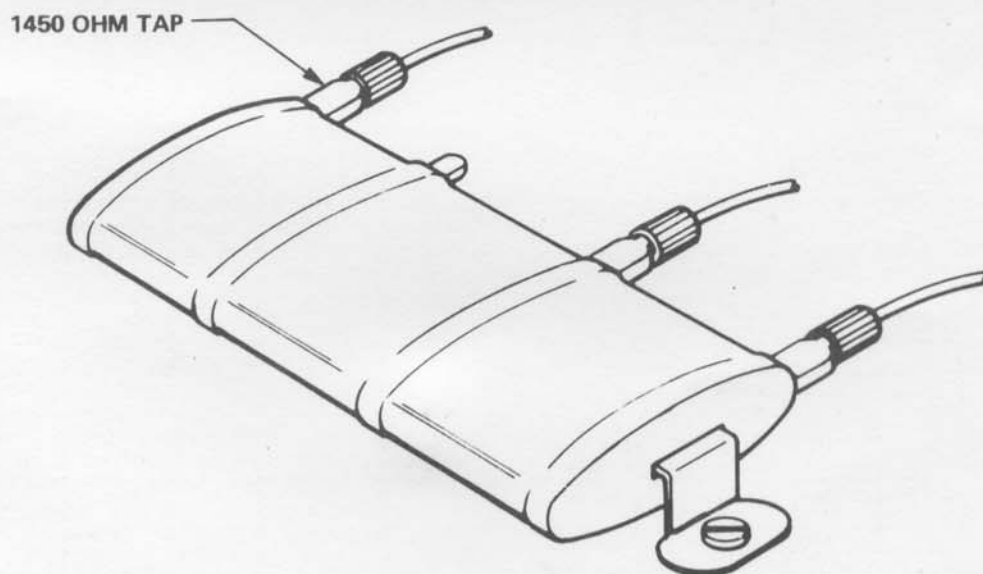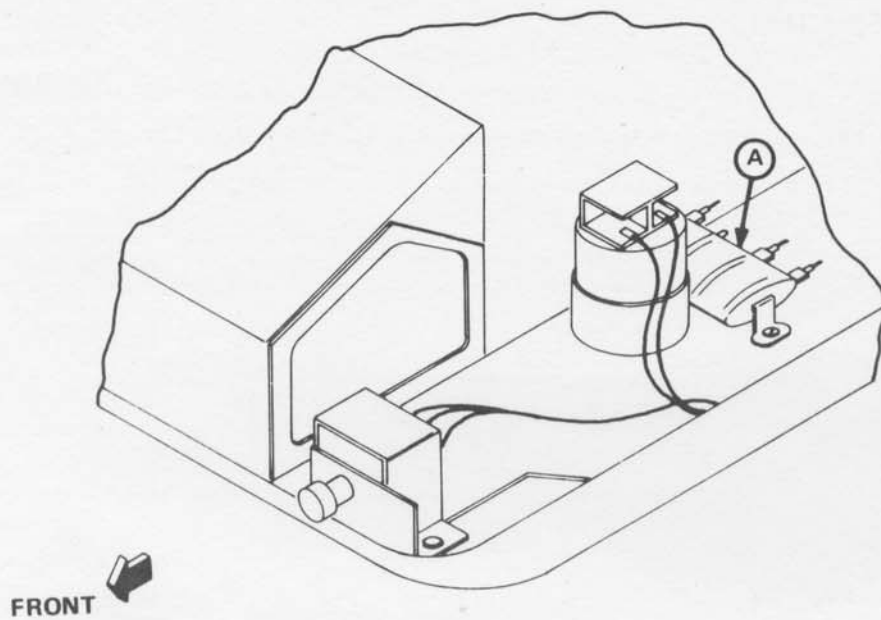
DETAIL A

Figure A-2. Teletypewriter Resistor Connection

APPENDIX B

EIA RS-232-C CABLING

Figure B-1 shows the wiring for the 743 KSR cable attached between connector P2 on the TM 90/10XM and a 743 KSR data terminal. Also shown is the relationship between cable wires and signals to the serial interface, the TMS 9902. Figure B-2 shows the cable configuration for the 733 data terminal.

NOTE

When using an RS-232-C device, disconnect jumper J11 and insert jumper J7 in the EIA position.
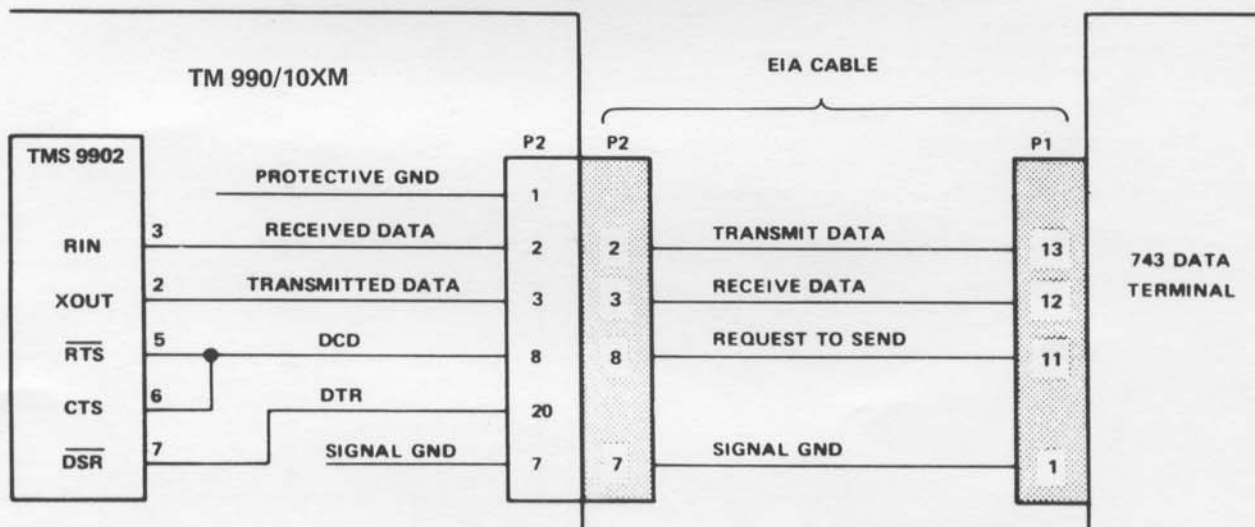
NOTE: Suggested EIA cable connectors (ITT Cannon or TRW Cinch):
P2: DB-25P
P1: DE-15S

Figure B-1. EIA RS-232-C Cabling For 743 Data Terminal
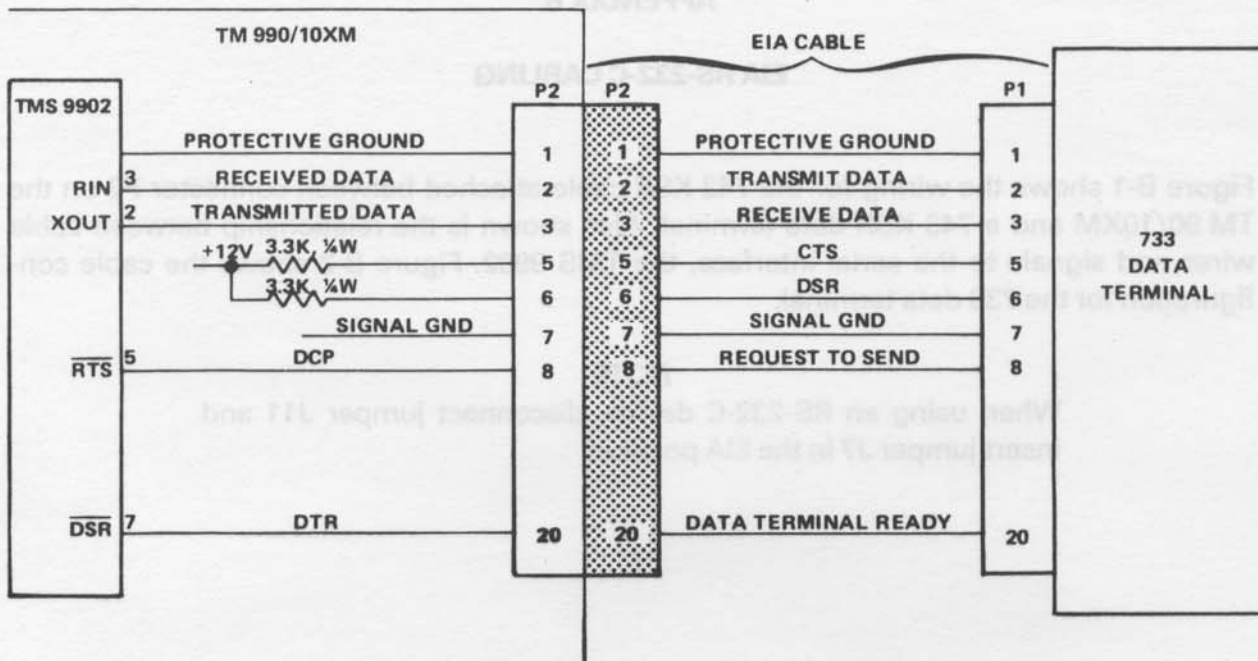
Figure B-2. EIA RS-232-C Cabling For 733 Data Terminal

APPENDIX C

ASCII CODE

TABLE C-1. *ASCII CONTROL CODES

CONTROL		BINARY CODE	HEXADECIMAL CODE
NUL	– Null	000 0000	00
SOH	– Start of heading	000 0001	01
STX	– Start of text	000 0010	02
ETX	– End of text	000 0011	03
EOT	– End of transmission	000 0100	04
ENQ	– Enquiry	000 0101	05
ACK	– Acknowledge	000 0110	06
BEL	– Bell	000 0111	07
BS	– Backspace	000 1000	08
HT	– Horizontal tabulation	000 1001	09
LF	– Line feed	000 1010	0A
VT	– Vertical tab	000 1011	0B
FF	– Form feed	000 1100	0C
CR	– Carriage return	000 1101	0D
SO	– Shift out	000 1110	0E
SI	– Shift in	000 1111	0F
DLE	– Data link escape	001 0000	10
DC1	– Device control 1	001 0001	11
DC2	– Device control 2	001 0010	12
DC3	– Device control 3	001 0011	13
DC4	– Device control 4 (stop)	001 0100	14
NAK	– Negative acknowledge	001 0101	15
SYN	– Synchronous idle	001 0110	16
ETB	– End of transmission block	001 0111	17
CAN	– Cancel	001 1000	18
EM	– End of medium	001 1001	19
SUB	– Substitute	001 1010	1A
ESC	– Escape	001 1011	1B
FS	– File separator	001 1100	1C
GS	– Group separator	001 1101	1D
RS	– Record separator	001 1110	1E
US	– Unit separator	001 1111	1F
DEL	– Delete, rubout	111 1111	7F

*American Standards Institute Publication X3.4-1968

TABLE C-2. *ASCII CHARACTER CODE

CHARACTER	BINARY CODE	HEXADECIMAL CODE	CHARACTER	BINARY CODE	HEXADECIMAL CODE
Space	010 0000	20	P	101 0000	50
!	010 0001	21	Q	101 0001	51
'' (dbl. quote)	010 0010	22	R	101 0010	52
#	010 0011	23	S	101 0011	53
$	010 0100	24	T	101 0100	54
%	010 0101	25	U	101 0101	55
&	010 0110	26	V	101 0110	56
' (sgl. quote)	010 0111	27	W	101 0111	57
(	010 1000	28	X	101 1000	58
)	010 1001	29	Y	101 1001	59
* (asterisk)	010 1010	2A	Z	101 1010	5A
+	010 1011	2B	[	101 1011	5B
, (comma)	010 1100	2C	\	101 1100	5C
− (minus)	010 1101	2D	]	101 1101	5D
. (period)	010 1110	2E	Λ	101 1110	5E
/	010 1111	2F	_ (underline)	101 1111	5F
0	011 0000	30		110 0000	60
1	011 0001	31	a	110 0001	61
2	011 0010	32	b	110 0010	62
3	011 0011	33	c	110 0011	63
4	011 0100	34	d	110 0100	64
5	011 0101	35	e	110 0101	65
6	011 0110	36	f	110 0110	66
7	011 0111	37	g	110 0111	67
8	011 1000	38	h	110 1000	68
9	011 1001	39	i	110 1001	69
:	011 1010	3A	j	110 1010	6A
;	011 1011	3B	k	110 1011	6B
<	011 1100	3C	l	110 1100	6C
=	011 1101	3D	m	110 1101	6D
>	011 1110	3E	n	110 1110	6E
?	011 1111	3F	o	110 1111	6F
@	100 0000	40	p	111 0000	70
A	100 0001	41	q	111 0001	71
B	100 0010	42	r	111 0010	72
C	100 0011	43	s	111 0011	73
D	100 0100	44	t	111 0100	74
E	100 0101	45	u	111 0101	75
F	100 0110	46	v	111 0110	76
G	100 0111	47	w	111 0111	77
H	100 1000	48	x	111 1000	78
I	100 1001	49	y	111 1001	79
J	100 1010	4A	z	111 1010	7A
K	100 1011	4B	{	111 1011	7B
L	100 1100	4C	|	111 1100	7C
M	100 1101	4D	}	111 1101	7D
N	100 1110	4E	~	111 1110	7E
O	100 1111	4F			

*American Standards Institute Publication X3.4-1968

BINARY, DECIMAL AND HEXADECIMAL NUMBERING

A-1 GENERAL

This appendix covers numbering systems to three bases (2, 10, and 16) which are used throughout this manual.

D-2 POSITIVE NUMBERS

D-2.1 Decimal (Base 10).

When a numerical quantity is viewed from right to left, the right-most digit represents the base number to the exponent 0. The next digit represents the base number to the exponent 1, the next to the exponent 2, then exponent 3, etc. For example, using the base 10 (decimal):

$$10^6 \quad 10^5 \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 10^0$$
$$X, \quad X \quad X \quad X, \quad X \quad X \quad X$$

or

$$
\begin{array}{ccccccc}
 & & 1,000,000 & & & & \\
 & & \downarrow 100,000 & & & & \\
 & & \downarrow \downarrow 10,000 & & & & \\
 & & \downarrow \downarrow \downarrow 1000 & 100 & 10 & 1 \\
X, & X & X & X, & X & X & X
\end{array}
$$

For example, 75,264 can be broken down as follows:

$$
\begin{array}{lll}
75, \; 264 & & \\
\quad 4 \times 10^0 = 4 \times 1 & = & 4 \\
\quad 6 \times 10^1 = 6 \times 10 & = & 60 \\
\quad 2 \times 10^2 = 2 \times 100 & = & 200 \\
\quad 5 \times 10^3 = 5 \times 1000 & = & 5000 \\
\quad 7 \times 10^4 = 7 \times 10,000 & = & +70,000 \\
& & \overline{75264}_{10}
\end{array}
$$

D-2.2 Binary (Base 2).

As base 10 numbers use ten digits, base 2 numbers use only 0 and 1. When viewed from right to left, they each represent the number 2 to the powers 0, 1, 2, etc., respectively as shown below:

$$2^{15} \qquad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$
$$(32{,}768) \; \bullet\bullet\bullet \; (64) \; (32) \; (16) \; (8) \; (4) \; (2) \; (1)$$
$$X \qquad\qquad X \quad X \quad X \quad X \quad X \quad X \quad X$$

For example, 11011_2 can be translated into base 10 as follows:

$$
\begin{array}{ccccc}
1 & 1 & 0 & 1 & 1
\end{array}
$$

$$1 \times 2^0 = 1 \times 1 = 1$$
$$1 \times 2^1 = 1 \times 2 = 2$$
$$0 \times 2^2 = 0 \times 4 = 0$$
$$1 \times 2^3 = 1 \times 8 = 8$$
$$1 \times 2^4 = 1 \times 16 = +16$$

$$27_{10}$$

or 11011_2 equals 27_{10}.

Binary is the language of the digital computer. For example, to place the decimal quantity 23 23_{10}) into a 16-bit memory cell, set the bits to the following:

0   15

0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1

which is $1 + 2 + 4 + 16 = 23_{10}$.

D-2.3 HEXADECIMAL (Base 16).

Whereas binary uses tow digits and decimal uses ten digits, hexadecimal uses 16 (0 to 9, A, B, C, D, E, and F).

The letters A through F are used to represent the decimal nunbers 10 through 15 as shown on the following page.

N_{10}	N_{16}	N_{10}	N_{16}
0	0	8	8
1	1	9	9
2	2	10	A
3	3	11	B
4	4	12	C
5	5	13	D
6	6	14	E
7	7	15	F

When viewed from right to left, each digit in a hexadecimal number is a multiplier of 16 to the powers 0, 1, 2, 3, etc., as shown below:

$$16^3 \qquad 16^2 \qquad 16^1 \qquad 16^0$$
$$(4096) \quad (256) \quad (16) \quad (1)$$
$$X \qquad X \qquad X \qquad X$$

For example, $7\,B\,A\,5_{16}$ can be translated into base 10 as follows:

7　　B　　A　　5

$$10 \times 16^0 = 5 \times 1 \qquad = \qquad 5$$
$$10 \times 16^1 = 10 \times 16 \qquad = \qquad 160$$
$$11 \times 16^2 = 11 \times 256 = \qquad 2,816$$
$$7 - 16^3 = 7 \times 4096 \qquad = \quad \underline{28,672}$$
$$31,653_{10}$$

or $7\,B\,A\,5_{16}$ equals $31,653_{10}$.

Because it would be awkward to write out 16-digit binary numbers to show the contents of a 16-bit memory word, hexadecimal is used instead. Thus

$$003E_{16} \text{ or } > 003E \ (> \text{indicates hexadecimal})$$

is used instead of

$$0000\ 0000\ 0011\ 1110_2$$

to represent 62_{10} as computed below:

BASE 2

1　1　1　1　1　0_2

$$1 \times 2^1 \quad = \quad 0$$
$$1 \times 2^1 \quad = \quad 2$$
$$1 \times 2^2 \quad = \quad 4$$
$$1 \times 2^3 \quad = \quad 8$$
$$1 \times 2^4 \quad = \quad 16$$
$$1 \times 2^5 \quad = \quad \underline{32}$$
$$62_{10}$$

BASE 10

$$6\;2_{10}$$

$$2 \times 10^0 = 2$$
$$6 \times 10^1 = 60$$
$$\overline{62_{10}}$$

BASE 16

$$3\;E_{16}$$

$$14 \times 16^0 = 14$$
$$3 \times 16^1 = 48$$
$$\overline{62_{10}}$$

Note that the 16 binary bits into four-bit parts facilitates recognition and translation into hexadecimal.

0000	0000	0011	$1110_2/$		C	7	B	F_{16}
↓	↓	↓	↓	or	↓	↓	↓	↓
0	0	3	E_{16}		1100	0111	1011	1111_2

Table D-1 is a conversion chart for converting decimal to hexadecimal and vice versa. Table D-2 shows binary, decimal and hexadecimal equivalents for numbers 0 to 15. Note that Table D-1 is divided into four parts, each part representing four of the 16-bits of a memory cell or word (bits) 0 to 15 with bit 0 being the most significant bit (MSB) and bit 15 being the least significant bit (LSB). Note that the MSB is on the left and represents the highest power of 2 and the LSB on the right represents the 0 power of 2 ($2^0 = 1$). As explained later, the MSB can also be used to signify number polarity (+ or −).

NOTE

To convert a binary number to decimal or hexadecimal, convert the *positive* binary value as described in Section D-4.

TABLE D-1. HEXADECIMAL/DECIMAL CONVERSION CHART

MSB					LSB

	16^3		16^2		16^1		16^0	
BITS	0 1 2 3		4 5 6 7		8 7 8 11		12 13 14 15	
	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	32 768	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	45 056	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

To convert a number from hexadecimal, add the decimal equivalents for each hexadecimal digit. For example, $7A82_{16}$ would equal in decimal $28,672 + 2,560 + 128 + 2$. to convert hexadecimal to decimal, find the nearest decimal number in the above table less than or equal to the number being converted. Set down the hexadecimal equivalent then subtract this number from the nearest decimal number. Using the remainder(s), repeat this process. For example:

$$31,362_{10} = 7000_{16} + 2690_{10} \qquad 7000$$
$$2,690_{10} = A00_{16} + 130_{10} \qquad A00$$
$$130_{10} = 80_{16} + 2_{10} \qquad 80$$
$$2_{10} = 2_{16} \qquad 2$$
$$\overline{7A82_{16}}$$

TABLE D-2. BINARY, DECIMAL, AND HEXADECIMAL EQUIVALENTS

BINARY (N₂)	DECIMAL (N₁₀)	HEXADECIMAL (N₁₆)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F
10000	16	10
10001	17	11
10010	18	12
10011	19	13
10100	20	14
10101	21	15
10110	22	16
10111	23	17
11000	24	18
11001	25	19
11010	26	1A
11011	27	1B
11100	28	1C
11101	29	1D
11110	30	1E
11111	31	1F
100000	32	20

D-3   ADDING AND SUBTRACTING BINARY

Adding and subtracting inbinary uses the same conventions for decimal; carrying over in addition and borrowing in subtraction.

Basically,

```
  0          1                                                    10
+ 1        + 1                                                   - 1
---        ---                                                   ---
  1         10     (the carry, 1, is carried to the left)         01     (1 is borrowed from
                                                                          top left)
```

```
 1 }
 1 }  = 0 + carry 1                                               11
                                                                  1
+ 1    = 0 (from above) + 1 = 1                                 + 1
----                                                            ----
 11                                                             101
    \___carry                        carry 1 + 1 = 10___/
```

```
 1 }
 1 }  = 0 + 1 carry          1000 )                 ( 0110
                            - 1  }  Borrow the 1    { - 1
 1 }                        -----                    -----
 1 }  = 0 + 1 carry          0111 )                 ( 0111
+ 1 }
----
100
   \_0 + 0 = 0
   \_carry 1 + carry 1
```

D-4   POSITIVE/NEGATIVE CONVERSION (Binary)..

To compute the negative equivalent of a positive binary or hexadecimal number, or interpret a binary or hexadecimal negative number (determine its positive equivalent) use the two's complement of the binary number.

NOTE

To convert a binary number to decimal, convert the *positive* binary value (*not* the negative binary value) and add the sign.

Two's complementing a binary number includes two simple steps:

a. Obtain one's complement of the number (1's become 0's, 0's becomes 1's) (invert bits).

b. Add 1 to the one's complement.

For example, with the MSB (left-most bit) being a sign bit:

$\underline{010}$ ($+2_2$)	$\underline{111}$ (-1_2)	$\underline{110}$ (-2_2)	$\underline{101}$ (-3_2)
101 Invert	000 Invert	001 Invert	010 Invert
$\underline{+\ 1}$ Add 1	$\underline{+\ 1}$ Add 1	$\underline{+\ 1}$ Add 1	$\underline{+\ 1}$
110 (-2_2)	001 ($+1_2$)	010 ($+2_2$)	011 ($+3_2$)

This can be expanded to 16-bit positive numbers:

($=39F6_{16}$)	$\underline{0011}$	$\underline{1001}$	$\underline{1111}$	$\underline{0110}$	($39F6_{16}$ =	$+14,838_{10}$)
	1100	0110	0000	1001	Invert	
				$\underline{+1}$	Add 1	
($=C60A_{16}$)	1100	0110	0000	1010	($C60A_{16}$ =	$-14,838_{10}$) Two's Complement

SIGN BIT(−)

And to 16-bit negative numbers:

($=C60A_{16}$)	1100	0110	0000	1010	($C60A_{16}$ =	$-14,838_{10}$)
	0011	1001	1111	0101	Invert	
				$\underline{+1}$	Add 1	
($=39F6_{16}$)	0011	1001	1111	0110	($39F6_{16}$ =	$+14,838_{10}$) Two's Complement

SIGN BIT(+)

E-1 SYMBOLIC ASSEMBLER ERROR CODES

Error No.	Explanation
1	Invalid Symbol: Symbols must be alphanumerical with the first character alphabetical. Use correct format.
2	Multiply Defined Symbol: Symbol is used to define location of a previous source line. Use other symbol.
3	Symbol Table Overflow: Symbol table cannot accept any further entries. Restrict quantity of symbols to amount shown in Table 4-1.
4	Mnemonic Size Too Large. Op-code mnemonic is more than four characters.
5	Undefined Mnemonic: Op-code mnemonic is not one of the valid TM 990 mnemonics not a defined XOP mnemonic. Valid TM 990 mnemonics are listed in Appendix H.
6	Illegal Register Number: Register number is greater than 15.
7	CRU Instruction Displacement: Displacement of TB, SBZ, or SBO) instruction is greater than 256 (plus or minus).
8	Jump Instruction Displacement: Displacement of jump instruction is greater than 256 (plus or minus).
9	Invalid Shift Count: Shift count must be from zero to 15.
10	Non-increasing Location Counter: Memory address in AORG assembler directive is a smaller value than current value of location counter. This will occur when second AORG value is less than the value in a prior AORG directive. Location Counter contents must be in ascending order.
11	Byte Value Too Large: Operand of BYTE assembler directive is a value larger than 256.

12   No Start of Text: Character string following a TEXT assembler directive did not start with a single quote. Such character strings are delimited with single quotes. A single quote within a TEXT directive is specified by two single quotes.

13   IDT Length Error: Character string of IDT assembler directive (at beginning of program) has character string of more than eight characters.

14   Invalid IDT: Character string of IDT assembler directive did not begin with a single quote.

15   Illegal Text Statement: Text statement contained character that cannot be interpreted.

16   Illegal XOP Number: XOP number above 15 specified. XOP numbers are from 0 to 15.

17   Undefined Symbol: Symbol was used in an instruction which had not been defined in the symbol field of the source statement.

18   Input I/O Error: Error in reading source from cassette; probably a checksum error or improper casette connection.

19   No END Directive: Must have an END directive as last statement in program (see paragraph F.2.7 in Appendix F). Use Text Editor to input this statement.

20   Illegal Mathematical Expression. Delete expression (explained in paragraph 4.7).

E-2   TEXT EDITOR ERROR CODES

Error No.	Explanation
21	Commands: G, K, Q
	Meaning: Error in transmission of source statements from cassette to memory buffer, could be a checksum error or error in data communications format.
	Recovery Procedure: The program segment with the error is lost and will not be written to cassette with a Keep or Quit command. A Quit command will write the remaining program segments to the output cassette; then, the truncated program can be read in again and the lost source records re-entered. Or, after the Quit command, read in the original source cassette (with error) to see if the error reoccurs (any editing in the first session will not be on this cassette). Or, the program segment can be considered lost and the user can continue editing the remaining source segments, finishing the session with a Quit command, without the program segment in error.

22 Command: G
Meaning: There is a source line in the memory buffer with a line number larger than the first line number of a segment brought in from cassette. Incoming source lines should be ascendingly greater than line numbers in the buffer.

Recovery Procedure: Further input of a program segment is prohibited (i.e., Get command is prohibited). Commands other than Get can be executed; however, editing out of the line(s) causing this error will not permit Get command to execute. A Quit command causes a return to the monitor without reading in remaining program segments from input cassette. User can re-edit original source on the input cassette or edit the truncated source on the output cassette.

23 Command: G
Meaning: Get command cannot be performed because end-of-file (EOF) has been detected; thus complete program file has been read.

Recovery Procedure: Execute Quit command to write memory buffer contents to cassette.

24 Command: G
Meaning: Insufficient space in memory buffer to receive new source records.

Recovery Procedure: Execute Keep command to write buffer contents to cassette. Re-execute Get command.

25 Command: All
Meaning: Text Editor command input is not valid.

Recovery Procedure: Insert valid Text Editor command.

26 Command: K, R
Meaning: Resequencing of line numbers requested after Keep command executed. Entire program through EOF marker must have the same line sequence numbering.

Recovery Procedures:
(1) Continue using present sequencing value
(2) or execute Quit command, reenter Text Editor, and specify new resequencing value before first Keep command. Last Resequence command will apply.

27 Command: P
Meaning: In Print command, first (beginning) line number to be printed was larger than second (ending) line number.

Recovery Procedure: Reenter command with first line number smaller than second line number.

28 Command: P, I
Meaning: Line number to be printed is too small; a line number of higher value has been sent to the output cassette.

Recovery Procedure: Reenter command; use a source line number with a value higher than the highest source line number in the output cassette.

29 Commands: Insertion or change commands
Meaning: Command issued to insert or change a line having a higher value than the highest-numbered line in the memory buffer.

Recovery Procedure: If line to be inserted or changed has a higher number than the highest line number memory buffer, use Get command to read in the program segment containing line numbers in the range desired.

30 Command: Insertion Command
Meaning: Not enough memory buffer space for line to be inserted or line to replace existing line.

Recovery Procedure: Decrease memory buffer contents using a Keep command or the line deletion command. Then attempt the line insertion again. If insertion command also causes a line to be deleted (replace existing line), then line to be replaced was deleted but new line was not inserted in its place.

31 Command: R
Meaning: Starting line number in Resequence command was a value greater than 9000 (decimal).

Recovery Procedure: Re-execute Resequence Command with a starting-line less than 9000.

32 Command: R
Meaning: Resequencing command executed and tried to generate a line number greater than 9999 during a Keep or a Quit function. Maximum line number allowed is 9999.

Recovery Procedure: When the line number reaches 9999, an end-of-file is written to the cassette, and data following line 9999 is lost. Re-edit the source as required and begin resequencing lines with a lower beginning number. The resequencing command increments line numbers from 1 to 8999 by ten, and increments line numbers from 9000 to 9999 by one.

E-3   RELOCATING LOADER ERROR CODES

Error No.	Explanation

51   Meaning: An invalid load tag was found. The loader interprets several load tags including 9 (absolute load address), A (relocatable load address), B (absolute data), and C (relocatable data). Load tags used by the 990 family of computers are explained in Appendix G.

Recovery: Rewind cassette and re-execute the load operation. If the error re-occurs, reassemble the program to obtain a new object on cassette.

52   Meaning: Checksum error occurred. When the object code is formatted into object records by the assembler, the last load tag field of each record is a checksum value which is the two's complement of the sum of all ASCII character values representing the object code; this includes all characters beginning with the first tag character in the object record up to and including the "7" tag of the checksum field. The loader makes a similar computation when loading the object and compares the results to the checksum value. If the comparison is a match, the loaded data is considered valid.

Recovery: Rewind cassette and re-execute the load operation. If the error reoccurs, reassemble the program to obtain a new object on cassette.

E-4   BAUD RATE CHANGE ERROR CODE

Error No.	Explanation

91   Meaning: Baud rate specified is not one of the following: 110, 300, 1200, 2400, 4800, 9600, or 19200 Hertz.

Recovery: Specify one of the above baud rates.

F-1   GENERAL

This appendix defines the following eleven assembler directives recognized by the Symbolic Assembler (described in Section 4). Note that at least two of these must be used in every program: the IDT and END directives. These directives and corresponding paragraph number are:

● AORG	Absolute origin of statement (absolute location)	F.2.1
● BSS	Block of memory starting with symbol	F.2.2
● BYTE	Eight-bit immediate value	F.2.3
● DATA	Sixteen-bit immediate value	F.2.4
● DREG	Define registers as being preceded by an R	F.2.5
● DXOP	Define XOP opcode mnemonic	F.2.6
● END	End of source code	F.2.7
● EQU	Label equated to symbol or value	F.2.8
● EVEN	Location counter to even-numbered address	F.2.9
● IDT	Identifying symbol for program	F.2.10
● TEXT	Code character string in ASCII code	F.2.11

F-2   DIRECTIVE FORMATS

Syntax used in this subsection (see paragraph 1.7):

     < > Items enclosed in these must be supplied by the user

     [ ] Items enclosed in these are optional

     Λ Indicates at least one space

     > Indicates hexadecimal quantity follows

F-2.1   AORG DIRECTIVE

Format:

     [label] Λ <AORG> Λ <location> Λ [comment]

The AORG directive places a value in the location counter and defines the source statement code as beginning at that location. The location value must be in decimal or

hexadecimal. By default, the location counter for the assembler begins at 0000_{16} and is incremented by two for each word occupied by the instruction. The first AORG in a program can be preceded only by an IDT directive. If successive AORG's are used, the AORG value must be higher than the current contents of the location counter. In the case of more than one AORG, the length of the object module for the Relocating Loader is computed as the difference between the lowest AORG value and the highest address listed in the assembler listing plus two. The loading address is that of the first AORG. When a label is used with the AORG directive, it is assigned the value that the directive places in the location counter. Comment field is optional. Example:

AORG >FC00   Begin assembling source code at location
                          counter value of >FC00

F.2.2 BSS DIRECTIVE

Format:

[label] Λ <BSS> Λ <number of bytes> Λ [comment]

The BSS (block with starting symbol) directive advances the location counter (which the assembler uses to count the bytes of machine code) a quantity of bytes as specified in the directive. In essence, it "reserves" a block of bytes starting at the next location counter value; this block will be void of object code when loaded later by the Relocating Loader. Code assembler after this directive will be loaded following this block when loaded by the loader. An optional label (in the label field) can be specified to identify the first location in the block. The byte count must be in decimal or hexadecimal.

F.2.3 BYTE DIRECTIVE

Format:

[label] Λ <BYTE> Λ <1-8 bit value,..., 1-8 bit value> Λ [comment]

This directive places one or more decimal or hexadecimal (8 bits maximum) values in successive bytes. If the value specified is larger than 8 bits (one byte), an error message is printed and the right-most eight bits are assembled into the byte. When a label is used, it is assigned the location of the first byte in the directive. If more than one byte is specified, successive bytes will be separated by commas. Successive entries to this directive will be placed in consecutive byte locations. Specified bytes will begin at even or odd addresses, depending upon preceding code.

Example:

BYTE 125,240,>FF,0,33

Successive bytes to be assembled with hexadecimal values of each

F.2.4 DATA DIRECTIVE

Format:

[label] Λ <DATA> Λ <1-16 bit value ,..., 1-16 bit value> Λ [comment]

This directive is similar to the BYTE directive (paragraph F.2.3) except that it places 16 bit values into (successive) memory locations. Data is placed in even address locations.

Example:

DATA <FFFF,1764,>BB,0,444

Assemble as 00BB,0000

F.2.5 DXOP DIRECTIVE

Format:

[label] Λ <DXOP> Λ <symbol,XOP no.> Λ [comment]

This directive allows the user to specify a one- to four-character mnemonic in place of the XOP mnemonic and the XOP number used in an extended operation instruction. This directive permits the use of a mnemonic to define the use of the particular XOP, useful when this instruction is used repeatedly. The DXOP opcode and space are followed by (1) the mnemonic to be used as the operand of the XOP instruction and (2) the XOP number, both separated by a comma. The label field is optional, the label will be assigned the current location counter value.

NOTE

The DXOP directive must be used in the program prior to using the abbreviated format it defines.

For example:

    XOP   @>FC00,14

can be simplified by specifying

    DXOP   ECHO,14

then using

    ECHO   @>FC00

F.2.6 END DIRECTIVE

Format:

    [label] Λ <END> Λ [entry point] Λ [comment]

This directive is a mandatory for each program. It designates to the assembler that this is the final input from the source program. In essence, it informs the assembler that all source lines have been read in and that the next phase of assembling should begin. This is the last statement in the program, and any statements following it will be ignored. When the optional label is used, it is assigned the current value in the location counter. The optional operand field contains a symbol (absolute or relocatable) specifying the entry point of the program. When the entry-point operand is used, the entry point is specified in the object code so that the Program Counter will be set to this value by the loader immediately after loading. The program can then be executed by the EX command of the Program Debugger.

Example:

    END   START

                        location labelled START is entry point for program

F.2.7 EQU DIRECTION

Format:

<label> Λ <EQU> Λ <expression> Λ [comment]

This directive assigns a value to a label for use during assembly. The operand field can contain a symbol or expression which has been previously defined (e.g., any constant such as a numerical value or a previously used symbol or expression of these). This directive allows the user to substitute easily remembered mnemonics in program source lines. The optional label will be assigned the current value in the location counter. Mathmatical expressions (e.g., LABL + 5) cannot be used with this directive.

Examples:

(1) SUM   EQU   1

allows using SUM for register 1 such as

     MOV   @>FC00,SUM   MOVE QTY TO R1

instead of

     MOV   @>FC00,1   MOVE QTY TO R1

(2) INT   EQU   9681

allows this constant value to be used in subsequent source lines

     LI   R1,INT   PLACE CONSTANT IN R1

instead of remembering the constant value.

(3) If INT has been previously defined as above, the following

     MOV   @INT+4,@INT1

will result in moving the value located four bytes beyond location INT into location INT1.

F.2.8 EVEN DIRECTIVE

Format:

[label] Λ <EVEN> Λ [comment]

This directive is used to set the location counter at an even numbered value. If it is at an even numbered value, no action is taken. If the location counter is at an odd value, the quantity of one is added to it. This directive ensures that code will start at an even address.

F.2.9 IDT DIRECTIVE

Format:

[label] Λ <IDT> Λ <'character string'> Λ [comment]

This is a <u>mandatory</u> directive and should be the first statement in a program, preceding any source statements that will result in object code. The mnemonic is followed by a space and a character string of one to eight alphanumeric characters in single quotes. The optional label will be assigned the value of the location counter.

F.2.10 TEXT DIRECTIVE

Format:

[label] Λ <TEXT> Λ [-] <'character string'> Λ [comment]

This directive, like the BYTE and DATA directives, is used to generate absolute data for program use. BYTE and DATA statement operands are interpreted as numerical values. The TEXT statement operand contains alphanumeric characters which are to be interpreted into ASCII code. Inputs in the operand field are enclosed in single quotes. If it is desired to have the last character in the string negated (left-hand sign bit set to one), place a minus sign before the character string. This latter feature can be used to identify the final character in the string. The optional label field will be assigned the value in the location counter; this value will identify the location of the first character in the string.

Examples:

(1) CMNT   TEXT   'LOAD TAPE, HIT CR'

(2) CMNT   TEXT   -'LOAD TAPE, HIT CR.'

Minus sign preceding text causes final character to be negated (two's complement) (the period, >2E, becomes >D2).

990 OBJECT CODE FORMAT

G.1 GENERAL

In order to correctly load a program into memory using a loader, the program in hexadecimal machine code must be in a particular format called object format. Such a format is required by the Relocating Loader (Section 5 explains loader execution). This object format has a tag character for each 16-bit word of coding which flags the loader to perform one of several operations. These operations include:

- Load the code at a user-specified absolute address and resolve relative addresses. (Most assemblers assemble a program as if it was loaded at memory address 0000_{16}; thus relative addresses have to be resolved.)

- Load entire program at a specific address.

- Set the program counter to the entry address after loading.

- Check for checksum errors that would indicate a data error in an object record.

NOTE

The TM 990/302 Symbolic Assembler does not provide relocatable object code; thus the relocating feature of the first operation example above is not required. The Symbolic Assembler utilizes the following object tags only (further described in Table G-1): 0, 1, 7, 9, B, and F.

G.2 STANDARD 990 OBJECT CODE

Standard 990 object code consists of a string of hexadecimal digits, each representing four bits, as shown in Figure G-1.

TAG CHARACTERS

```
00000SAMPROG 90040C0000A0020BC06DB000290042C0020A0024BC81BC002A7F21AF
A0028B0241B0000BCB41B0002B0380A00CAC0052C00A2B02E0C0032B0200B0F0F7F1DEF
A00D6BC0A0C00CAB04C3BC160C00CCBC1A0C00D0BC072B0281B3A00A00ECB02217F151F
A00EEB0900B06C1A00EAB1102A00F2B0543B11F8B2C20C0032BC101B0B44BE0447F18EF
A0100BDD66B0003B0282C00A2B11EDB03407F832F
200CE0010C        7FCABF
```

CHECKSUM FIELD

LENGTH OF RELOCATABLE CODE

RELOCATABLE ENTRY ADDRESS (BEGINNING OF EXECUTABLE CODE)

END OF OBJECT CODE MARKER

Figure G-1. Object Code Example

The object record consists of a number of tag characters, each followed by one or two fields as defined in Table G-1. The first character of a record is the first tag character, which tells the loader which field or pair of fields follows the tag. The next tag character follows the end of the field or pair of fields associated with the preceding tag character. When the assembler has no more data for the record, the assembler writes the tag character 7 followed by the checksum field, and the tag character F, which requires no fields. The assembler then fills the rest of the record with blanks, and begins a new record with the appropriate tag character.

Tag character 0 is followed by two fields. The first field contains the number of bytes of relocatable code, the second field contains the program identifier assigned to the program by an IDT assembler directive. When no IDT directive is entered, the field contains blanks. The loader uses the program identifier to identify the program, and the number of bytes of relocatable code to determine the load bias for the next module or program. The PX9ASM assembler is unable to determine the value for the first field until the entire module has been assembled, so PX9ASM places a tag character 0 followed by a zero field and the program identifier at the beginning of the object code file. At the end of the file, PX9ASM places another tag character zero followed by the number of bytes of relocatable code and eight blanks.

Tag characters 1 and 2 are used with entry addresses. Tag character 1 is used when the entry address is absolute. Tag character 2 is used when the entry address is relocatable. The hexadecimal field contains the entry address. One of these tags may appear at the end of the object code file. The associated field is used by the loader to determine the entry point at which execution starts when the loading is complete.

Tag characters 3 and 4 are used for external references. Tag character 3 is used when the last appearance of the symbol in the second field is in relocatable code. Tag character 4 is used when the last appearance of the symbol is absolute code. The hexadecimal field contains the location of the last appearance. The symbol in the second field is the external reference. Both fields are used by the linking loader to provide the desired linking to the external reference.

For each external reference in a program, there is a tag character in the object code, with a location, or an absolute zero, and the symbol that is referenced. When the object code field contains absolute zero, no location in the program requires the address that corresponds to the reference (an IDT character string, for example). Otherwise, the address corresponding to the reference will be placed in the location specified in the object code by the linking loader. The location specified in the object code similarly contains absolute zero or another location. When it contains absolute zero, no further linking is required. When it contains a location, the address corresponding to the reference will be placed in that address by the linking loader. The location of each appearance of a reference in a program contains either an absolute zero or another location into which the linking loader will place the referenced address.

TABLE G-1. OBJECT OUTPUT TAGS SUPPLIED BY ASSEMBLERS

TAG CHARACTER	HEXADECIMAL FIELD (FOUR CHARACTERS)	SECOND FIELD	MEANING
0	Length of all relocatable code	8-character program identifier	Program start
1	Entry address	None	Absolute entry address
2	Entry address	None	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	None	Checksum
8	Ignore checksum	None	Do not checksum for error
9	Load address	None	Absolute load address
A	Load address	None	Relocatable load address
B	Data	None	Absolute data
C	Data	None	Relocatable data
D	Load bias value*	None	Load point specifier
F	None	None	End-of-record
G	Location	6-character symbol	Relocatable symbol definition
H	Location	6-character symbol	Absolute symbol definition

*Not supplied by assembler

Tag characters 5 and 6 are used for external definitions. Tag character 5 is used when the location is relocatable. Tag character 6 is used when the location is absolute. Both fields are used by the linking loader to provide the desired linking to the external definition. The second field contains the symbol of the external definition.

Tag character 7 precedes the checksum, which is an error detection word. The checksum is formed as the record is being written. It is the 2's complement of the sum of the 8-bit ASCII values of the characters of the record from the first tag of the record through the checksum tag 7. If the tag character 7 is replaced by an 8, the checksum will be ignored. The 8 tag can be used when object code is changed in editing and it is desired to ignore checksum.

Tag characters 9 and A are used with load addresses for data that follows. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is relocatable. The hexadecimal field contains the address at which the following data word is to be loaded. A load address is required for a data word that is to be placed in memory at some address other than the next address. The load address is used by the loader.

Tag characters B and C are used with data words. Tag character B is used when the data is absolute; an instruction word or a word that contains text characters or absolute constants, for example. Tag character C is used for a word that contains a relocatable address. The hexadecimal field contains the data word. The loader places the word in memory location specified in the preceding load address field, or in the memory location that follows the preceding data word.

To have object code loaded at a specific memory address, precede the object program with the D tag followed by the desired memory address (e.g., DFD00).

Tag character F indicates the end of record. It may be followed by blanks.

Tag characters G and H are used when the symbol table option is specified with other 990 assemblers. Tag character G is used when the location or value of the symbol is relocatable, and tag character H is used when the location or value of the symbol is absolute. The first field contains the location or value of the symbol, and the second field contains the symbol to which the location is assigned.

The last record of an object code file has a colon (:) in the first character position of the record, followed by blanks. This record is referred to as an end-of-module separator record.

Figure G-2 is an example of an assembler source listing and corresponding object code. A comparison of the object tag characters and fields with the machine code in the source listing will show how object code is constructed for use by the loader.

SOURCE STATEMENT NO.

LOCATION COUNTER (ADDRESS RELATIVE TO FIRST OBJECT BYTE)

MACHINE CODE

SAMPLE   SDSMAC 945278 **

PAGE 0001

```
0001                          IDT  'SAMPLE'
 ʔ02 0000 0006'               DATA WSPACE
  03 0002 008A'               DATA START
0004 0004 0000               DATA 0
0005 0006              WSPACE BSS  32
0006 0026              TABLE  BSS  100
0007 008A              START
0008 008A 04CC                CLR  12
0009 008C 04C0                CLR  0
0010 008E 0202                LI   2, TABLE
     0090 0026'
0011 0092 C800                MOV  0, @TABLE+2
     0094 0028'
0012 0096 1001                JMP  $+4
0013 0098              LOOP
0014 0098 0204                LI   4, >1234
     009A 1234
0015 009C 0244                ANDI 4, >FEED
     009E FEED
0016 00A0 DC84                MOVB 4, *2+
0017 00A2 0205                LI   5, >5555
     00A4 5555
0018 00A6 C805                MOV  5, @TABLE
     00A8 0026'
0019                          END
NO ERRORS
```

000AASAMPLE   A0000C0006C008AB0000A008AB04CCB04C0Bʔʔ0ꞌC00꞉6BC8007F200F   000
C0028B1001B0204B1234B0244BFEEDBDC84B0205B5555BC805C00267F3C1F   000
:   SAMPLE   00/00/00   08:14:23   SDSMAC 945278 ♦♦

Figure G-2. Source Code And Corresponding Object Code

INSTRUCTION SET

INSTRUCTION SET, ALPHABETICAL INDEX

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION
A	A000	1	0-4	X	Add (word)
AB	B000	1	0-5	X	Add (byte)
ABS	0740	6	0-2	X	Absolute Value
AI	0220	8	0-4	X	Add Immediate
ANDI	0240	8	0-2	X	AND Immediate
B	0440	6	—		Branch
BL	0680	6	—		Branch and Link (R11)
BLWP	0400	6	—		Branch; New Workspace Pointer
C	8000	1	0-2		Compare (word)
CB	9000	1	0-2,5		Compare (byte)
CI	0280	8	0-2		Compare Immediate
CKOF	03C0	7	—		User Defined
CKON	03A0	7	—		User Defined
CLR	04C0	6	—		Clear Operand
COC	2000	3	2		Compare Ones Corresponding
CZC	2400	3	2		Compare Zeroes Corresponding
DCA	2C00	9	0-5,7		Correct BCD Addition (9940)
DCS	2C40	9	0-5,7		Correct BCD Subtraction (9940)
DEC	0600	6	0-4	X	Decrement (by one)
DECT	0640	6	0-4	X	Decrement (by two)
DIV	3C00	9	4		Divide
IDLE	0340	7	—		Computer Idle
INC	0580	6	0-4	X	Increment (by one)
INCT	05C0	6	0-4	X	Increment (by two)
INV	0540	6	0-2	X	Invert (One's Complement)
JEQ	1300	2	—		Jump Equal (ST2=1)
JGT	1500	2	—		Jump Greater Than (ST1=1), Arithmetic
JH	1B00	2	—		Jump High (ST0=1 and ST2=0), Logical
JHE	1400	2	—		Jump High or Equal (ST0 or ST2=1), Logical
JL	1A00	2	—		Jump Low (ST0 and ST2=0), Logical
JLE	1200	2	—		Jump Low or Equal (ST0=0 or ST2=1), Logical
JLT	1100	2	—		Jump Less Than (ST1 and ST2=0), Arithmetic
JMP	1000	2	—		Jump Unconditional
JNC	1700	2	—		Jump No Carry (ST3=0)
JNE	1600	2	—		Jump Not Equal (ST2=0)
JNO	1900	2	—		Jump No Overflow (ST4=0)
JOC	1800	2	—		Jump On Carry (ST3=1)

ASSEMBLY LANGUAGE MNEMONIC	MACHINE LANGUAGE OP CODE	FORMAT	STATUS REG. BITS AFFECTED	RESULT COMPARED TO ZERO	INSTRUCTION
JOP	1C00	2	—		Jump Odd Parity (ST5·1)
LDCR	3000	4	0-2,5	X	Load CRU
LI	0200	8	—	X	Load Immediate
LIIM	2C80	9	14,15		Load Interrupt Mask Immediate (9940)
LIMI	0300	8	12-15		Load Interrupt Mask Immediate
LREX	03E0	7	12-15		Load and Execute
LWPI	02E0	8	—		Load Immediate to Workspace Pointer
MOV	C000	1	0-2	X	Move (word)
MOVB	D000	1	0-2,5	X	Move (byte)
MPY	3800	9	—		Multiply
NEG	0500	6	0-2	X	Negate (Two's Complement)
ORI	0260	8	0-2	X	OR Immediate
RSET	0360	7	12-15		Reset AU
RTWP	0380	7	0-15		Return from Context Switch
S	6000	1	0-4	X	Subtract (word)
SB	7000	1	0-5	X	Subtract (byte)
SBO	1D00	2	—		Set CRU Bit to One
SBZ	1E00	2	—		Set CRU Bit to Zero
SETO	0700	6	—		Set Ones
SLA	0A00	5	0-4	X	Shift Left Arithmetic
SOC	E000	1	0-2	X	Set Ones Corresponding (word)
SOCB	F000	1	0-2,5	X	Set Ones Corresponding (byte)
SRA	0800	5	0-3	X	Shift Right (sign extended)
SRC	0B00	5	0-3	X	Shift Right Circular
SRL	0900	5	0-3	X	Shift Right Logical
STCR	3400	4	0-2,5	X	Store From CRU
STST	02C0	8	—		Store Status Register
STWP	02A0	8	—		Store Workspace Pointer
SWPB	06C0	6	—		Swap Bytes
SZC	4000	1	0-2	X	Set Zeroes Corresponding (word)
SZCB	5000	1	0-2,5	X	Set Zeroes Corresponding (byte)
TB	1F00	2	2		Test CRU Bit
X	0480	6	—		Execute
XOP	2C00	9	6		Extended Operation
XOR	2800	3	0-2	X	Exclusive OR

MACHINE LANGUAGE OP CODE (HEXADECIMAL)	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
0200	LI	Load Immediate	8	0-2
0220	AI	Add Immediate	8	0-4
0240	ANDI	And Immediate	8	0-2
0260	ORI	Or Immediate	8	0-2
0280	CI	Compare Immediate	8	0-2
02A0	STWP	Store WP	8	—
02C0	STST	Store ST	8	—
02E0	LWPI	Load WP Immediate	8	—
0300	LIMI	Load Int. Mask	8	12-15
0340	IDLE	Idle	7	—
0360	RSET	Reset AU	7	12-15
0380	RTWP	Return from Context Sw.	7	0-15
03A0	CKON	User Defined	7	—
03C0	CKOF	User Defined	7	—
03E0	LREX	Load & Execute	7	—
0400	BLWP	Branch; New WP	6	—
0440	B	Branch	6	—
0480	X	Execute	6	—
04C0	CLR	Clear to Zeroes	6	—
0500	NEG	Negate to Ones	6	0-2
0540	INV	Invert	6	0-2
0580	INC	Increment by 1	6	0-4
05C0	INCT	Increment by 2	6	0-4
0600	DEC	Decrement by 1	6	0-4
0640	DECT	Decrement by 2	6	0-4
0680	BL	Branch and Link	6	—
06C0	SWPB	Swap Bytes	6	—
0700	SETO	Set to Ones	6	—
0740	ABS	Absolute Value	6	0-2
0800	SRA	Shift Right Arithmetic	5	0-3
0900	SRL	Shift Right Logical	5	0-3
0A00	SLA	Shift Left Arithmetic	5	0-4
0B00	SRC	Shift Right Circular	5	0-3
1000	JMP	Unconditional Jump	2	—
1100	JLT	Jump on Less Than	2	—
1200	JLE	Jump on Less Than or Equal	2	—
1300	JEQ	Jump on Equal	2	—
1400	JHE	Jump on High or Equal	2	—
1500	JGT	Jump on Greater Than	2	—
1600	JNE	Jump on Not Equal	2	—
1700	JNC	Jump on No Carry	2	—
1800	JOC	Jump on Carry	2	—
1900	JNO	Jump on No Overflow	2	—
1A00	JL	Jump on Low	2	—
1B00	JH	Jump on High	2	—
1C00	JOP	Jump on Odd Parity	2	—
1D00	SBO	Set CRU Bits to Ones	2	—
1E00	SBZ	Set CRU Bits to Zeroes	2	—
1F00	TB	Test CRU Bit	2	2
2000	COC	Compare Ones Corresponding	3	2

MACHINE LANGUAGE OP CODE (HEXADECIMAL	ASSEMBLY LANGUAGE MNEMONIC	INSTRUCTION	FORMAT	STATUS BITS AFFECTED
2400	CZC	Compare Zeroes Corresponding	3	2
2800	XOR	Exclusive Or	3	0-2
2C00	XOP	Extended Operation	9	6
2C00	DCA	Correct BCD Addition (9940)	9	0-5,7
2C40	DCS	Correct BCD Subtraction (9940)	9	0-5,7
2C80	LIIM	Load Interrupt Mask (9940)	9	14-15
3000	LDCR	Load CRU	4	0-2,5
3400	STCR	Store CRU	4	0-2,5
3800	MPY	Multiply	9	—
3C00	DIV	Divide	9	4
4000	SZC	Set Zeroes Corresponding (Word)	1	0-2
5000	SZCB	Set Zeroes Corresponding (Byte)	1	0-2,5
6000	S	Subtract Word	1	0-4
7000	SB	Subtract Byte	1	0-5
8000	C	Compare Word	1	0-2
9000	CB	Compare Byte	1	0-2,5
A000	A	Add Word	1	0-4
B000	AB	Add Byte	1	0-5
C000	MOV	Move Word	1	0-2
D000	MOVB	Move Byte	1	0-2,5
E000	SOC	Set Ones Corresponding (Word)	1	0-2
F000	SOCB	Set Ones Corresponding (Byte)	1	0-2,5

INSTRUCTION FORMATS

FORMAT 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 GENERAL USE

FORMAT								GENERAL USE
1	OP CODE	B	T_D	DR	T_S	SR		ARITHMETIC
2	OP CODE			SIGNED DISPLACEMENT				JUMP
3	OP CODE			WR	T_S	SR		LOGICAL
4	OP CODE			C	T_S	SR		CRU
5	OP CODE			C		R		SHIFT
6	OP CODE				T_S	SR		PROGRAM
7	OP CODE				NOT USED			CONTROL
8	OP CODE				N	R		IMMEDIATE
9	OP CODE			DR	T_S	SR		MPY, DIV, XOP

OP CODE	OPERATION CODE
B	BYTE INDICATOR (1=BYTE)
T_D	DESTINATION ADDRESS TYPE*
DR	DESTINATION REGISTER
T_S	SOURCE ADDRESS TYPE*
SR	SOURCE REGISTER
C	CRU TRANSFER COUNT OR SHIFT COUNT
R	REGISTER
N	NOT USED

*T_D OR T_S	ADDRESS MODE TYPE
00	DIRECT REGISTER
01	INDIRECT REGISTER
10	PROGRAM COUNTER RELATIVE, NOT INDEXED (SR OR DR = 0)
10	PROGRAM COUNTER RELATIVE + INDEX REGISTER (SR OR DR>0)
11	INDIRECT REGISTER, AUTOINCREMENT REGISTER

SCHEMATIC OF CABLE TM 990/508
BETWEEN AUDIO CASSETTES
AND TM 990/302 BOARD
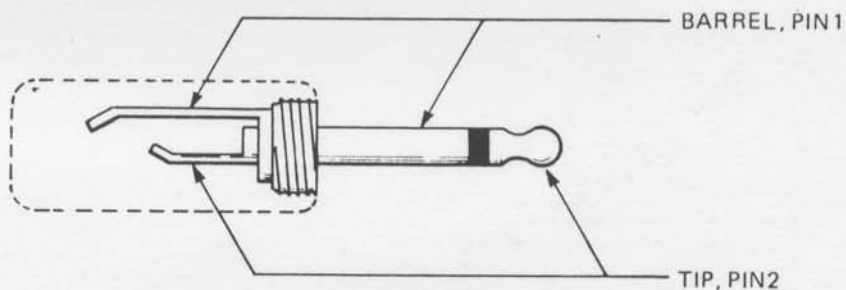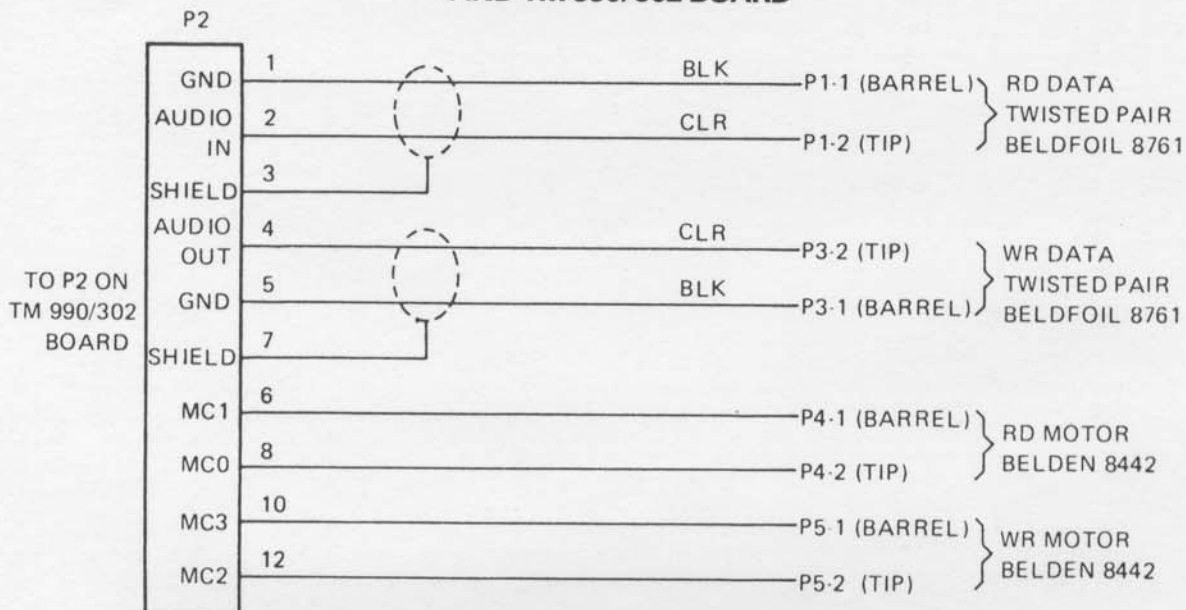
CABLE PLUG	AUDIO CASSETTE SOCKET
RD DATA	EAR OR MONITOR
WR DATA	AUX OR MIC OR LINE IN
RD MOTOR	REM
WR MOTOR	REM

CAUTION

Do not plug the AUX input and EAR output plugs into the same recorder at the same time as this may cause ground loop problems with some recorders.

Texas Instruments

TM 990/302 REFERENCE CARD

INSTRUCTIONS — ALPHABETICAL

MNEMONIC	OPERANDS	FORMAT	OP CODE	STATUS AFFECTED	INSTRUCTION
A	G,G	1	A000	0-4	ADD (WORD)
AB	G,G	1	B000	0-5	ADD (BYTE)
ABS	G	6	0740	0-4	ABSOLUTE VALUE
AI	WRC,I	8	0220	0-4	ADD IMMEDIATE
ANDI	WRC,I	8	0240	0-2	AND IMMEDIATE
B	G	6	0440		BRANCH
BL	G	6	0680		BRANCH AND LINK (R11)
BLWP	G	6	0400		BRANCH, LOAD WORKSPACE POINTER
C	G,G	1	8000	0-2	COMPARE (WORD)
CB	G,G	1	9000	0-2,5	COMPARE (BYTE)
CI	WR,I	8	0280	0-2	COMPARE IMMEDIATE
CKOF		7	03C0		CLOCK OFF (NOTE 1,3)
CKON		7	03A0		CLOCK ON (NOTE 1,3)
CLR	G	6	04C0		CLEAR OPERAND
COC	G,WR	3	2000	2	COMPARE ONES CORRESPONDING
CZC	G,WR	3	2400	2	COMPARE ZEROES CORRESPONDING
DEC	G	6	0600	0-4	DECREMENT (BY ONE)
DECT	G	6	0640	0-4	DECREMENT (BY TWO)
DIV	G,WR	9	3C00	4	DIVIDE
IDLE		7	0340		COMPUTER IDLE (NOTE 1,3)
INC	G	6	0580	0-4	INCREMENT (BY ONE)
INCT	G	6	05C0	0-4	INCREMENT (BY TWO)
INV	G	6	0540	0-2	INVERT
JEQ	PC	2	1300		JUMP EQUAL (ST2=1)
JGT	PC	2	1500		JUMP GREATER THAN (ST1=1)
JH	PC	2	1B00		JUMP HIGH (ST0=1 AND ST2=0)
JHE	PC	2	1400		JUMP HIGH OR EQUAL (ST0 OR ST2=1)
JL	PC	2	1A00		JUMP LOW (ST0=0 AND ST2=0)
JLE	PC	2	1200		JUMP LOW OR EQUAL (ST0=0 OR ST2=1)
JLT	PC	2	1100		JUMP LESS THAN (ST1 AND ST2=0)
JMP	PC	2	1000		JUMP UNCONDITIONAL
JNC	PC	2	1700		JUMP NO CARRY (ST3=0)
JNE	PC	2	1600		JUMP NOT EQUAL (ST2=0)
JNO	PC	2	1900		JUMP NO OVERFLOW (ST4=0)
JOC	PC	2	1800		JUMP ON CARRY (ST3=1)
JOP	PC	2	1C00		JUMP ODD PARITY (ST5=1)
LDCR	G,NOTE 4	4	3000	0-2,5	LOAD CRU
LDD	G	6	07C0		LONG DISTANCE DESTINATION (NOTE 1,2)
LDS	G	6	0780		LONG DISTANCE SOURCE (NOTE 1,2)
LI	WRC,I	8	0200	0-2	LOAD IMMEDIATE
LIMI	WRC,I	8	0300	12-15	LOAD INTERRUPT MASK IMMEDIATE (NOTE 1)
LMF		10	0320		LOAD MAP FILE (NOTE 1,2)
LREX		7	03E0	12-15	LOAD ROM AND EXECUTE (NOTE 1,3)
LWPI	WRC,I	8	02E0		LOAD IMMEDIATE TO WORKSPACE POINTER
MOV	G,G	1	C000	0-2	MOVE (WORD)
MOVB	G,G	1	D000	0-2,5	MOVE (BYTE)
MPY	G,WR	9	3800		MULTIPLY
NEG	G	6	0500	0-4	NEGATE (TWO'S COMPLEMENT)
ORI	WRC,I	8	0260	0-2	OR IMMEDIATE
RSET		7	0360	12-15	RESET AU (NOTE 1,3)
RTWP		7	0380	0-15	RETURN FROM INT. SUBR. (NOTE 1,8)
S	G,G	1	6000	0-4	SUBTRACT (WORD)
SB	G,G	1	7000	0-5	SUBTRACT (BYTE)
SBO	CRU	2	1D00		SET CRU BIT TO ONE
SBZ	CRU	2	1E00		SET CRU BIT TO ZERO
SETO	G	6	0700		SET ONES
SLA	WR,NOTE 6	5	0A00	0-4	SHIFT LEFT ARITHMETIC
SOC	G,G	1	E000	0-2	SET ONES CORRESPONDING (WORD)
SOCB	G,G	1	F000	0-2,5	SET ONES CORRESPONDING (BYTE)
SRA	WR,NOTE 6	5	0800	0-3	SHIFT RIGHT ARITHMETIC
SRC	WR,NOTE 6	5	0B00	0-3	SHIFT RIGHT CIRCULAR
SRL	WR,NOTE 6	5	0900	0-3	SHIFT RIGHT LOGICAL
STCR	G,NOTE 4	4	3400	0-2,5	STORE FROM CRU
STST	WR	8	02C0		STORE STATUS REGISTER
STWP	WR	8	02A0		STORE WORKSPACE POINTER
SWPB	G	6	06C0		SWAP BYTES
SZC	G,G	1	4000	0-2	SET ZEROES CORRESPONDING (WORD)
SZCB	G,G	1	5000	0-2,5	SET ZEROES CORRESPONDING (BYTE)
TB	CRU	2	1F00	2	TEST CRU BIT
X	G	6	0480		EXECUTE
XOP	G,NOTE 7	9	2C00	6	EXTENDED OPERATION
XOR	G,WR	3	2800	0-2	EXCLUSIVE OR

G — INDICATES ADDRESS INTO WHICH THE RESULT IS PLACED WHEN 2 OPERANDS ARE SPECIFIED.

NOTES

1. PRIVILEGED INSTRUCTIONS — MODEL 990/10
2. MODEL 990/10 WITH MAP ONLY
3. THE SECOND OPERAND IS THE INTERRUPT MASK
4. THE SECOND OPERAND IS THE NUMBER OF BITS TO BE TRANSFERRED, 0-15, 0=16
5. THE SECOND OPERAND SPECIFIES A MEMORY MAP FILE, 0 OR 1. THE SECOND OPERAND IS THE SHIFT COUNT, 0-15. 0 MEANS COUNT IS IN BITS 12-15 OF WORKSPACE REGISTER 0. WHEN COUNT = 0 AND BITS 12-15 OF WORKSPACE REGISTER 0=0, COUNT IS 16.
7. SECOND OPERAND SPECIFIES THE EXTENDED OPERATION, 0-15. DISPOSITION OF RESULT MAY OR MAY NOT BE IN THE FIRST OPERAND ADDRESS, DETERMINED BY THE USER.
8. WHEN PRIVILEGED INSTRUCTION BIT (7) OF STATUS REGISTER IS SET, ONLY BITS 0 TO 6 RETURNED TO STATUS REGISTER BY RTWP.

INSTRUCTIONS — NUMERICAL

OP CODE	MNEMONIC	FORMAT	STATUS AFFECTED
0200	LI	8	0-2
0220	AI	8	0-4
0240	ANDI	8	0-2
0260	ORI	8	0-2
0280	CI	8	0-2
02A0	STWP	8	
02C0	STST	8	
02E0	LWPI	8	
0300 (NOTE 1,8)	LIMI	8	12-15
0320 (NOTE 1,2)	LMF	10	
0340 (NOTE 1,3)	IDLE	7	
0360 (NOTE 1,3)	RSET	7	12-15
0380 (NOTE 1)	RTWP	7	0-15
03A0 (NOTE 1,3)	CKON	7	
03C0 (NOTE 1,3)	CKOF	7	
03E0 (NOTE 1,3)	LREX	7	12-15
0400	BLWP	6	
0440	B	6	
0480	X	6	
04C0	CLR	6	
0500	NEG	6	0-4
0540	INV	6	0-2
0580	INC	6	0-4
05C0	INCT	6	0-4
0600	DEC	6	0-4
0640	DECT	6	0-4
0680	BL	6	
06C0	SWPB	6	
0700	SETO	6	
0740	ABS	6	0-4
0780 (NOTE 1,2)	LDS	6	
07C0 (NOTE 1,2)	LDD	6	
0800	SRA	5	0-3
0900	SRL	5	0-3
0A00	SLA	5	0-4
0B00	SRC	5	0-3
1000	JMP	2	
1100	JLT	2	
1200	JLE	2	
1300	JEQ	2	
1400	JHE	2	
1500	JGT	2	
1600	JNE	2	
1700	JNC	2	
1800	JOC	2	
1900	JNO	2	
1A00	JL	2	
1B00	JH	2	
1C00	JOP	2	
1D00	SBO	2	
1E00	SBZ	2	
1F00	TB	2	2
2000	COC	3	2
2400	CZC	3	2
2800	XOR	3	0-2
2C00	XOP	3	6
3000	LDCR	4	0-2,5
3400	STCR	4	0-2,5
3800	MPY	9	
3C00	DIV	9	4
4000	SZC	3	0-2
5000	SZCB	3	0-5
6000	S	3	0-4
7000	SB	3	0-5
8000	C	3	0-2
9000	CB	3	0-2,5
A000	A	3	0-4
B000	AB	3	0-5
C000	MOV	3	0-2
D000	MOVB	3	0-2,5
E000	SOC	3	0-2
F000	SOCB	3	0-2,5

DEVICE NUMBERS (DEVNO's)

DEVNO	Device
0	Dummy
1	Terminal connected to P2
2	Cassette (RD MOTOR plug)
3	Cassette (WR MOTOR plug)
4	Terminal connected to P3

INSTRUCTION FORMATS

GENERAL USE — ADDRESS MODE TYPE

ARITHMETIC, JUMP, LOGICAL, CRU, SHIFT, PROGRAM CONTROL, IMMEDIATE, MPY DIV XOP

SR — SOURCE REGISTER
DR — DESTINATION REGISTER
CRU TRANSFER COUNT OR SHIFT COUNT
N — REGISTER
NOT USED
SIGNED DISPLACEMENT

ADDRESS MODE TYPE:
00 — DIRECT REGISTER
01 — INDIRECT REGISTER
10 — PROGRAM COUNTER RELATIVE *NOT PROGRAM COUNTER RELATIVE — INDEX REGISTER
11 — INDIRECT REGISTER AUTOINCREMENT REGISTER

B — BYTE INDICATOR (1=BYTE)
D — DESTINATION ADDRESS TYPE*
S — SOURCE ADDRESS TYPE*

STATUS REGISTER

0	1	2	3	4	5	6	7	8	9	11	12	15
L>	A>	C	O	P	X	MF	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
7 - PRIVILEGED MODE (0=PRIVILEGED)-MODEL 990/10
8 - MAP FILE — MODEL 990/10 WITH MAP OPTION

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

TEXT EDITOR*

Call: .TE \ <DEVNO>, <DEVNO>

Commands

?G	Get source
?P[1st Line], [last line]	Print line(s)
?<Line No.>, <Statement>	Insert line
?<Line No.>	Delete line
?R<1st Line>	Resequence lines
?K	Keep lines
?Q	Quit, enter monitor

Errors

21	Transmission error (e.g., checksum)
22	Line number too big
23	EOF detected before G command
24	Buffer too full for new source
25	Command not valid
26	Cannot resequence: K issued
27	Cannot print, 1st line no.> 2nd line no.
28	Cannot print, line number too small
29	Line number too high
30	Buffer too full for new lines
31	Resequence number above 9000
32	Line number to be above 9999

DUMP MEMORY*

.DM, <DEVNO>
?<Start Addr>, <Stop Addr>

BAUD RATE OF P3*

.SR
?<Baud Rate>
Rates: 110, 300, 1200, 2400, 4800, 9600, 19200 Hz

UPLINK*

.UL
?<L Addr>, <L. Bias>, <L. Length>, <S. Addr>

SYMBOLIC ASSEMBLER*

Call: .SA \ <DEVNO>, <DEVNO>, <DEVNO>
Source   Object   Listing

Assembler Directives

AORG	Absolute origin
BSS	Block starting with symbol
BYTE	Initialize byte
DATA	Initialize word
DREG	Define registers (R prefix)
DXOP	Define XOP
END	Program end
EQU	Define assembly-time constant
EVEN	Place on word boundry
IDT	Program identifier
TEXT	Text of ASCII characters

Errors

1	Invalid symbol
2	Multiply-defined symbol
3	Symbol table overflow
4	Mnemonic size too large
5	Undefined mnemonic
6	Illegal register number
7	Displacement too large (CRU inst)
8	Displacement too large (jump inst)
9	Shift count too large
10	Non-increasing location counter (AORG)
11	Byte value too large
12	No single quote to start TEXT string
13	IDT over 6 characters
14	IDT not beginning with quote
15	Illegal character in TEXT string
16	XOP number above 15
17	Undefined symbol
18	I/O Error
19	END directive needed

RELOCATING LOADER*

Call:
.RL <DEVNO>
?<Load Addr>, <Bias>, <Length>, <Start Address>

Errors

51	Invalid load tag
52	Checksum error

PROGRAM DEBUGGER*

Call: .DP

Commands

?EX	Execute
?IC \ <Address>, [Bits]	Inspect CRU
?IM \ <S. Addr>, <E. Addr>	Inspect Memory
?IR	Inspect hdwr regs
?IW \ [Reg. No.]	Inspect sftwr regs
?RU \ [No. of steps]	Run program
?SB<Addr>, [Addr]	Set breakpoint
?ST \ [No. of steps]	Set trace

EPROM PROGRAMMER*

Call: .EP
Program EPROM:
?PP<Type>, <M. Start>, <M. Stop>, <E. Start>, <P/I>, [B. Start]
Compare EPROM to Memory:
?CE - same as above -
Read EPROM into Memory:
?RE - same as above -
Verify EPROM is Erased:
?VE<Type>, <S. Addr>, <End Addr>

TM 990/302 SOFTWARE USER'S GUIDE
USER RESPONSE SHEET

It is our desire to provide our customers with the best documentation possible. After using this manual, please complete this sheet and mail it, postpaid, to us. Your comments will be appreciated.

1. Is the manual well organized? Yes___ No___ Comments:_____

_____

2. What subject matter could be expanded or clarified?_____

_____

3. Are the SDB commands adequtely covered? Yes___ No___ Comments:_____

_____

4. Are the system interconnections adequately covered? Yes ___ No ____
Comments:_____

_____

5. Is the cassette operation adequately covered? Yes ___ No ____ Comments:____

_____

_____

6. Place an X after the subject(s) that need further explanation:
Text Editor ____   Symbolic Assembler ____   Relocating Loader _____
Program Debugger _____   EPROM Programmer ____   Other _____
Comments:_____

_____

_____

_____

_____

_____

_____

_____

_____

Name_____

Address _____ State_____ ZIP____

School (if applicable)_____ Major_____ Yr____

Revision E

(Note: The customer should retain a record of the model number and
serial number identifying the returned product in the event that tracing
of the product...)

FACTORY REPAIR AND EXCHANGE POLICY

I. GENERAL

This policy specifies the conditions upon which a TM 990 product may be
returned for repair subject to the terms of TI's warranty for such product.

WHEN A PRODUCT IS DEEMED ACCEPTABLE FOR REPAIR BY TI AND NO REQUEST HAS BEEN
MADE FOR THE RETURN OF THE SAME SERIAL NUMBERED PRODUCT, TI RESERVES THE
OPTION TO REPAIR OR EXCHANGE THE PRODUCT.

EPROMS CONTAINING CUSTOMER GENERATED SOFTWARE SHOULD BE REMOVED FROM A PRODUCT
PRIOR TO SHIPMENT TO TI. THE REPLACEMENT BOARD MINUS SUCH EPROMS WILL THEN
BE SHIPPED TO THE CUSTOMER. TI ACCEPTS NO RESPONSIBILITY FOR CUSTOMER
GENERATED SOFTWARE SENT IN ON A UNIT TO BE REPAIRED.

II. NORMAL WARRANTY EXCHANGE OR REPAIR

Repair or exchange will be made free-of-charge provided:

    i.   The customer notifies TI of product failure within the applicable
        warranty period (90 days from the date of purchase from TI or from
        a TI authorized distributor) and

    ii. TI's inspection discloses that the product is defective and that the
        defect is not the result of accident, misuse, neglect, alteration,
        improper installation, unauthorized repair or improper testing.

THE CUSTOMER SHALL BE RESPONSIBLE FOR PROVIDING PROOF OF THE DATE OF PURCHASE.

III. NON-WARRANTY EXCHANGE OR REPAIR

Non-warranty exchanges or repairs will be made for a charge in accordance with
the schedule set forth in section VII below; provided such product ·is
·"repairable."

Product will be deemed "repairable" when the cost of repair does not exceed
cost of replacement. If a product is not "repairable," the customer will be
advised that repair cannot be effected and the product will be returned to the
customer.

IV. SHIPPING INSTRUCTIONS

    A. The following information must accompany the returned product. The TI
        Factory Repair and Exchange Questionnaire should be used to ensure
        that the required information is provided:

- Customer name and phone number
- Purchase order number (if applicable)
- Model number
- Serial number
- "Ship To" address; instructions for insurance and method of
  shipment (unless otherwise specified, TI will ship UPS, insured
  for the minimum)
- "Invoice To" address
- Description of symptoms of malfunction
- Type of service requested

(Note: The customer should retain a record of the model number and
serial number identifying the returned product in the event that tracing
of the product should be necessary.)

B. The product must be returned freight prepaid, F.O.B. TI's Factory
Repair Center at:

> Factory Repair Center
> TM 990 Microcomputer Systems
> Texas Instruments, Inc.
> 8500 Commerce Park Dr., Suite 110
> Houston, TX 77036

C. Should the customer have any questions regarding this policy or the
returned product status, he may contact the factory directly at:
(713) 778-5729.

V. CUSTOMER CONFIGURATION

When the customer has made modifications to the product, repair or replacement
will be "non-warranty."

TI will attempt the repair of such a product provided the customer has
restored the product to its standard configuration. Labor and material will
be charged at TI's then current standard rate for all necessary removals or
repairs to customer-made modifications, if such is required to test the
returned product in accordance with TI's specification for that product. TI
reserves the right to refuse the repair of any product that has been modified
such that the configuration as changed is untestable.

TI accepts no responsibility for additional memory, I/O terminator devices, or
other devices added to a particular configuration and shipped on a product to
be repaired.

VI. EXPEDITED SERVICE

Expedited service is available for an additional charge. It is available only
on <u>normal</u> exchanges, subject to availability of replacement product <u>confirmed</u>
<u>by phone prior</u> to shipment to TI. Replacement product will be shipped after
receipt of the customer's defective product.

VII. CHARGES, TURNAROUND TIME AND METHOD OF PAYMENT

		Charge * (In Warranty)	Charge * (Non-Warranty)	Estimated Maximum Turnaround
1.	Normal exchange/repair	N/C	$150	10 normal working days
2.	Repair with customer configurations	N/A	$200	15 normal working days
3.	Expediting charge	$50	$50	48 hours

* Subject to change without notice. When the total estimated repair charge
exceeds the standard charge such repair must have the prior written approval
of the customer.

The customer must send payment, either in the form of a money order or check (company, cashier, or certified) with the product unless a purchase order is submitted. Payment must be made payable to Texas Instruments.

Product returns from the repair facility shall be made FOB TI factory. Transportation and insurance charges (if applicable) will be added on the invoice.

TI will invoice the customer on the date TI returns the repaired or exchanged product to the customer.

VIII. WARRANTY

Repaired or exchanged assemblies may contain either new parts or refurbished parts of like quality and are warranted to be free of defects in material and workmanship for a period of 30 days from date of shipment, provided such warranty repair shall not operate to reduce the original product warranty.

THE FOREGOING WARRANTIES FOR GOODS ARE IN LIEU OF ALL WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND OF ANY OTHER WARRANTY OBLIGATION ON THE PART OF TI.

TM 990 MICROCOMPUTER SYSTEMS
TI Factory Repair and Exchange Questionnaire
(Please complete one questionnaire per product submitted)

SHIP TO			SOLD TO		
NAME/COMPANY			NAME/COMPANY		
ADDRESS-LINE 1			ADDRESS-LINE 1		
ADDRESS-LINE 2			ADDRESS-LINE 2		
CITY	STATE	ZIP	CITY	STATE	ZIP
USER NAME			USER PHONE NO. ( ) -		

(CHECK ONE) ☐ WARRANTY ☐ NON-WARRANTY	IF IN WARRANTY, IS PROOF OF PURCHASE DATE ENCLOSED? (CHECK ONE) ☐ YES ☐ NO

CHECK TYPE OF SERVICE REQUESTED ☐ NORMAL EXCHANGE/REPAIR ☐ REPAIR WITH CUSTOMER CONFIGURATIONS ☐ EXPEDITE	CUSTOMER P.O. NO.	AMT. OF PREPAID ($)
	MODEL NO. TM 990/	SERIAL NO.

RETURN SHIPPING-PACKING INSTRUCTIONS*	INSURED? (CHECK ONE)* ☐ YES ☐ NO	LEVEL OF INSURANCE ($)*

DESCRIPTION OF MALFUNCTION

BELOW FOR FACTORY USE ONLY					
ORDER REC'D DATE	SUFFICIENT PAYMENT	R/E	FRN	ORDER SHIP DATE	

* Unless otherwise specified, TI will ship UPS, insured for the minimum.

R&E-3

Please save this portion and return
it with your product should
repair be necessary.

Use this form if the form attached
to the warranty card is lost.

P/N 1602102-9701

IMPORTANT NOTICES

Texas Instruments reserves the right to make changes at any time in order to improve design and to supply the best product possible.

TI cannot assume any responsibility for any circuits shown or represent that they are free from patent infringement.

TEXAS INSTRUMENTS
INCORPORATED