CHAPTER 3

A First Encounter: Getting Your Hands on a 9900

PURPOSE/ WHERE TO BEGIN

PURPOSE

Remember the common saying, "What you've always wanted to know about subject X, but were always afraid to try." The same applies, and probably especially so, to persons who have contact with the world of digital electronics; who have heard about computers and minicomputers and·even operated them; who have seen and experienced the advances made in the functional capabilities and low cost of digital integrated circuits by owning and operating handheld calculators; who have worked around and even built electronic equipment; who have heard about microprocessors and their amazing capabilities — but have not tried them.

▶3 If you are one of these people, this chapter is for you, for in it we want to help you try out a microprocessor, work it together, operate it, have success with it. In this way we hope to demonstrate that microprocessor systems are not that difficult to use. That, even though they require an understanding of a new side of electronic system design — "software" — if a base of understanding is established, and if an engineering approach is followed, there is no need to fear getting involved.

So that's the purpose of this first encounter — to get your hands on a 9900 microprocessor system and operate it.

WHERE TO BEGIN

It would be very easy to be satisfied with a paper example for a first encounter, however, it has been demonstrated that a great deal more is learned by actually having the physical equipment and doing something with it. Therefore, this first encounter example requires that specific pieces of equipment be purchased.

However, the purchase is not to be in vain. The first encounter has been chosen so that is may be followed with more extensive applications described in Chapter 9. Applications that will help to bring understanding of the 9900 microprocessor system to the point that actual control applications, akin to automating an assembly line, can be implemented. Outputting control of ac and dc voltage for motors or solenoids and producing controlled logic level signals are examples. In this way, useful outcomes are being accomplished, the equipment is being expanded, and problem solutions are demonstrated. At all times, of course, the base foundation of knowledge about microprocessor systems is growing.

To get underway then, purchase the following items from your industrial electronics distributor that handles Texas Instruments Incorporated products.

Quantity	Part #	Description
1	TM990/100M-1 (Assembly No. 999211-0001) (see *Figure 3-1*)	TMS9900 microcomputer module with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumpers option.
1	TM990/301 (see *Figure 3-2*)	Microterminal
1	TIH431121-50 or Amphenol 225-804-50 or Viking 3VH50/9N05 or Elco 00-6064-100-061-001	100 pin, 0.125" c-c, wire-wrap PCB edge connector (or equivalent solder terminal unit)
1	TIH421121-20 or Viking 3VH20/1JND5	40 pin, 0.1" c-c, wire-wrap PCB edge connector (or equivalent solder terminal unit)

In addition, some small electronic parts to interconnect the light emitting diode displays that will be used will be needed. These are listed later on so you may want to continue to read further before purchasing the module and microterminal so that all necessary parts can be obtained at the same time.

WHAT YOU HAVE

In *Figure 3-3* is shown a generalized computer system, it has a CPU (central processing unit) which contains an arithmetic and logic unit (ALU), all the control and timing circuits, and interface circuits to the other major parts. It has a memory unit. It has some peripheral units for inputting data such as tape machines, disk memories, terminals and keyboards. It has output units such as printers, CRT screens, tape machines, disk memories.

The TM990/100M-1 microcomputer shown in *Figure 3-1* is a miniature version of this computer system as shown in *Figure 3-4.* It has a CPU centered around the TMS9900 microprocessor, a memory unit — in this case a random access memory (RAM) and a read only memory (ROM). It does not have the input/output units indicated in *Figure 3-3* but it does have circuitry (TMS9901, 9902) for interface to such units. The TMS9901 will handle parallel input/output data and single bit addressed data as will be shown in this first encounter. The TMS9902 handles serial input/output data interface either through a TTY interface. A more complete interconnection of the components of the microcomputer is shown in the block diagram of *Figure 3-5.* The physical position of these units on the board is identified in *Figure 3-1.*

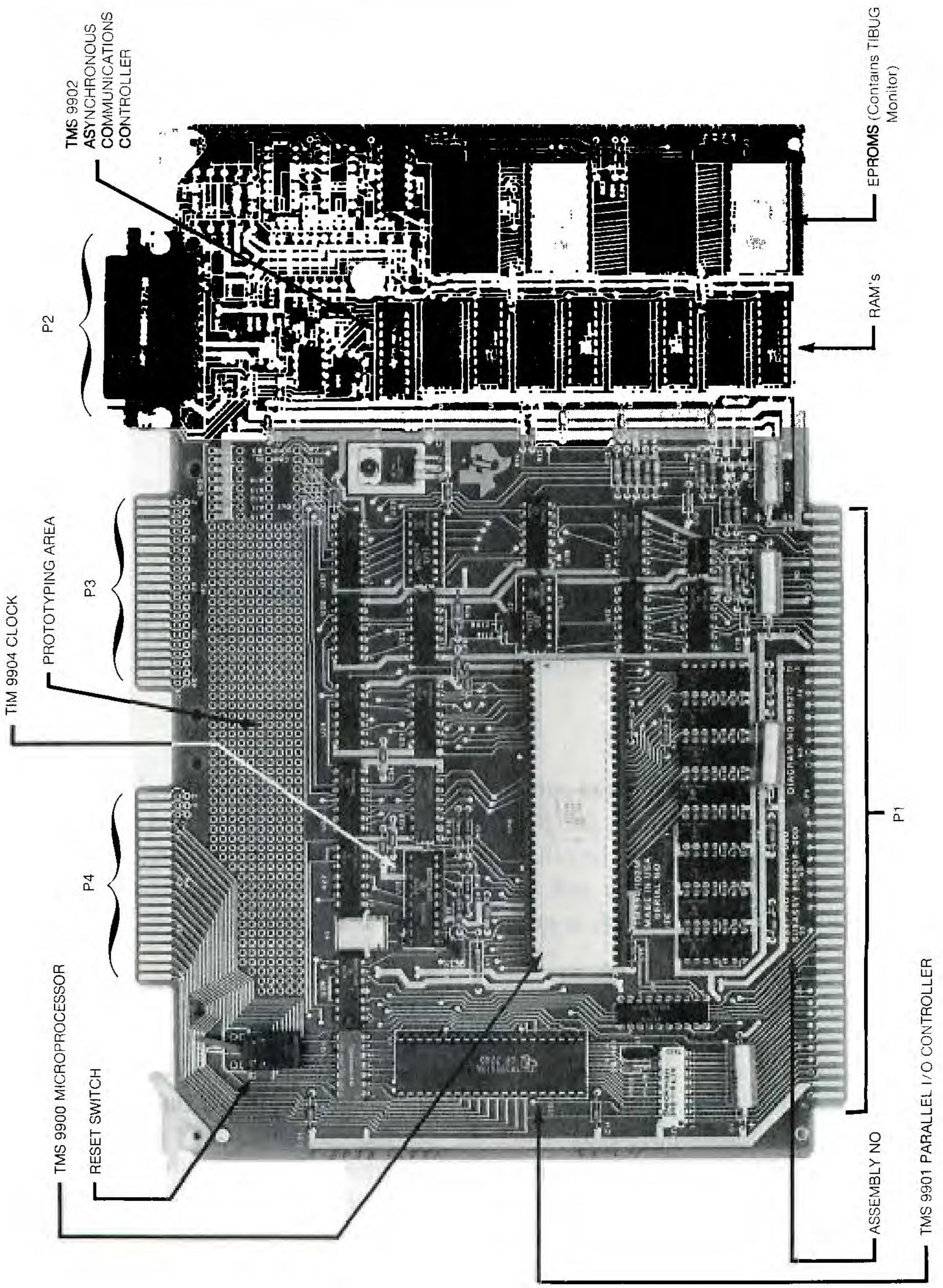

TMS 9902 ASYNCHRONOUS COMMUNICATIONS CONTROLLER
EPROMS (Contains TIBUG Monitor)
P2
RAM's
PROTOTYPING AREA
TIM 9904 CLOCK
P3
P1
P4
TMS 9900 MICROPROCESSOR
RESET SWITCH
ASSEMBLY NO
TMS 9901 PARALLEL I/O CONTROLLER

Figure 3-1. TM 990/100M-1 Microcomputer

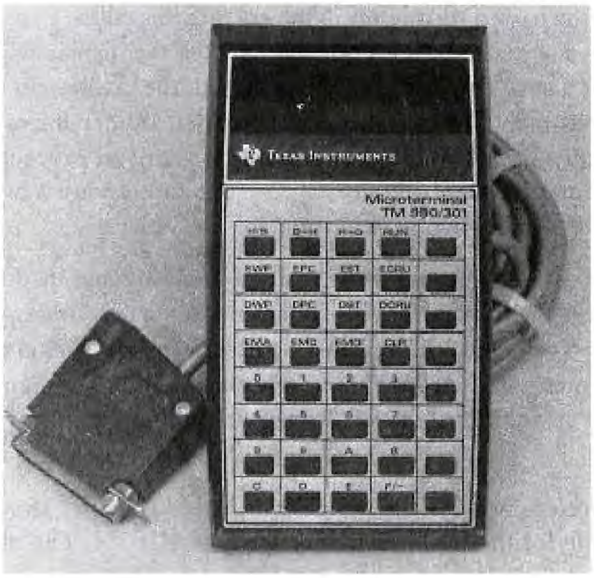


Figure 3-2. TM 990/301 Microterminal

Just think, a complete microcomputer with: **1)** 256 16-bit words of random access memory to hold program steps and program data, expandable to 512 words; **2)** 1024 16-bit words of read only memory which contains pre-programmed routines (TIBUG Monitor) that provides the steps necessary for the TM990/100M-1 microcomputer to accept input instructions and data and to provide output data. This ROM capability can be expanded to 4096 words to provide program flexibility; **3)** input/output interface that can handle 16 parallel lines expandable to 4096 and an interface for serial characters of 5-8 bits at a programmable data rate; **4)** an input terminal to input the sequence of steps to solve a problem — the program.

GETTING IT TOGETHER

Of course, in order to operate the microprocessor system, it must be put together. It must be interconnected.

What function will it perform? The first encounter application is shown in *Figure 3-6.* The microcomputer will be used to provide basic logic level outputs to turn on and off, in sequence, light emitting diode segments of a 7 segment numeric display element, the TIL303. This will demonstrate the "software" techniques used to provide dc logic levels at the I/O interface which through proper drivers can later be used to control solenoids, motors, relays, lights, etc.

In the first encounter application, the microterminal shown in *Figure 3-2* will be used to input the instructions and data required to perform the function.

Recall that a light emitting diode (LED) is made of semiconductor material and emits light when a current is passed through it in the correct direction. Each segment of the 7-segment display is a separate LED. Four segments of the display will turn on in the sequence f, b, e, c at a slow or a fast rate depending on the position of a switch, as shown in *Figure 3-6.* Each segment will first be turned on, then a short delay, then off, then a short delay. The sequence is continued with the next segment; proceeding around through 4 segments and then starting over again. The rate is varied by changing the delay in the sequence. The switch position controls the delay.

A 7-segment display is used because of its ready availability and its dual-in-line package. Only 4 of the segments will be programmed into the sequence although driver capability will be provided for 6 segments. This allows flexibility for the person doing the first encounter to experiment on their own to include the remaining 2 segments. A next step would be to provide an additional driver. In this way all 7 segments of the display can be included.

Here's what's required to provide the segment display. *Figure 3-7* shows the integrated circuit driver package for the LED segments, the SN74H05N. The physical package and a schematic are shown. It contains 6 open collector inverters, each capable of "sinking" 20 ma. A 14- or 16-pin dual-in-line socket is required. A wire-wrap one is shown. However, it could be a solder terminal unit just as well.

Figure 3-8 shows the 7-segment display physical package and schematic and a 14- or 16-pin DIP socket for interconnection. 100 ohm resistors for limiting current through the LEDs are also required.

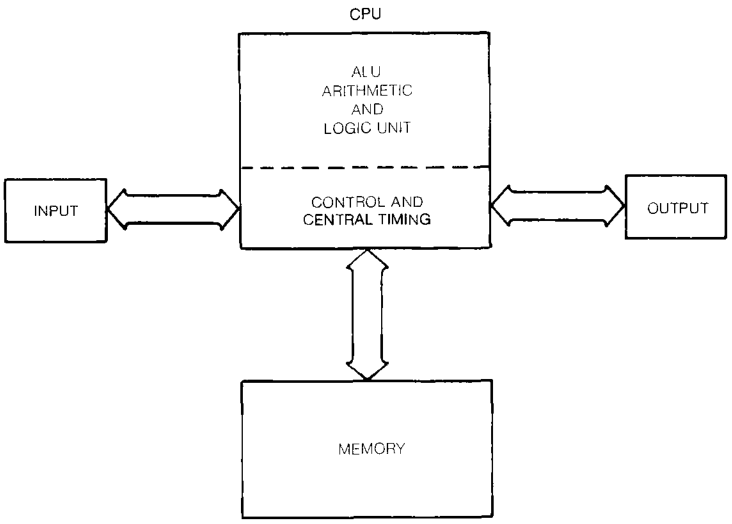


Figure 3-3. Generalized Computer

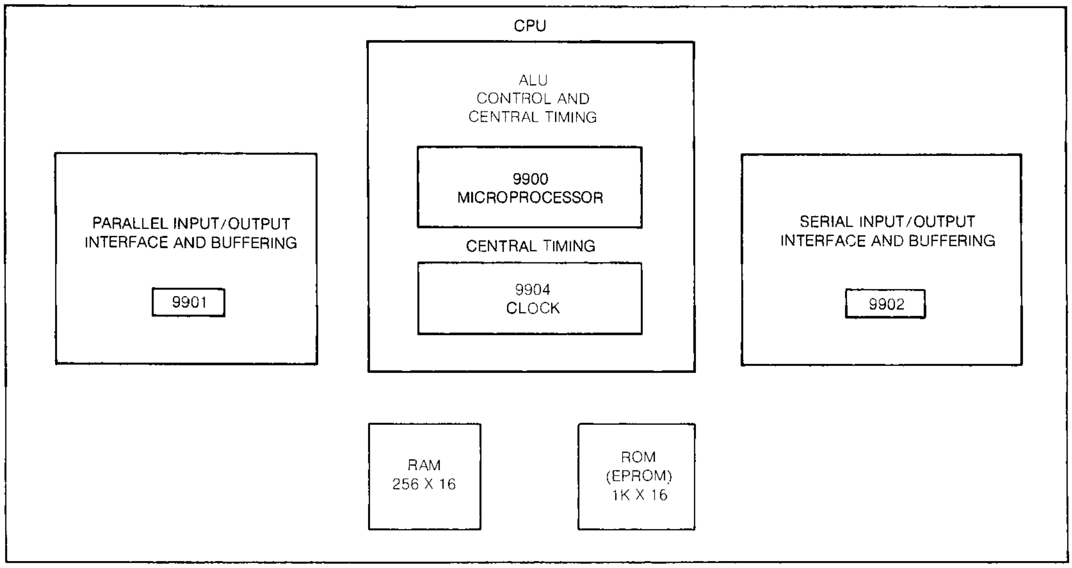


Figure 3-4. Miniature Computer System on TM 990/100M-1 Module

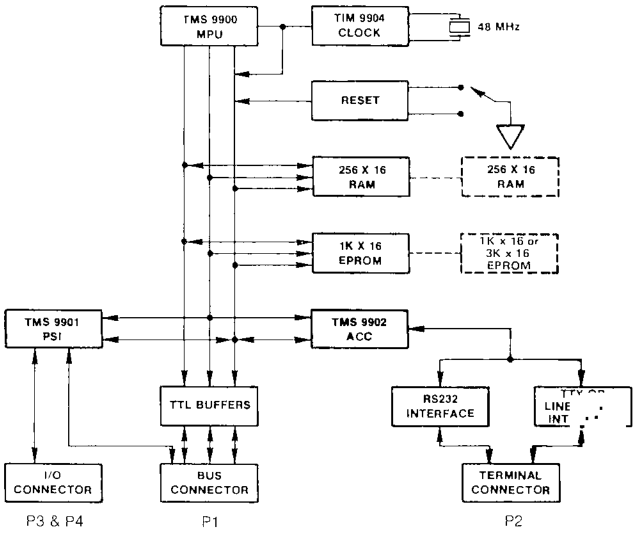


Figure 3-5. TM 990/100M-1 Block Diagram

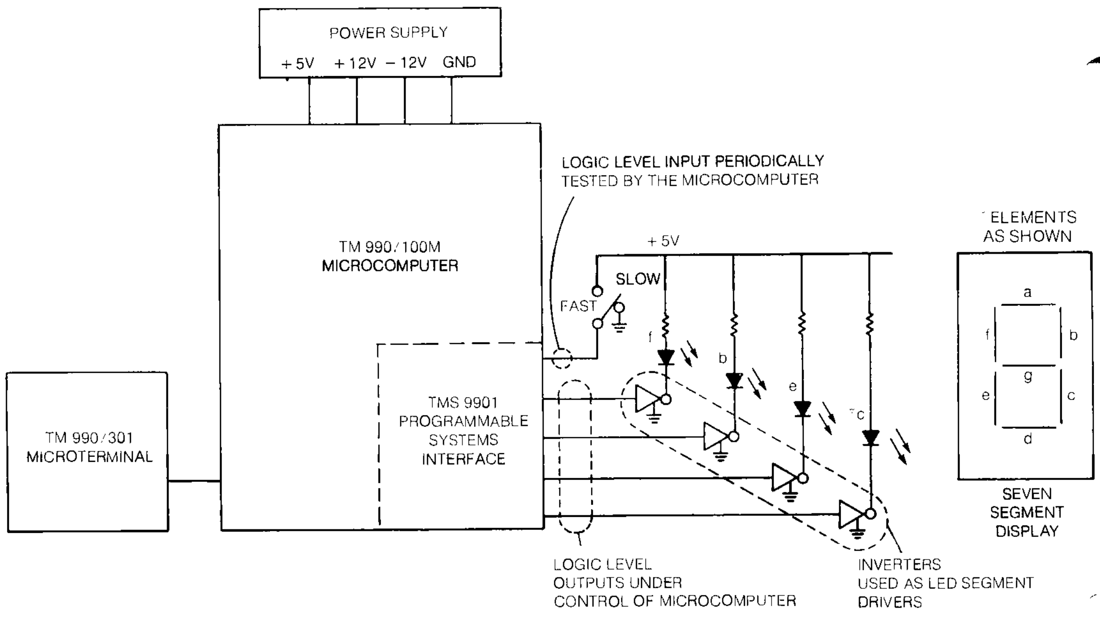
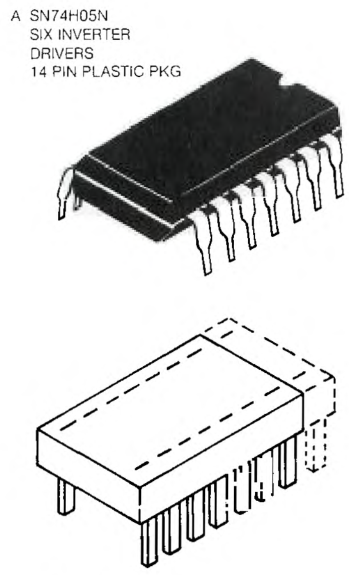
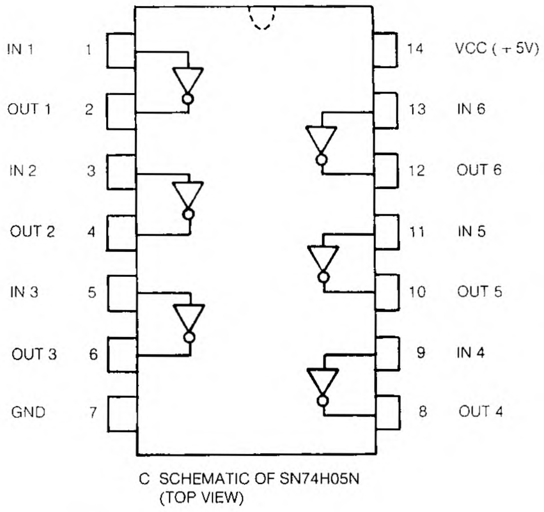

POWER SUPPLY
+5V +12V −12V GND
TM 990/100M MICROCOMPUTER
TMS 9901 PROGRAMMABLE SYSTEMS INTERFACE
TM 990/301 MICROTERMINAL
LOGIC LEVEL INPUT PERIODICALLY TESTED BY THE MICROCOMPUTER
+5V
SLOW
FAST
f
b
e
c
ELEMENTS AS SHOWN
a
g
d
SEVEN SEGMENT DISPLAY
LOGIC LEVEL OUTPUTS UNDER CONTROL OF MICROCOMPUTER
INVERTERS USED AS LED SEGMENT DRIVERS

Figure 3-6. The First Encounter Task

All of the components of *Figure 3-7* and *3-8* are wired together on a separate printed circuit board as shown in *Figure 3-9.* The Radio Shack #276-152 board provides individual plated surfaces around holes to make it easy to anchor components and to interconnect all components with wire-wrap. J4, the 40 pin wire-wrap PCB edge connector accepts the edge connections of P4 on the TM990/100M-1 board shown in *Figure 3-1.* After wiring this connector, put a piece of tape across the top of this connector so that it is correctly oriented before the board is plugged in; or the same can be done here as for P_1 discussed a little later. Note also on *Figure 3-1* that there is an area on the 990/100M-1 board for prototyping. The components of *Figure 3-9* may be wired in this area rather than using a separate printed circuit board. Using a separate board allows this area to be used for more permanent components for a specific dedicated application of the 990/100M module.



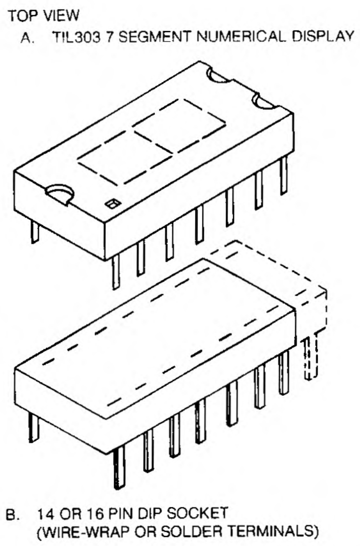


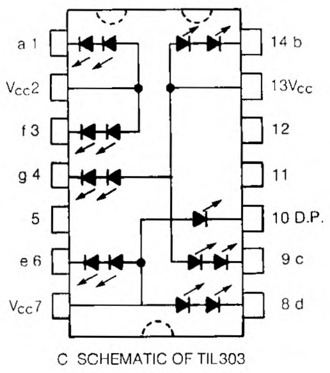
COMPONENT PARTS

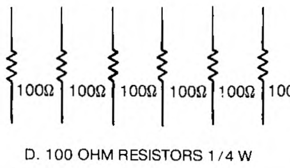
1 — SN 74H05N HEX DRIVER
(EACH DRIVER CAPABLE OF SINKING 20 MA)

1 — 14 OR 16 PIN DIP SOCKET
(RADIO SHACK #276-1993, 94)
(TI # 811604 M&C — 16 PIN WIRE-WRAP)

Figure 3-7. LED Driver Parts


TOP VIEW
A. TIL303 7 SEGMENT NUMERICAL DISPLAY
B. 14 OR 16 PIN DIP SOCKET
(WIRE-WRAP OR SOLDER TERMINALS)





COMPONENT PARTS

1 — 7 SEGMENT DISPLAY TIL303

1 — 14 OR 16 PIN DIP PACKAGE
(C-811604 M&C — 16 PIN WIRE WRAP)
(RADIO SHACK — 276 — 1993, 94)

6 — 100 OHM RESISTORS, 1/4 W

Figure 3-8. Segment Display Parts

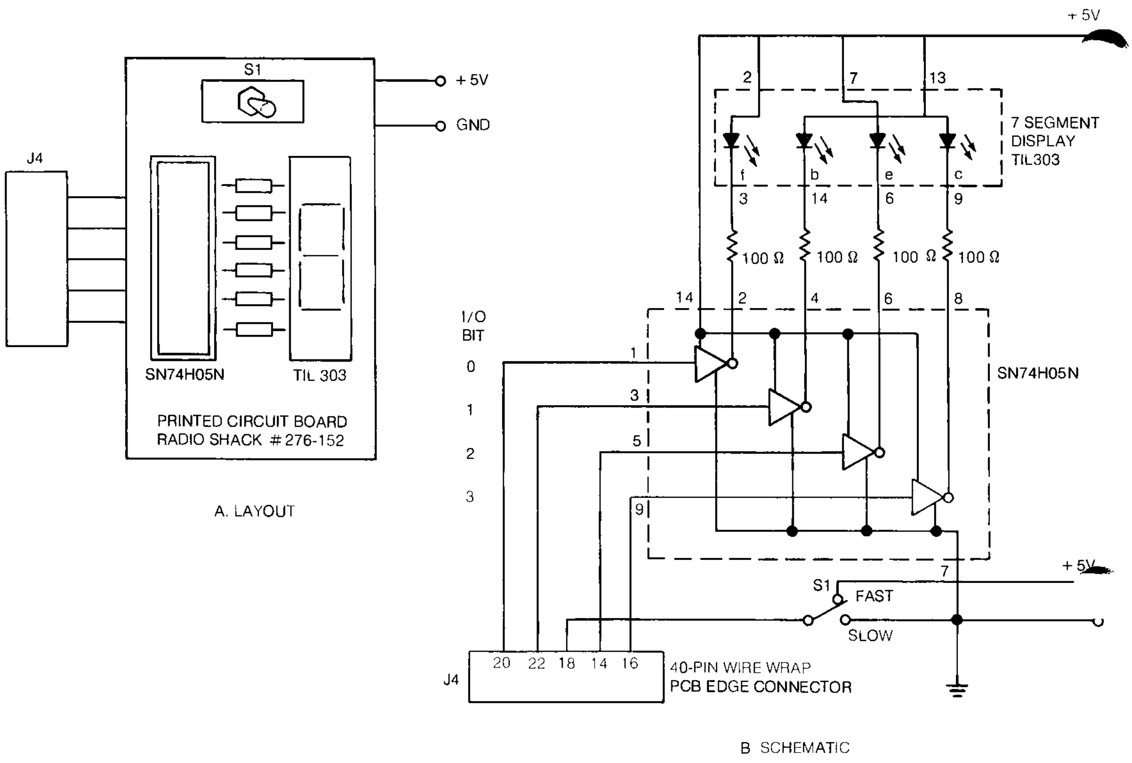


Figure 3-9. The Output Board

Following is a complete list of the parts, tools and supplies required. This is the list that was referred to earlier. Check carefully that all necessary parts are purchased.

PARTS LIST

A. *Microcomputer*

1 – TM990/100M-1	TMS9900 Microcomputer module with TIBUG monitor in two TMS 2708 EPROM's and EIA or TTY serial I/O jumper option.

B. *Terminal*

1 – TM990/301	Microterminal

C. *Output*

1 — Hex LED Driver	SN74H05N
1 — 7 Segment Display	TIL303
2 — 14 or 16 Pin Dip Sockets	TI wire-wrap; 16 Pin — C-811604 M&C; Radio Shack wire-wrap; 14 Pin 276-1993; 16 Pin 276-1994
6 — 100 ohm Resistors, ¼ W	
1 — Switch, Toggle or Slide, SPST or DPST	
1 — J4, 40 pin, 0.1″ c-c, wire-wrap	TIH421121-20
PCB Edge Connector (or equiv. solder terminal unit)	Viking 3VH20/1JND5
1 — Printed Circuit Board	Radio Shack #276-152

D. *Bus Connector* (Use for Power in First Encounter)

1 — J1, 100-pin, 0.125″ c-c, wire-wrap	TIH431121-50
PCB Edge Connector (or equiv. solder terminal unit)	AMPHENOL 225-804-50 Viking 3VH50/9N05 Elco 00-6064-100-061-001

E. *Power Supplies* — Regulated

Voltage	*Regulation*	*Current*
+5V	±3%	1.3A
+12V	±3%	0.2A
−12V	±3%	0.1A

F. *Tools*

Wire-wrap connector tool
Wire-wrap disconnecting tool
Wire stripper (30 G)

Soldering Iron

Long-nose pliers
Diagonal cutter
VOM, DVM, DMM

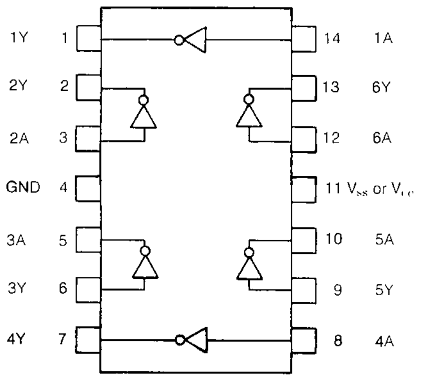
G. *General Supplies*

Wire (30 G Kynar)
Solder
Plugs and jacks for power supply connections

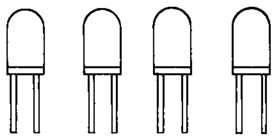
Note the power supplies required, the voltages, currents, and regulation. Assure that there is a common ground between all units.

(Electronic shops or laboratories might have available individual LEDs, therefore, *Figure 3-10* is provided in case this alternate method of display is chosen. The necessary drivers and resistors are identified. The necessary substitutions can be made on *Figure 3-9.*)

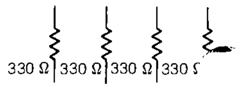
After wiring the output board, what remains is to supply power to the board. This is accomplished through P1 on the 990/100M-1 board. *Figure 3-11* shows how the edge connector is wired to supply power. Be careful to use the correct pins as numbered on P1 on the board; *these pin numbers may not correspond to the number on the particular edge connector used.* Label the top side of the edge connector "TOP" and the bottom "TURN OVER." This will prevent incorrect connection of power to board. Wire the connector pins so that the top and bottom connections on the board are used to supply power, e.g., 1 & 2 for ground; 3 & 4 for +5V; 73 & 74 for − 12V; and 75 & 76 for + 12V. Plugs or jacks may be placed on the end of the power supply wires to make easy interface. With both the P1 and P4 connectors and the output board wired, the total system is ready for interconnection.



A. SN75492N
MOS-LED DRIVER
(OPEN COLLECTOR)



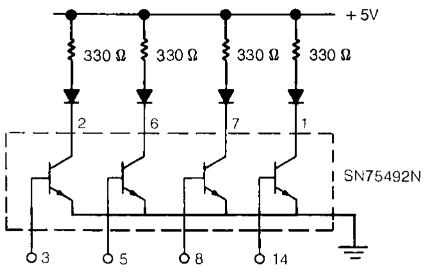
B. LIGHT EMITTING DIODES
TIL209



C 330 OHM RESISTORS
1/4 W

COMPONENTS PARTS

1 — SN75492N DRIVER
1 — 14-16 PIN DIP SOCKET
4 — TIL209 LED'S
4 — 330 Ω RESISTORS



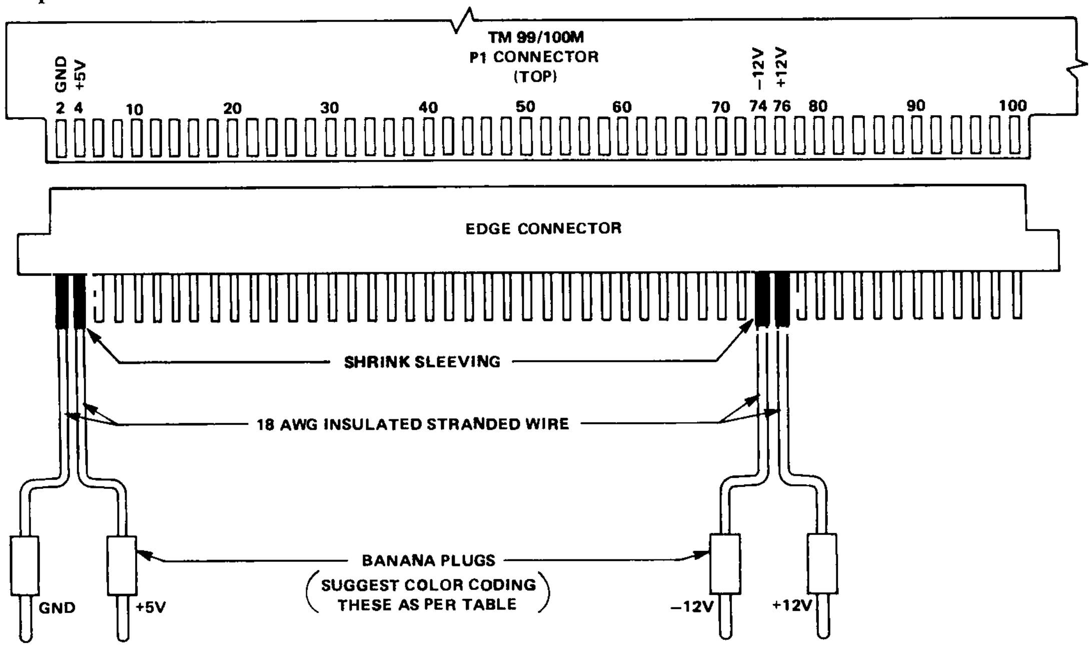
D. SCHEMATIC

Figure 3-10. Alternative LED Output Display

UNPACKING AND CHECKING THE MICROCOMPUTER (TM990/100M-1)

It is very important to realize that the microcomputer module has MOS (metal-oxide-semiconductor) integrated circuits on it. These circuits are particularly sensitive to static charge and can be damaged permanently if such charge is discharged through their internal circuitry. Therefore, make sure to ground out all body static charge to workbench, table, desk or the like before handling the microcomputer board or any components that go onto it.

After unpacking the TM990/100M-1 module from its carton and examining it for any damage due to shipping, compare it to *Figure 3-12* to determine the correct location of all parts. Additional detail is available in the user's guide shipped with the board. Make sure that EPROM TIBUG Monitor (TM990/401-1) units are in the U42 and U44 positions on the board. Make sure that the RAM integrated circuits are in the U32, 34, 36, and 38 positions.



VOLTAGE	P1 PIN*	SUGGESTED PLUG COLORS
+5V	3, 4	RED
+12V	75, 76	BLUE
−12V	73, 74	GREEN
GND	1, 2	BLACK

*ON BOARD, ODD-NUMBERED PADS ARE DIRECTLY BENEATH EVEN-NUMBERED PADS.

Figure 3-11. Power Supply Hookup for 990/100M-1 Microcomputer

CAUTION: Before connecting the power supply to P1, use a volt-ohmmeter to verify that correct voltages are present as shown in Figure 3-11.

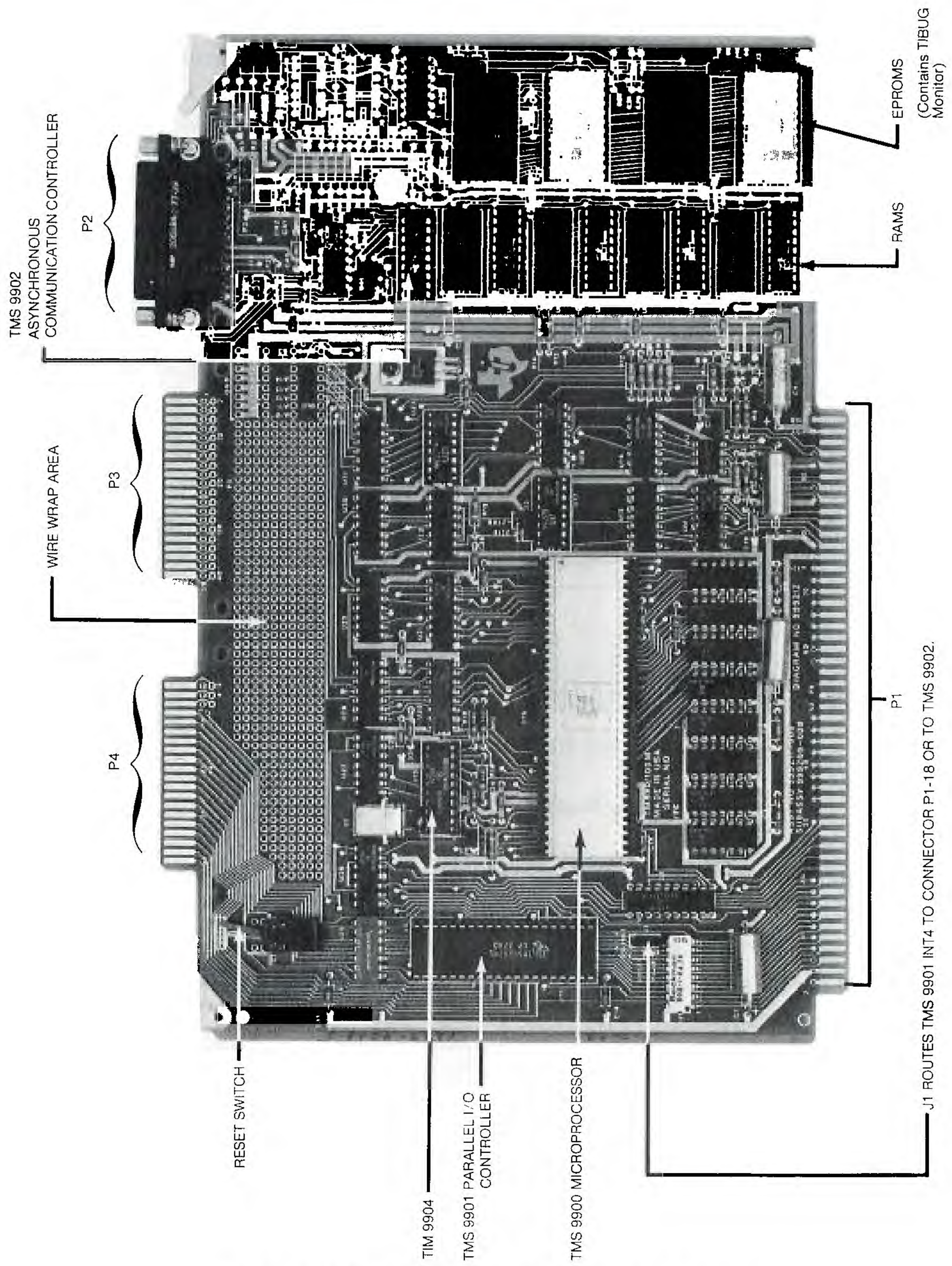

TMS 9902 ASYNCHRONOUS COMMUNICATION CONTROLLER
P2
WIRE WRAP AREA
P3
P4
RESET SWITCH
TIM 9904
TMS 9901 PARALLEL I/O CONTROLLER
TMS 9900 MICROPROCESSOR
J1 ROUTES TMS 9901 INT4 TO CONNECTOR P1-18 OR TO TMS 9902.
P1
RAMS
EPROMS (Contains TIBUG Monitor)

Figure 3-12. TM 990/100M-1 Module as Shipped

Compare the board to *Figure 3-12 & 3-13.* Make sure that the jumpers are in the following positions:

JUMPER	POSITION	JUMPER	POSITION
J1	P1-18	J4	08, 08
J2	2708	J7	EIA
J3	08, 08	J11	OPEN

They assure that memory locations are identified correctly and that the microterminal interfaces correctly.

CONNECTING THE MICROTERMINAL TM990/301

The microterminal *(Figure 3-2)* should be examined to verify there is no damage due to shipment. It will be connected to the microcomputer through P2 on *Figure 3-12.* Jumpers J13, J14, and J15 must be installed on the TM990/100M-1 board in order to supply power to the microterminal. Using the extra jumpers provided, short pins on the board at J13, J14, and J15 *(Figure 3-13).* Attach the plug on the microterminal cable to the P2 connector on the board.

OPERATING THE MICROCOMPUTER

Check once more that all wiring is correct for the output board *(Figure 3-9)*, the power connector *(Figure 3-11)* and the jumpers, then follow these steps:

Step 1 Begin with connectors to P1 or P4 disconnected

Step 2 Turn on power supplies and verify that all voltages are correct at the connector for P1. Turn off power supplies.

Step 3 Connect the power supply connector to P1. Make sure edge connector has the word "TOP" showing. Turn on −12V supply first, then +12V, then +5V.

Step 4 Verify the voltages of +5V, −12V, and +12V on the board printed wiring connections near the edge of the board between P2 and P3. Adjust power supplies or verify trouble if these are not correct.

Step 5 Verify the voltages of these terminals:

J13	+5V
J14	+12V
J15	−12V

If these are incorrect, correct the problem.

Step 6 Turn off power supplies. With the top edge of connector for P4 in correct position, connect output board to P4, turn on power supplies in same sequence as before, −12V, +12V, +5V.

The total setup should now look like *Figure 3-14* and the microcomputer is now ready to perform the task; all that's required is to tell it what to do.

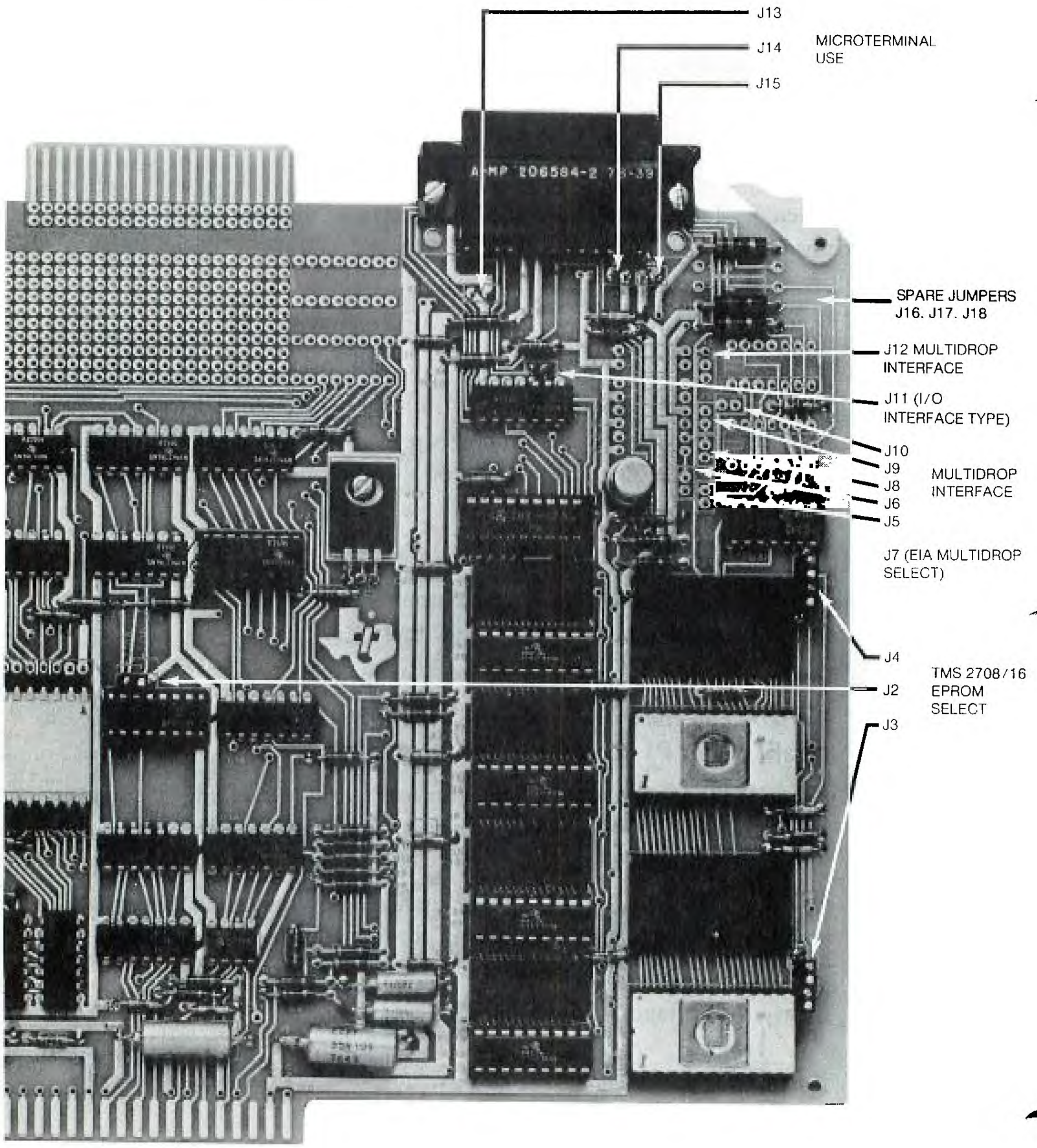


Figure 3-13. Jumpers used on TM 990/100M-1 Board for Option Selection

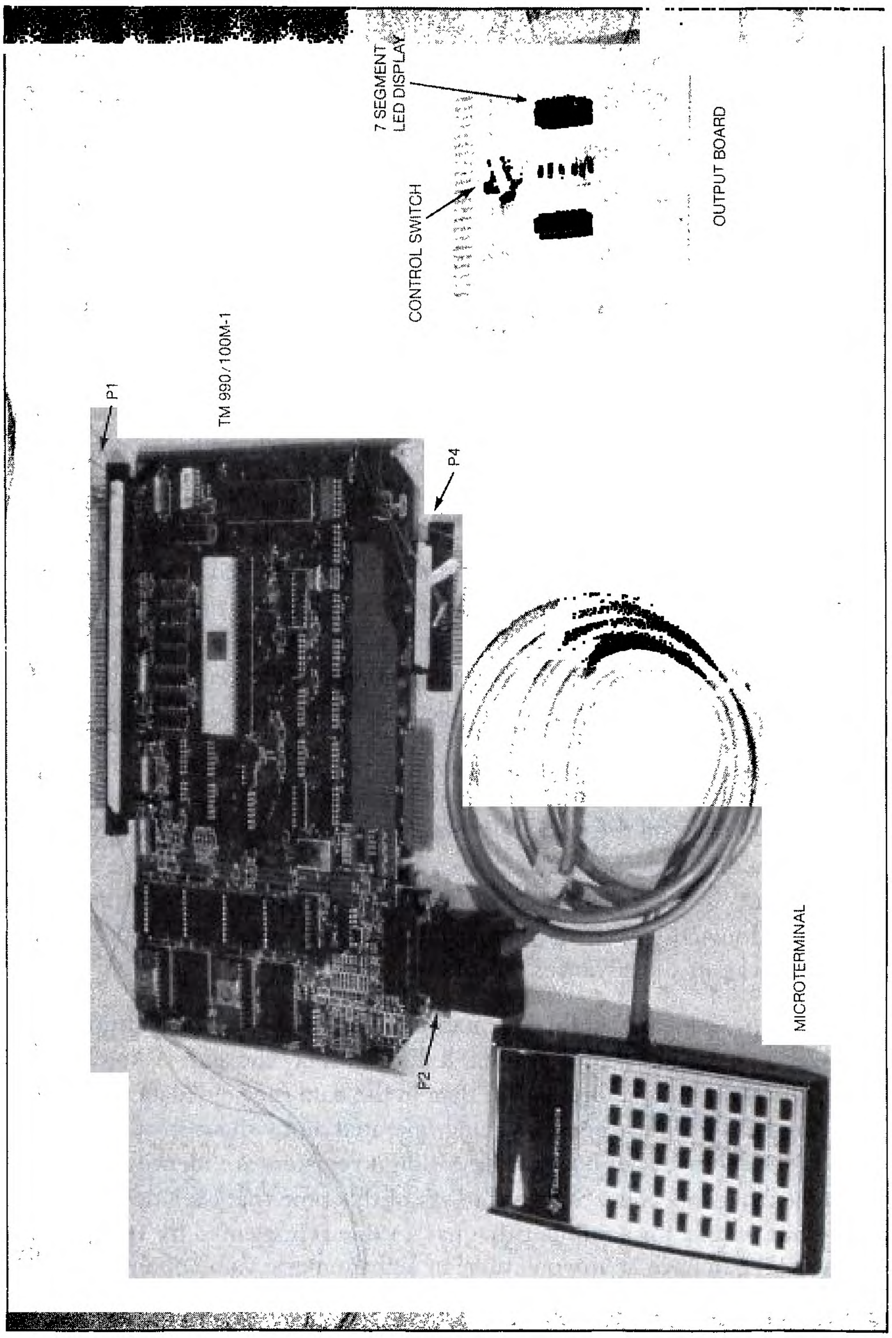


Figure 3-14. *Total System Connected*

TELLING THE MICROCOMPUTER WHAT TO DO

The microcomputer is told what to do through the microterminal keyboard. This is shown in *Figure 3-15.* Initial conditions are necessary so *Step 7* starts everything at an initial point.

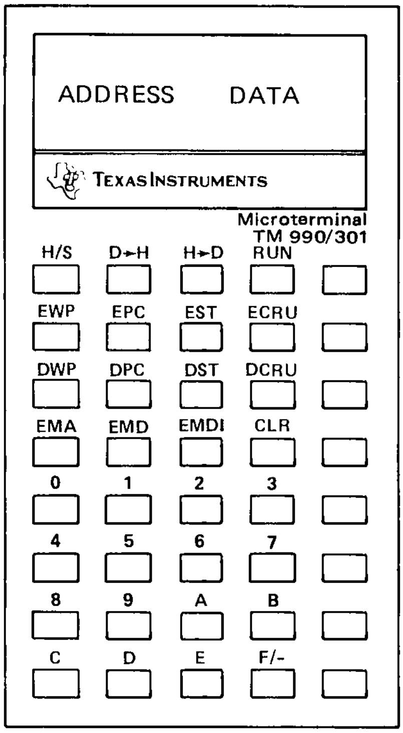
Step 7 *Figure 3-1* and *Figure 3-12* identify the RESET switch. Switch it all the way to the right (facing the toggle). Now depress the CLR (clear) key on the microterminal. Nothing will be on the display but to verify that it is working, press several of the number keys. The numbers pressed will appear in the display. Now press the CLR key again on the microterminal.

As we depress selected keys on the microterminal, the microcomputer is being given instructions — a step by step sequence of things to do to perform the first encounter task. The microcomputer is being *programmed* to do a job.

In order for the microcomputer to do its task according to the instructions given, it must also do many things dictated by other instructions that are stored in sequence in the TIBUG Monitor read-only memory (ROM). The program that performs the first encounter task is stored in the random access memory of the microcomputer and used in sequence. As a result, as the microcomputer accomplishes the task for which it is programmed, it performs each of the steps dictated by the "main program" in the RAM and by TIBUG in ROM.

There are only a few keys used on the microterminal for the first encounter. Identify these on *Figure 3-15* and on the microterminal. Three of these are: EMA (enter memory address) is used to display a specific memory address and give the user the ability to change the contents of that location. EMD (enter memory data) changes the contents of the memory location and EMDI (enter memory data and increment) changes the contents of the memory location and advances the address by two.

Note that *Figure 3-15* identifies the information given by the display. There are two banks of 4 digits each that are displayed. The left 4 digits display the address register (memory address) and the right 4 digits display the data in the data register (data to be stored in memory, being read from memory, or being operated on by the microcomputer). It is of no concern at the moment but both of these 4 digit registers are identifying the value of their data in hexadecimal code. Suffice it to say at this time that each hexadecimal digit represents 4 bits of data for a number that has a value represented by 16 bits. Each hexadecimal digit can have at any one time an alphanumeric value of any one of the following: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The decimal value of these numbers are shown in *Figure 3-16* as they occur in the place value position of the 4 bit display. Hexadecimal numbers will be identified with a subscript of 16 in the text, e.g., $02E0_{16}$ or 0100_{16} whenever there is need to avoid confusion.



The display of the microterminal is divided into two 4 hexadecimal digit banks. The left bank displays address register information and the right bank displays data registers.

Figure 3-15. Microterminal Keyboard and Display

Every program starts at a particular place in the RAM memory. The first encounter program will start at memory location identified by the hexadecimal address FE00. This is a 16 bit address which in machine code looks like this: 1111 1110 0000 0000 (F = 15; E = 14; 0 = 0; 0 = 0) and from *Figure 3-16* has a decimal value of 61,440 + 3584 + 0 + 0 = 65,024. The program starts at memory location 65,024.

To start the sequence of instruction steps for out first encounter, the starting address is entered and the EMA (enter memory address) key is depressed on the microterminal. This is program *Step 2* in *Step 8.* To help verify the steps the display data is also recorded.

Step 8

		Display
KEYSTROKES	ADDRESS	DATA
0. $\boxed{\text{CLR}}$	—	—
1. $\boxed{\text{F/-}}$ $\boxed{\text{E}}$ $\boxed{0}$ $\boxed{0}$		FE00
2. EMA	FE00	XXXX (X = Don't care)
3. $\boxed{0}$ $\boxed{2}$ $\boxed{\text{E}}$ $\boxed{0}$	FE00	02E0

This keystroke at Step 3 is a hexadecimal code — an instruction — that is telling the microcomputer to load a register with data. The data, however, is at the next address location. Therefore, with the next keystroke $\boxed{\text{EMDI}}$ (enter memory data and increment), the instruction 02E0 is stored at address location FE00 and the next memory address for an instruction is brought into the display by incrementing (advancing) the FE00 address by 2 (the reason for advancing by 2 will become clear as more is learned about the 9900 microprocessor).

Step 9

KEYSTROKE	ADDRESS	DATA
4. $\boxed{\text{EMDI}}$	FE02	XXXX

MSB **LSB**

	16^3		16^2		16^1		16^0	
BITS	0 1 2 3		4 5 6 7		8 9 10 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 decimal to hexadecimal find the nearest number in the above table less than or equal to the number being converted. Set down the hexadecimal equivalent then subtract its decimal number from the original decimal number. Using the remainder(s), repeat this process. For example:

$$\begin{array}{rcll} 31{,}362_{10} &=& 7000_{16} + 2690_{10} & \qquad 7000 \\ 2{,}690_{10} &=& A00_{16} + 130_{10} & \qquad A00 \\ 130_{10} &=& 80_{16} + 2_{10} & \qquad 80 \\ 2_{10} &=& 2_{16} & \qquad \underline{2} \\ & & & \qquad 7A82_{16} \end{array}$$

Figure 3-16. Place Value of Hexadecimal Digits in Significant Bit Positions

Program *Step 4* of operating *Step 9* shows this. Memory location identified by address FE02 is now ready for the data that will be put into the register identified by the instruction 02E0 at location FE00. The data is FF20.

5. $\boxed{F}$ $\boxed{F}$ $\boxed{2}$ $\boxed{0}$	FE02	FF20
6. $\boxed{EMDI}$	FE04	XXXX

Program *Step 6* has now advanced to the next memory location which is awaiting the next instruction which is keystroked in by program *Step 7.*

7. $\boxed{0}$ $\boxed{2}$ $\boxed{0}$ $\boxed{1}$	FE04	0201

Step 10

Continue now to program steps through the end of the program. Note how the address memory location advances by 2 each time $\boxed{EMDI}$ is pressed. This is how the program will be followed when it is run. The starting address FE00 will be loaded into the program counter. The program counter will then count by 2 and advance the microcomputer through each program step as the instructions are completed.

Keystroke	Address	Data
8. $\boxed{EMDI}$	FE06	XXXX
9. $\boxed{F}$ $\boxed{E}$ $\boxed{2}$ $\boxed{E}$	FE06	FE2E
10. $\boxed{EMDI}$	FE08	XXXX
11. $\boxed{0}$ $\boxed{2}$ $\boxed{0}$ $\boxed{C}$	FE08	020C
12. $\boxed{EMDI}$	FE0A	XXXX
13. $\boxed{0}$ $\boxed{1}$ $\boxed{2}$ $\boxed{0}$	FE0A	0120
14. $\boxed{EMDI}$	FE0C	XXXX
15. $\boxed{1}$ $\boxed{D}$ $\boxed{0}$ $\boxed{0}$	FE0C	*1d00
16. $\boxed{EMDI}$	FE0E	XXXX
17. $\boxed{0}$ $\boxed{6}$ $\boxed{9}$ $\boxed{1}$	FE0E	0691
18. $\boxed{EMDI}$	FE10	XXXX
19. $\boxed{1}$ $\boxed{E}$ $\boxed{0}$ $\boxed{0}$	FE10	1E00
20. $\boxed{EMDI}$	FE12	XXXX
21. $\boxed{0}$ $\boxed{6}$ $\boxed{9}$ $\boxed{1}$	FE12	0691
22. $\boxed{EMDI}$	FE14	XXXX
23. $\boxed{1}$ $\boxed{D}$ $\boxed{0}$ $\boxed{1}$	FE14	*1d01
24. ᵛ	FE16	XXXX
25. $\boxed{0}$ $\boxed{6}$ $\boxed{9}$ $\boxed{1}$	FE16	0691
26. $\boxed{EMDI}$	FE18	XXXX
27. $\boxed{1}$ $\boxed{E}$ $\boxed{0}$ $\boxed{1}$	FE18	1E01
28. $\boxed{EMDI}$	FE1A	XXXX
29. $\boxed{0}$ $\boxed{6}$ $\boxed{9}$ $\boxed{1}$	FE1A	0691
30. $\boxed{EMDI}$	FE1C	XXXX
31. $\boxed{1}$ $\boxed{D}$ $\boxed{0}$ $\boxed{2}$	FE1C	*1d02

*As displayed on 301 Terminal

KEYSTROKE	ADDRESS	DATA
32. EMDI	FE1E	XXXX
33. 0 6 9 1	FE1E	0691
34. EMDI	FE20	XXXX
35. 1 E 0 2	FE20	1E02
36. EMDI	FE22	XXXX
37. 0 6 9 1	FE22	0691
38. EMDI	FE24	XXXX
39. 1 D 0 3	FE24	*1d03
40. EMDI	FE26	XXXX
41. 0 6 9 1	FE26	0691
42. EMDI	FE28	XXXX
43. 1 E 0 3	FE28	1E03
44. ·· |	FE2A	XXXX
45. 0 6 9 1	FE2A	0691
46. ··	FE2C	XXXX
47. 1 0 E F	FE2C	10EF
48. EMDI	FE2E	XXXX
49. 1 F 0 4	FE2E	1F04
50. EMDI	FE30	XXXX
51. 1 3 0 5	FE30	1305
52. EMDI	FE32	XXXX
53. 0 2 0 3	FE32	0203
54. ··	FE34	XXXX
55. F F F F	FE34	FFFF
56. EMDI	FE36	XXXX
57. 0 6 0 3	FE36	0603
58. EMDI	FE38	XXXX
59. 1 6 F E	FE38	16FE
60. ··	FE3A	XXXX
61. 0 4 5 B	FE3A	*045b
62. EMDI	FE3C	XXXX
63. 0 2 0 3	FE3C	0203
64. EMDI	FE3E	XXXX
65. 3 F F F	FE3E	3FFF
66. ·· |	FE40	XXXX
67. 0 6 0 3	FE40	0603
68. EMDI	FE42	XXXX
69. 1 6 F E	FE42	16FE
70. ··	FE44	XXXX
71. 0 4 5 B	FE44	*045b
72. EMDI	FE46	XXXX

Step 11

All the program steps are now entered. It remains to run the program, that is, send the microcomputer through its sequenced steps to determine if it will accomplish the task.

Recall, that the system must be set to the initial conditions and to the starting point. This means that the system must start at memory address FE00 because that is where the first instruction is located.

Inside the microcomputer there is a register (a temporary storage location for 16 bits) that always contains the address of an instruction. It was previously noted that as the memory location of instructions was incremented by 2 as the program was entered, so also will the program counter be incremented by 2 by the microcomputer to go to the next instruction. Therefore, the initial conditions are accomplished by loading the program counter with the address location FE00. This is accomplished by an EPC key on the microterminal. The EPC (enter program counter) key changes the value of the program counter. It will enter into the program counter the value that is in the data register of the microterminal display.

The DPC (display program counter) key on the microterminal is depressed to determine if the correct value has been entered into the program counter because it displays the current value of the program counter.

The RUN key is depressed to begin execution of the program starting with the address in the program counter.

To run the program, go through *Steps 1* thru *5*.

Keystroke	Address	Data
1. CLR	—	—
2. F/- E 0 0	—	FE00
3. EPC	—	FE00
4. DPC	—	FE00
5. RUN	—	run

VOILA!

The first encounter task is being accomplished. Switching the toggle switch will change the rate of the segment display.

Under program control output logic levels on a set of output lines have been set to a "1", held for a time, set to a "0", held for a time, etc. in a particular sequence. The delay between "1s" and "0s" also is under program control. Such output levels then have been interfaced to driver circuits to accomplish a given task — in this case lighting LED segments of a display.

Step 12

To stop the program, depress H/S . The RESET switch on the microcomputer could also be pressed. (However, in doing so, to return to the program, go through the initial five steps of running the program at the end of operating *Step 10.*) The program may be started again by depressing RUN after it was halted by H/S

Step 13

If for some reason the first encounter task is not being accomplished after completing *Step 10*, the program can be checked by entering FE00, the beginning address and depress EMA . The contents of memory and the instruction at FE00 will be displayed. ▶3 Each memory location can then be examined by depressing EMDI and reading the display. In this manner, the program can be examined for an error. When the error is located, the correct data can be entered as it was in the original program and EMD is pressed. The program can then be run by returning to the initial sequence of operating *Step 11.*

The program may be entered at any valid address by entering the address and pressing IMA and then proceeding step by step with EMDI. There is no need to go back to the beginning address each time.

HOW WAS IT DONE?

The question naturally arises – how was this task accomplished by the microcomputer, and more importantly, how was the task taken from idea to the actual program? How does one know what to tell the microcomputer to do?

Of course, this will take a great deal of study of this book and much operation of systems, starting with the TM990/100M-1 microcomputer. The way the *idea* is turned into a *program* for the first encounter is covered in the remaining part of this chapter. This is a good foundation for building knowledge of the 9900 microprocessor, applying the 990/100M microcomputer to many other tasks, and understanding the use of the 9900 in solving other types of problems.

BACK TO BASICS

The process of understanding how the task was taken from idea to instructions for the microcomputer begins by returning to some basic concepts to assure that these are understood.

Recall that *Figure 3-4* identified the functional blocks of our microcomputer. The central processing unit includes the 9900 microprocessor. Examining *Figure 3-5* further and the functional block diagram of *Figure 3-17* shows that the 990/100M microcomputer is bus oriented. Recall that a bus is one or more conductors running in parallel which are used for sending information. The 9900 microprocessor sends an address to memory, to identify data required, on the 15-bit address bus. It receives data from memory on a 16-bit data bus. It should be noted that the same 15-bit address bus goes to the input/output interface units. The address bus is used either to send an address to *memory* or an address to *input/output*, not both at the same time. When the signal $\overline{\text{MEMEN}}$ is a logic low, the address bus is for memory. If the address bus is not for memory then it can be used by I/O. When the address is for I/O, the selection of which lines will be inputs or outputs is under control of the 9900.

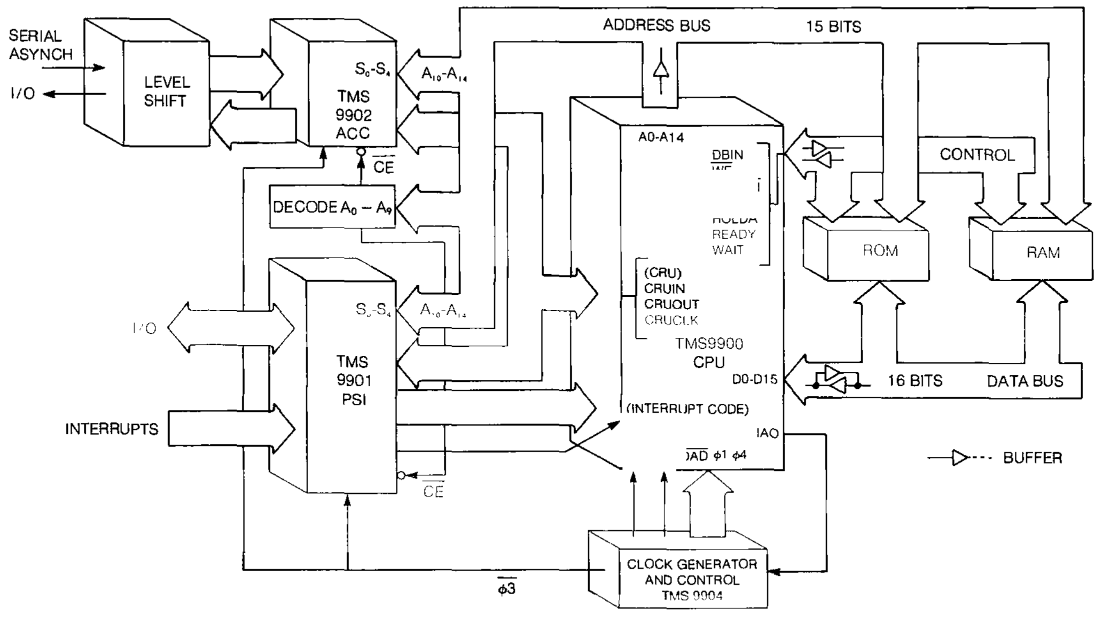


Figure 3-17. Functional Diagram of TM 990/100M-1 Microcomputer

Therefore, lines to accept data as input, or to deliver output data are selected by address bits in the same fashion that address bits locate data in a memory.

Examination of the architecture of the 9900 microcomputer in *Figure 3-18* reveals, as in *Figure 3-17,* the address bus, the data bus, signals for the CRU (the Communications Register Unit is an I/O interface for the 9900 architecture), signals for interrupt, control signals and master timing signals. Each of these are external signals. Further examination of internal parts is required to expand on more basic concepts, with emphasis on the ones that are used for the first encounter task.

REGISTERS

▶3 Recall that a register is a temporary storage unit for digital information. Inside the 9900 there are these types of registers: a memory address register, a source data register (data register), an instruction register, an interrupt register, some auxiliary registers like T_1 and T_2, and the registers that will be most applicable to the first encounter — the program counter, the workspace register, the status register and a shift register used as part of the hardware to select the input and output terminals. Additional parts include: 1) the ALU — it is the arithmetic and logic unit that performs arithmetic functions, logic and comparisons. 2) Multiplexers that direct the data over the correct path as a result of signals from the control ROM and control circuitry. 3) Timing circuits so that all operations are synchronized by the master timing.

Every time a piece of information is required to be stored in memory or retrieved (fetched) from memory, the memory must be told where the data is located or to be located. The memory address register holds the address to be put on the address bus for this purpose.

Data fetched from memory is received either by the source data register and distributed by the 9900 microprocessor as required, or by the instruction register when it is an instruction. The instruction is decoded and transmitted to the control ROM which sequences through microinstructions previously programmed into the control ROM to execute the instruction. The instruction might be "Increment register 1 by two". Instruction steps take the data from register 1 to the ALU which adds "2" and returns the data to register 1.

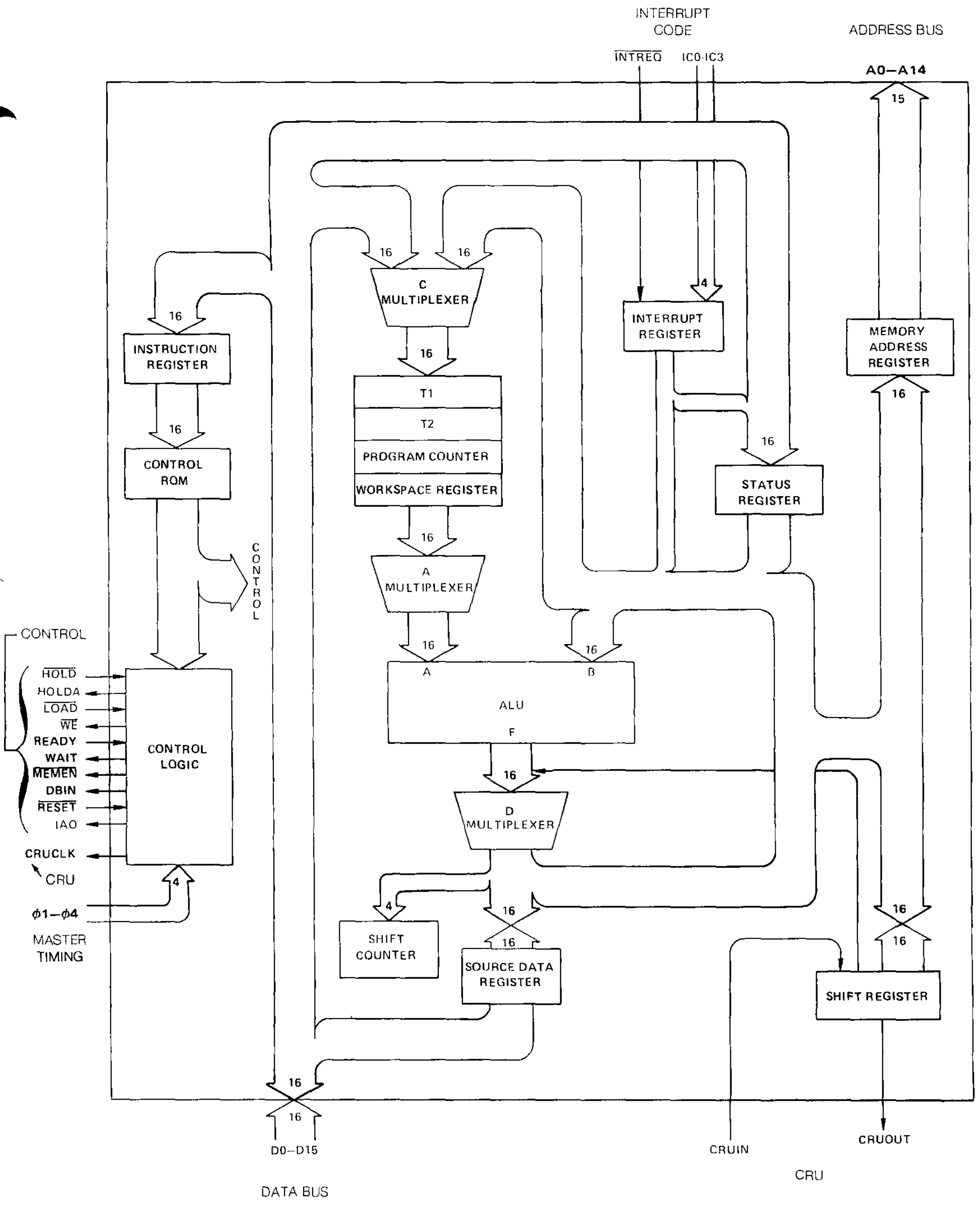


Figure 3-18. Architecture of 9900 Microprocessor

Two registers of significant concern for the first encounter task are the status register and the workspace register. The status register is just what the name implies. The 9900 microprocessor continually checks on how things are going (the status) by following instructions that command it to check various bits of the status register. *Figure 3-19* shows the bits of the status register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ST0	ST1	ST2	ST3	ST4	ST5	ST6	not used (=0)					ST12	ST13	ST14	ST15
L>	A>	=	C	O	P	X						Interrupt Mask			

Figure 3-19. Status Register

Each bit of the first 7 bits is concerned with identifying that a particular operation or event has or has not occurred as shown here.

Bit	Purpose	Bit	Purpose
0	Logical Greater Than	4	Overflow
1	Arithmetic Greater Than	5	Parity
2	Equal	6	XOP
3	Carry	12-15	Interrupt Mask

The last 4 bits are concerned with the interrupt signals and a priority code associated with the interrupts.

The first encounter uses bit 2, the "equals" status bit to change the time delay in the LED sequence.

Workspace

The workspace register is the same as the other registers, but it is used in a special way. As the 9900 microprocessor and the microcomputer step through program instructions, there is a need to have more registers than those available on the 9900. Instead of providing these registers in the 9900, a file of registers is set up in memory and a reference to this file saved in the workspace register. One of the rules in setting up this file is that it will always contain 16 registers in 16 contiguous (one following another in sequence) memory words. The workspace register on the 9900 is called the workspace pointer because, as shown in *Figure 3-20*, it contains the address of the first memory word in the contiguous register file, referred to for the application of the 9900 and in this book as "workspace registers" or just "workspace". The register file can be located anywhere within RAM that seems appropriate. In the total available memory space, there are certain reserved spaces for RAM, others for ROM, and others for special instructions. Therefore, the register file can only be set up in certain portions of memory. So, where 0200_{16} to $021E_{16}$ are the 16 locations shown in *Figure 3-20a,* with the workspace pointer being 0200_{16}, the file could have started at 0300_{16} and extended to $031E_{16}$ as long as these are allowable locations in the overall memory matrix. The workspace pointer would contain 0300_{16} in the second case.

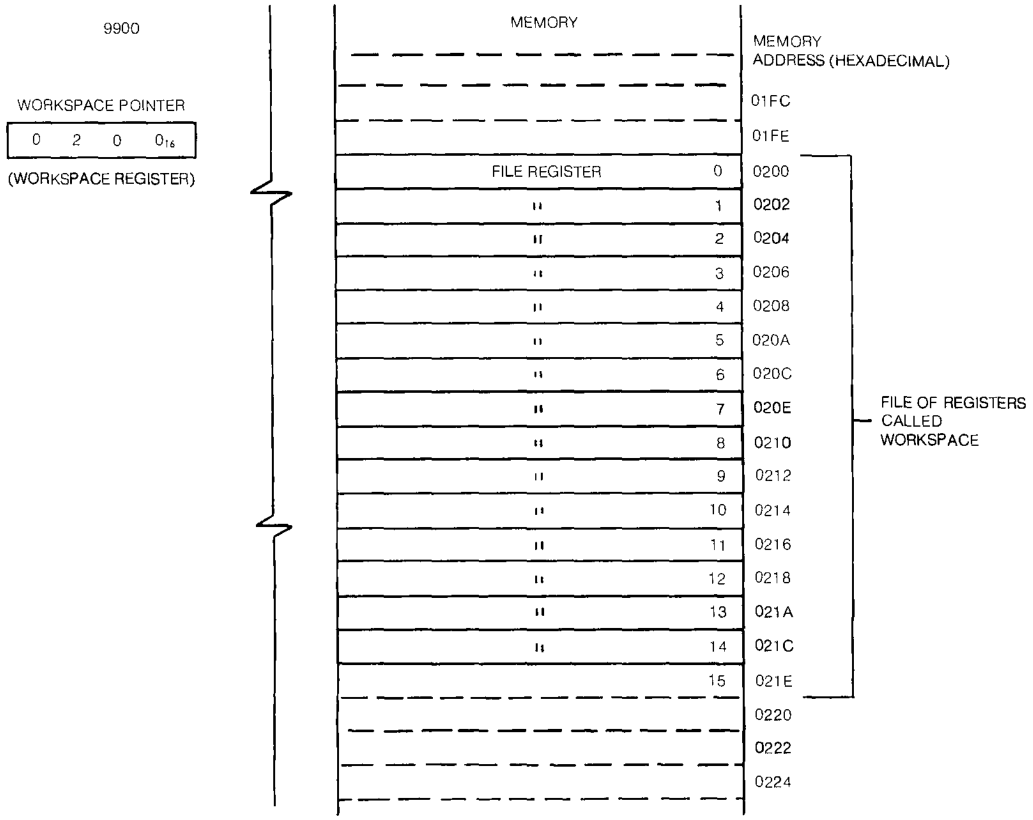


Figure 3-20a. *Workspace Registers*

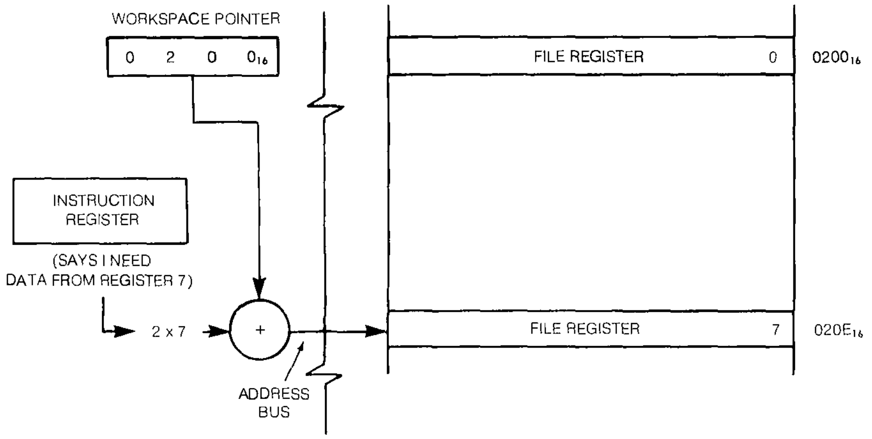


Figure 3-20b. *Locating Specific Register*

To locate a specific register in the workspace file, the 9900 microprocessor adds the register number to the workspace pointer address to obtain the address of the specific register in the file that is required. (It actually adds 2R, where R is the register number, so that the addresses advance by even numbers. The odd number addresses are used when the word contents are to be processed in 8-bit bytes.) For example, if register 7 contains the information required by the 9900 microprocessor, then the address 020E in *Figure 3-20a* is obtained by adding 14 to the workspace pointer at 0200_{16}. This is shown in *Figure 3-20b.* In like fashion, if the workspace pointer contained 0300_{16}, then adding 14 to 0300_{16} gives $030E_{16}$ the address of register 7, the 7th register down in the file.

Recall that to accomplish the first encounter task, logic levels on output lines had to be set to a "1" or a "0" in order for the LED drivers to turn on or turn-off the LED segment respectively. Recall, also, that the particular output lines could be selected. To understand how this is done, refer to *Figure 3-21.* This figure is divided into three bounded regions; the TMS 9900, Memory, and the TMS 9901. The output line from the 9900 microprocessor that will do the setting is the line "CRUOUT." It is coupled to the TMS 9901, the programmable systems interface.

The TMS 9901 contains more functional parts to handle the interrupt code and interrupt input signals but for now the part that is important is that shown in *Figure 3-21.* The portion shown is a demultiplexer. The data appearing at CRUOUT is strobed by CRUCLK into latches feeding the output pins. The particular latch and the particular output line is selected by the code that exists on the select bit lines S_0, S_1, S_2, S_3, and S_4, which, as shown in *Figure 3-21*, are the address lines A_{10} through A_{14}. The code on S_0 through S_4, and the CRU logic selects the output latch and line that is to be set. The "1" or "0" on CRUOUT does the setting. The latching occurs when CRUCLK strobes the data in.

SBZ AND SBO INSTRUCTIONS

Enough basics have now been covered to begin understanding several of the important instructions for the first encounter task. *Figure 3-21* will again be used and will be followed from left to right and top to bottom starting with the upper left corner. At a particular step in the program, controlled by the program counter, the instruction address (the bit contents of the program counter) is sent to memory over the address bus to obtain the instruction. Memory is read and the instruction is received by the 9900 on the data bus and placed in the instruction register. Via the control ROM and the control logic, the instruction is interpreted as an SBO instruction — "set CRU bit to one." The 9900 is designed so that it generates the correct S_0-S_4 address for the TMS9901 that selects the output line to be set to a "1" by the instruction. However, as indicated in *Figure 3-21,* first an ALU operation must occur before the correct address is obtained. The ALU adds the contents of one of the registers in the file, workspace register 12 (WR12), to a portion of the instruction, SBO. This portion of the SBO instruction is identified as DISP (meaning displacement) in *Figure 3-21.* It identifies the specific line to be used in the 9901 for the output. Eight bits are used for the signed displacement (7 and a sign). Bits 3 through 14 are used from the workspace register 12.

After the ALU operation, the address is sent out on the address bus. Because the $\overline{\text{MEMEN}}$ line is not active, this tells the 9901 that the address is for I/O. All 15 address bits are there; however, only A_3 through A_{14} are used for the effective CRU address. A_{10} through A_{14} provide S_0 through S_4 for the 9901, while bits A_0 through A_9 are used for decoding additional I/O as shown in *Figure 3-17.* A_0, A_1, and A_2 are set to zero for all CRU data transfer operations.

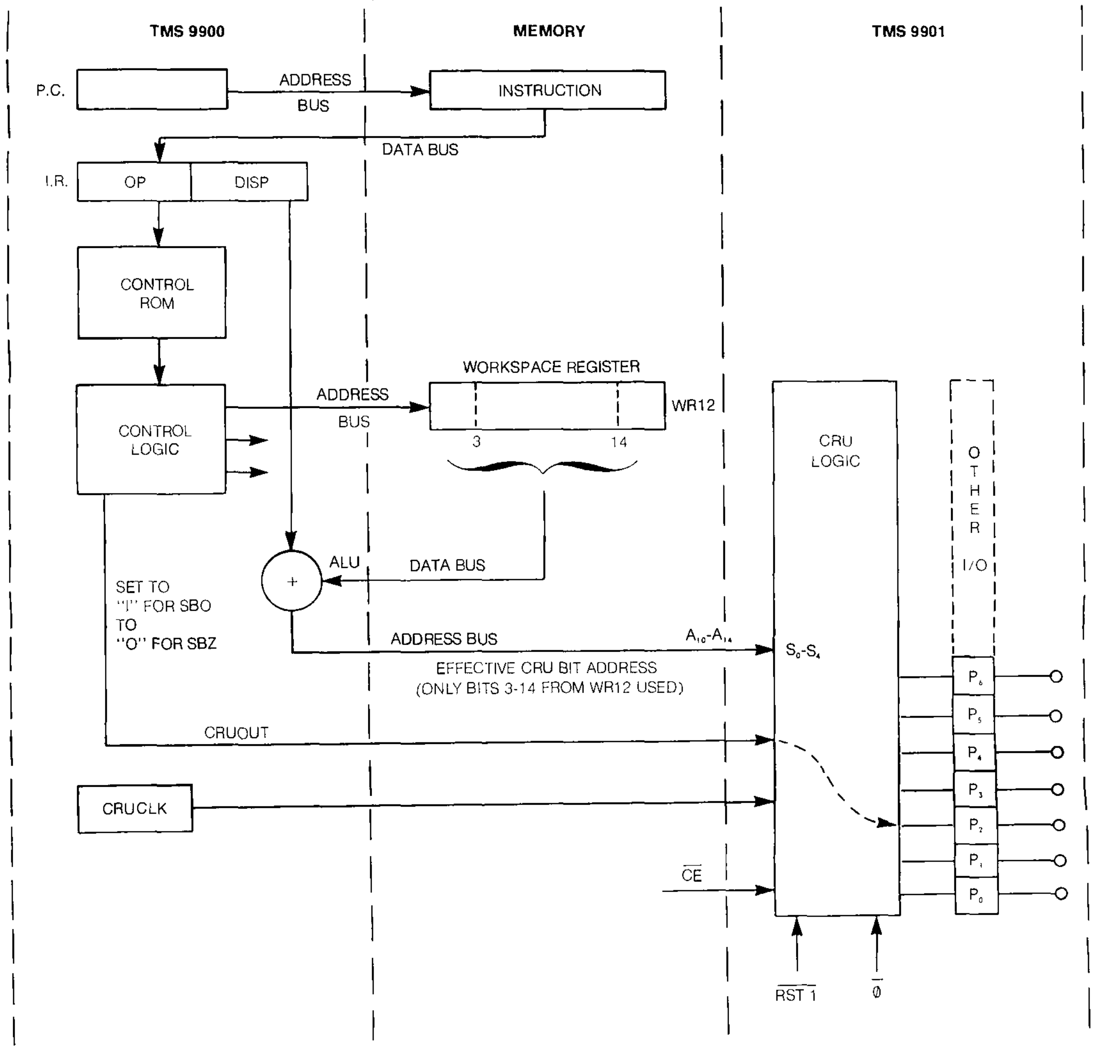


Figure 3-21. CRU Concept – Single Bit Output SBO or SBZ on CRUOUT.

TB INSTRUCTION

Besides setting logic levels on output pins, an additional system requirement for the first encounter task is to receive an input on an input line. One way of accomplishing this is to have the 9900 microprocessor look at a selected input line, bring the information present at a specified time into the 9900 and then examine the information, or test it, to determine if the information was a "1" or a "0". The TB instruction, "Test CRU Bit", accomplishes bringing the information into the 9900. Subsequent instructions are added to determine if the information was a "1" or a "0".

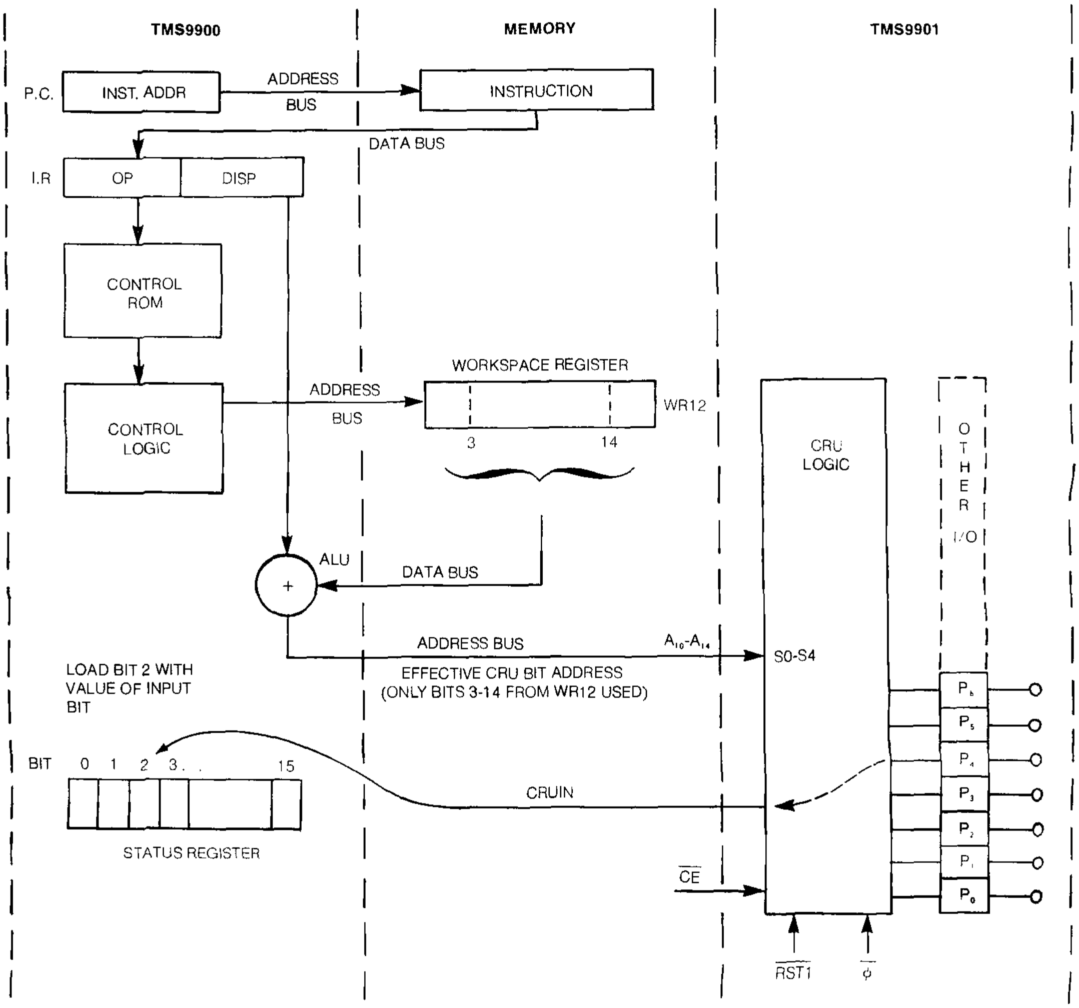


Figure 3-24. CRU Concept-Single Bit TB Input on CRUIN

The selection of the particular line in the I/O unit is the first concern. *Figure 3-24* shows that this is done in the same way as just explained for the SBO and SBZ instructions. The same portions of the 9901 are used as for the SBO or SBZ instructions except now these portions are a data selector. Data is selected from one of multiple input lines and sent to the 9900 microprocessor along the CRUIN line. The value of the information on the line is placed in bit 2 position of the status register. As discussed previously for the status register, instructions must follow the TB instruction that will examine bit 2 of the status register to determine what to do if this bit is a "1" and what to do if it is a "0". Conditional jump instructions are used to make the decision based on the value of the data. Note again that this is done one bit at a time.

Accomplishing a TB instruction requires that a base address be given for the particular input or output line desired. This hardware base address adjusted to a software base address is placed in workspace register 12. With the TB instruction, a displacement is given that identifies the particular line which needs to be sampled. This again is the same as for SBO. The line selected provides data straight through to the CRUIN line — there are no latches, as with the output data.

Thus, the basic concepts studied have shown the means of getting data to the output and bringing data in from an input — one bit at a time. They have shown how data is located, read, transferred, stored, and operated on arithmetically. With this, it should be possible now to get the first encounter idea into a sequence of steps — a program for the microcomputer to follow.

IDEA TO FLOWCHART

Bringing the idea from concept to program begins with a concept level diagram as shown in *Figure 3-25.* It has been decided that the microcomputer is to do the first encounter task; turn on and off 4 lights in sequence, with a time delay between each light activation.

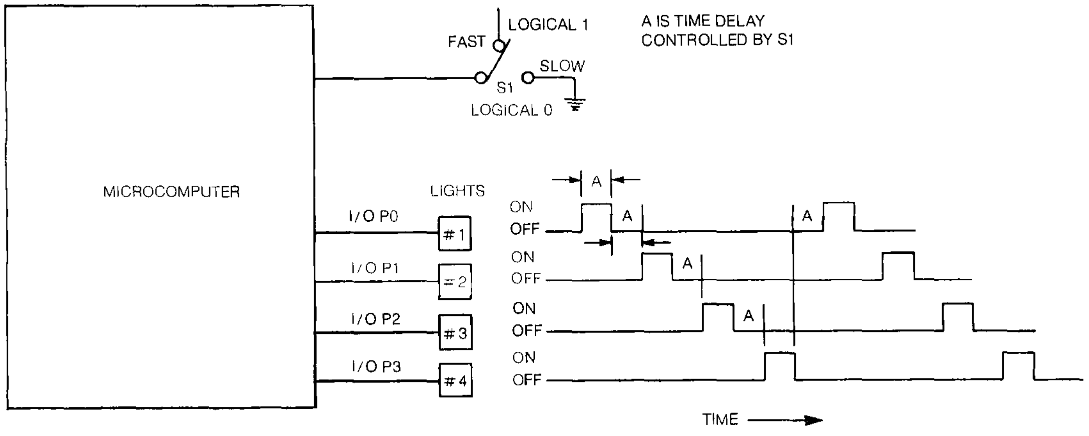


Figure 3-25. Concept Level Diagram

The time delay is to be under control of an external switch.

Understanding the basic concepts of the microcomputer led to the discussion that output lines could be selected and set to standard TTL logic levels to control drivers that would light the lights. In like fashion, a standard TTL logic level signal could be brought to the microcomputer as an input and examined. With this information, a decision could be made to vary the time delay. If the input is a "1," the lights would go on and off at a fast sequence. If the input is a "0," the sequence rate would be slow.

Obviously, other mechanical decisions also were made, such as:

1) The lights would be segments of a 7 segment light emitting diode numerical display because of the compatible packaging and ease of availability.

2) The microcomputer output pins, I/O identification and light number to 7 segment display segment were set as follows:

990/100M	*9901 I/O*	*Light No.*	*Display Segment*	*Note*
P_4 Connector				
20	P_0	1	f	
22	P_1	2	b	
14	P_2	3	e	
16	P_3	4	c	
18	P_4			to S_1

(These pin identifications are obtained from the schematics in the TM990/100M User's Guide and data sheet information on the TIL303.)

The microterminal TM990/301 was selected as the unit to use for communication with the microcomputer because of its low cost and ease of use. Terminals such as a TTY and a 743 KSR can be used and an application shown in Chapter 9 takes up this type interface.

FLOWCHARTS

The problem solution proceeds from concept to program by constructing a well defined flowchart to follow in an organized fashion while generating the sequence of steps required for the microcomputer to complete the task. *Figure 3-26* is such a flowchart of the first encounter task.

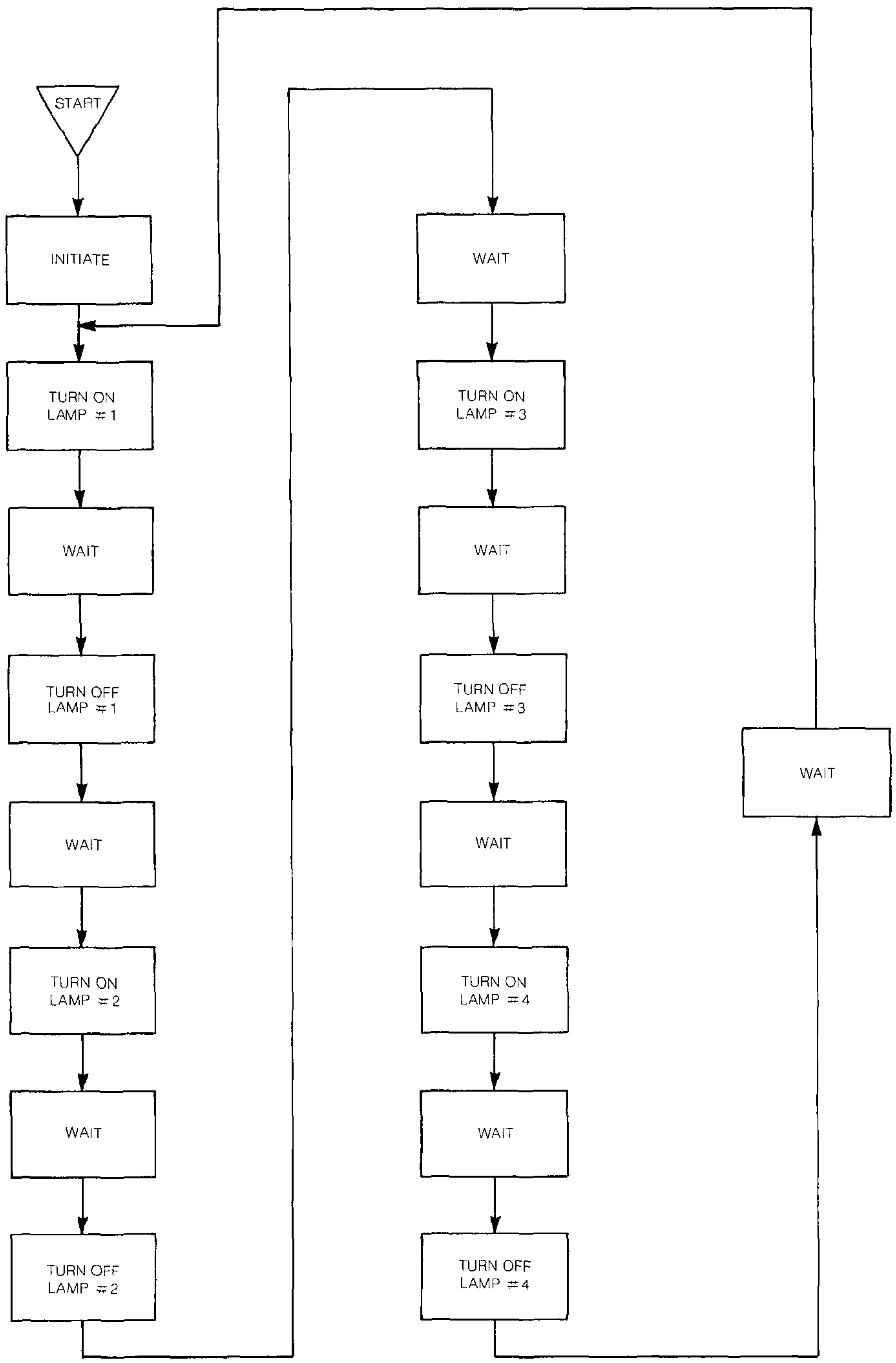


Figure 3-26. Flowchart

From START, which requires initial conditions, and a signal to begin — INITIATE — the task is diagrammed. Each light is turned on, the time delay occurs, the light is turned off, the time delay occurs, the next light is turned on, etc. The sequence continues until all lights have been turned on and off and the program begins again.

WAIT Subroutine

Note the time delay is identified as WAIT and it occurs over and over again in the sequence. Because of this, separate steps will be written for this sequence only one time rather than repeating it over and over in the program. In this manner, the main sequence of steps, the main program, can be directed to this identified set of steps, called a subroutine, by an instruction. The main program is then said to branch to the subroutine until it completes the steps in the subroutine, then it returns to the main program.

In simpler terms, the WAIT block of the flowchart requires a given number of program steps, say X. WAIT occurs 8 times in the flowchart. Instead of rewriting the X steps 8 times in the program, the X steps are written once, given the name WAIT, and referred to 8 times.

Because WAIT is a subroutine, a separate flowchart *(Figure 3-27)* is generated for it. In addition, the time delay is to be varied by the switch S_1, therefore, different steps are followed if the switch is "on" with a value of a logical "1" or "off" with a value of a logical "0". Note that when the subroutine WAIT is encountered, the first thing that occurs is to find out the position of the switch. Is it a logical "1" or a logical "0?" A decision is made on the basis of what is found. "Yes, the switch is on," (logical "1") makes the time delay short and the sequence fast. "No, the switch is off," (logical "0") makes the time delay long and the sequence slow.

There are a number of ways to provide a time delay. This flowchart uses one of the simplest — load a register with a number, keep subtracting one (decrementing) from the number until the number is zero. The number of cycles it takes to get the number to zero times the time for each cycle is the time delay. Larger numbers, longer counts, provide longer delays.

Each arm of the flowchart contains the same type of sequence, loading the number; decrementing; checking for zero; if not zero, jumping back and decrementing again; if zero, returning to the main program. Note that in the flowchart there is a branch decision and a branch decision with a jump back or a loop. The program runs in this loop until it comes to a condition where it can get out of the loop or "exit from the loop."

Subroutine Jump

Special things happen when a subroutine such as WAIT is encountered in the main program. *Figure 3-28* diagrams the steps. The main program has executed from *Step 1* to *Step 5.* At *Step 6,* the computer encounters the instruction telling it to branch to subroutine A and do subroutine A. Therefore, in order to return to the correct location in the main program after executing the subroutine, the branch instruction at *Step 6* also tells the computer to remember the address of *Step 7.*

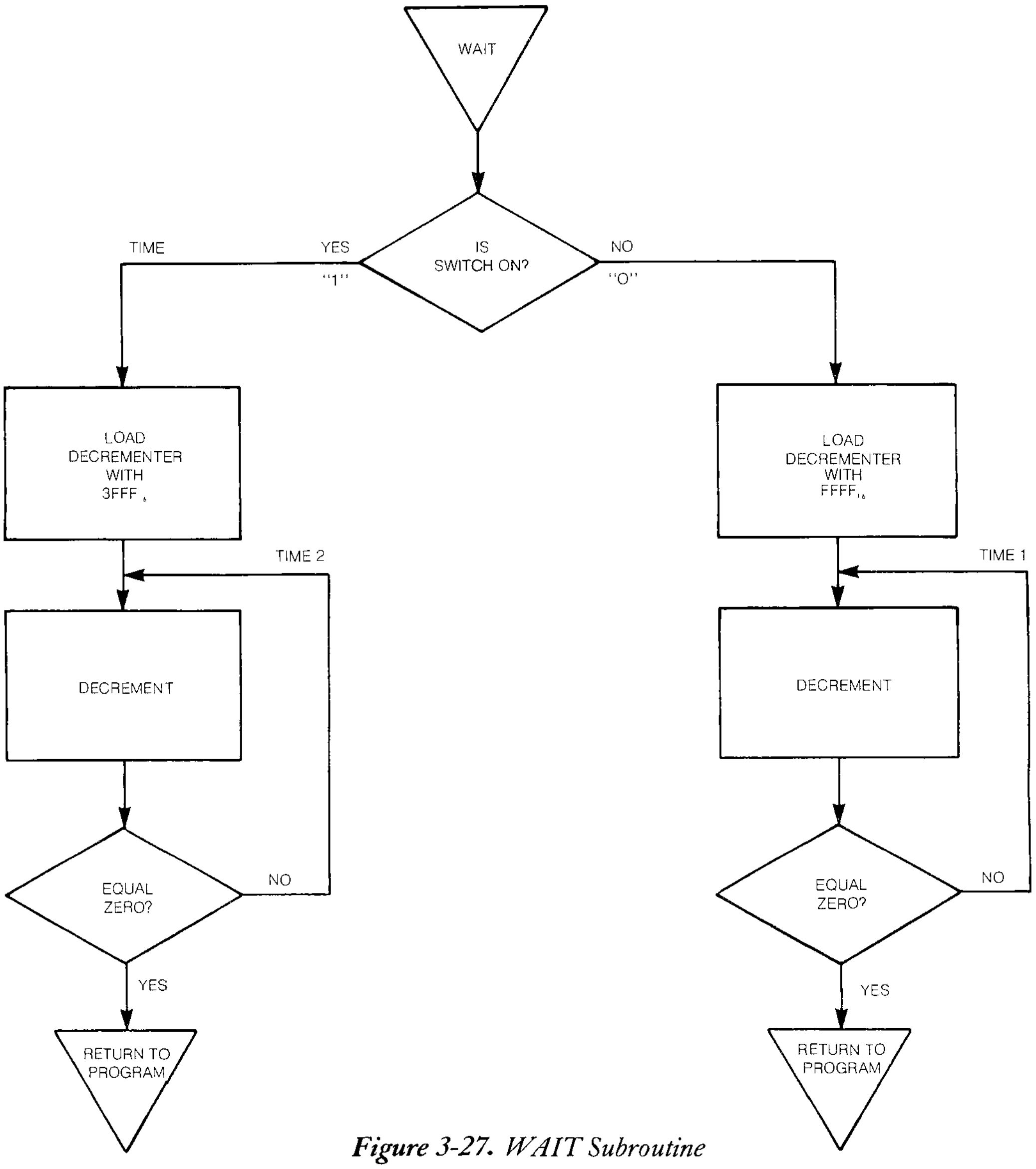


Figure 3-27. WAIT Subroutine

The subroutine is executed through *Step A-8.* Whereupon the computer encounters an instruction at *Step A-9* that tells it to return to the *Step 7* address which it remembered at *Step 6.* In this fashion, each subroutine can be executed and program control returned to the main program. Of course, there are branches that can occur from a subroutine to another subroutine but the principle is the same.

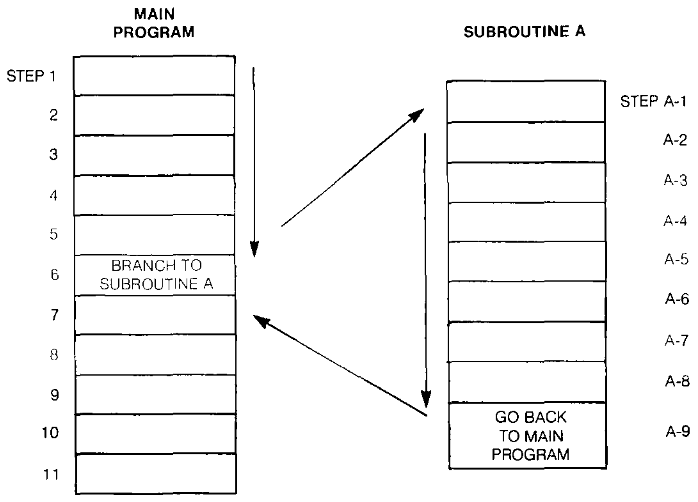


Figure 3-28. Branch to Subroutine

The instruction from the TM990/100M microcomputer instruction set that accomplishes the branch to a subroutine is called *Branch* and *Link.* This is called a "subroutine jump" instruction and will be identified by the letters BL and some additional information that tells the location of the address of the first instruction of the subroutine. In addition, recall that a register file is to be set up for general registers. Well, register 11 of this file (WR 11) is the storage place used to remember the main program address that is returned to after executing the subroutine.

The return instruction from the subroutine used is called an unconditional branch instruction. It is identified by *Branch.* Since the contents of register 11 must be returned to the program counter to return from a subroutine, this instruction will be identified as B*11. Note that the file register 11 must be reserved for this use by the programmer, otherwise its contents are likely to be changed at the wrong time and the computer misled into a wrong sequence.

A Loop Within the WAIT Subroutine

Within the WAIT subroutine is another common reoccurring concept — a loop. However, before examining this program sequence further, it would be beneficial to clearly understand the meaning of the blocks in the flow charts. The general meaning of the most commonly used blocks is shown in *Figure 3-29.* There is a symbol for the entry to or exit from a program (or for an off-page connection). This is identified with an appropriate symbol or label — START and STOP in this example. Rectangles identify operations. Inside the rectangle is an appropriate abbreviated statement to describe the operation. Decisions are identified with a diamond. Since programmed logic occurs in sequence, these blocks are relatively simple. A two-state decision answers a question of yes or no, true or false, etc. A three-state decision answers a comparison question of greater than, equal to, or less than (of course, there could be further mixtures of these). So decision blocks have appropriate questions identifying them.

In the WAIT subroutine of *Figure 3-27,* the first decision is "Is the switch ON?," and the consequences have already been discussed. The second decision has the question "The quantity examined — is it equal to zero?" Within this program sequence, if the quantity is not equal to zero, then the program goes through the same path again.

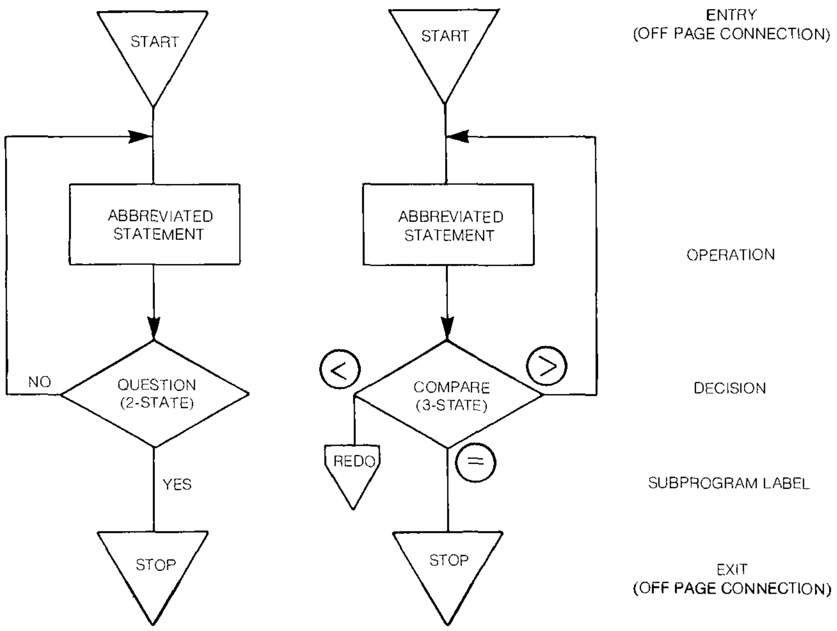


Figure 3-29. Common Flow Chart Blocks

The program loop is accomplished by a branch instruction from the instruction set called a conditional jump instruction. The conditional jump causes the microcomputer program to branch to a specified program step depending on the condition of certain bits in the status register. Recall in *Figure 3-19* that the status bits were identified and that the "equals bit" — bit 2 — was going to be used to change the time delay sequence. Therefore, the decision block in the program is really asking, "Is bit 2 of the status register set to a "1"?"

The status bit 2 is set to 1 by the program step before the decision block in *Figure 3-27* — the decrement step. An instruction *Decrement (by One)* causes a named file register to have one subtracted from its contents, comparison of the result to zero and the setting of the appropriate status bits (0-4) of the status register. When the register contents are equal to zero, the "equals" status bit (2) will be set to a "1".

When the status bit 2 is not set to a "1", the program must return to the decrement instruction and subtract one again from the register. JNE (label) is the conditional jump instruction that will be used to accomplish the loop. It is activated by the "equals bit" being "0". The program will jump to a point ahead of the decrement step which will be identified with an appropriate label. In the program this label must be included with the JNE *(Jump if not equal)* instruction.

A similar type of conditional jump instruction is used to answer the question of the switch in the first decision block of the WAIT subroutine. However, in this case, *Jump if Equal* (JEQ (label)), with the appropriate label will be used. Now the conditional jump will occur if the equal bit is set to a "1". Recall, this is the type instruction previously referred to that must follow the TB instruction so that the status bit can be examined and a decision made.

The number of steps in the decrement block is now the only remaining portion of the subroutine which has not been discussed.

LOADING A REGISTER FOR THE TIME DELAY

Assume that the switch is "ON" in the WAIT routine. A logical "1" is the input to the microcomputer. The TB instruction identifies the logical "1" and it sets the equals bit 2 of the status register to a "1" as previously described. The JEQ instruction jumps to a selected (labeled) instruction which loads a selected file register with a number, $3FFF_{16}$. As a 16 bit binary number, it is 0011 1111 1111 1111. No jump occurs in the program if the switch is inputting a logical "0". The program just proceeds to the next step.

Well, how does the data get loaded into the selected file register? Simply enough with a load instruction which is one of the data transfer instructions. *Load Immediate* (file register number), $3FFF_{16}$ will tell the microcomputer to load the hexadecimal number $3FFF_{16}$ into the selected register. What actually happens is that two memory words must be used for this instruction. The first word provides the operation code and register number and the second word the operand or data to be operated on. For the addressing mode used for the *Load Immediate* instruction, the word following the instruction LI 3, will contain the data to be put into register 3, $3FFF_{16}$. The programmer must remember that a memory word location (PC + 2) is used for the $3FFF_{16}$ data when the instruction is located at PC.

Following on then, new data is placed into the same register by a new *Load Immediate* instruction. For example, for a longer time delay, the file register R3 is loaded with $FFFF_{16}$. The instruction LI 3, $FFFF_{16}$ accomplishes this.

WHERE DOES THE PROGRAM START?

Most of the information is now in hand to write the program. The question is, "Where does the program start"? Recall that when the program was entered into the microcomputer through the microterminal, $FE00_{16}$ was chosen as the starting memory location. How was this decided?

The first step in the decision is to determine what words are available in memory — what addresses can be used.

Figure 3-30 is reproduced from the TM 990/100M Users Guide. There are address locations from 0000_{16} to $FFFE_{16}$ for 65,536 bytes (8-bit pieces), or 32,768 16-bit word locations. This is commonly called the address space. Word address locations move by an increment of 2, byte locations by 1. The incrementing of the program counter by 2 was previously noted. This is the reason.

Recall that the TM 990/100M microcomputer has 256 16-bit words of RAM into which the program is going to be placed and it also has 1024 16-bit words of ROM, or EPROM in this case. The EPROM is the TIBUG monitor that provides the necessary pre-programmed instructions that were referred to for accepting input and output data.

The 256 words of RAM occupy address space from $FE00_{16}$ to $FFFE_{16}$ as shown in *Figure 3-30.* The EPROM address space is from 0000_{16} through $07FE_{16}$ which is address space that is dedicated for this purpose and not available for change by the first encounter program. Notice that within this space are interrupt and XOP vectors. These are of no concern at this time.

Since not all the available memory sockets are filled, address space from 0800_{16} through $FDFE_{16}$ does not have memory cells — it is unpopulated.

It would seem that all the address spaces in RAM from $FE00_{16}$ to $FFFE_{16}$ are available. However, as shown in *Figure 3-30,* 40 words of RAM must be reserved for use by the TIBUG monitor and additional space is necessary for interrupts. Thus, the available space is from $FE00_{16}$ to $FF66_{16}$.

Obviously, some analysis of the possible length of the program in number of steps must be made, as well as some estimate of the number of file register blocks of 16 (workspaces) that will be used. This will determine whether adequate address space is available or whether additional memory space must be populated.

The first encounter assumptions are as follows:

▶3
1. The program will be less than 96 steps long — 96 words or 192 bytes.
2. Only one workspace will be required. (16 contiguous words)

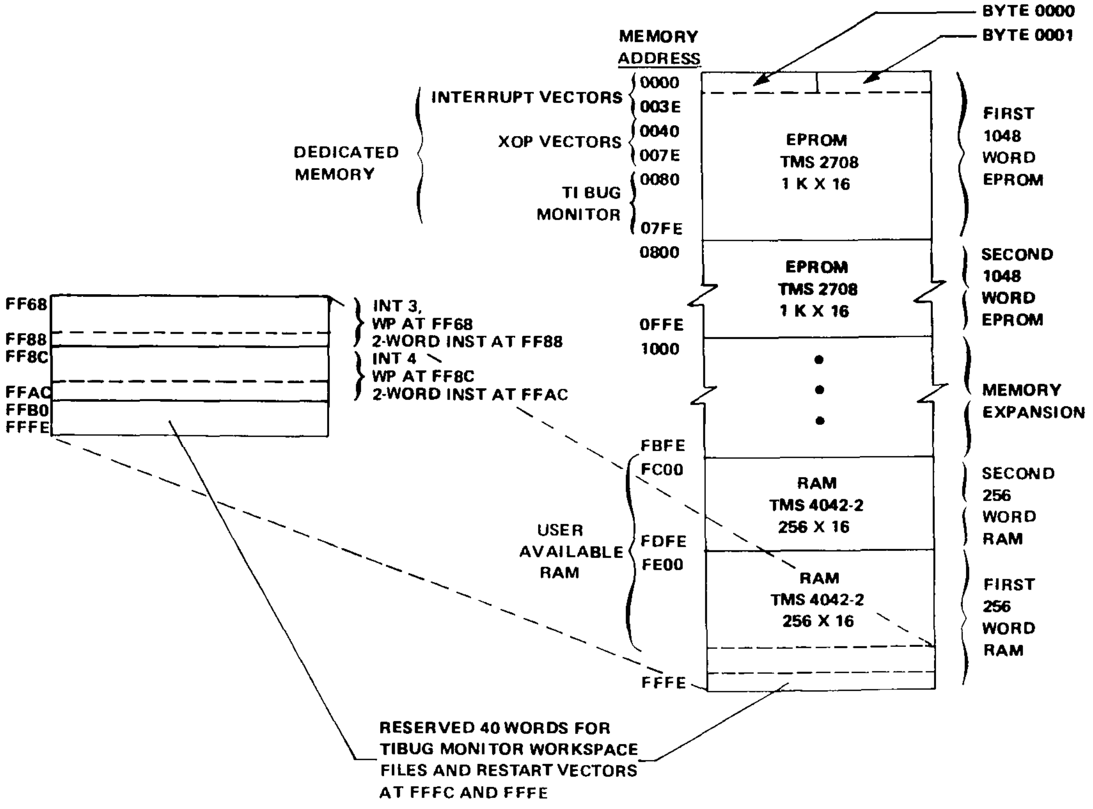


Figure 3-30. Memory Map

On this basis, the starting address of the program is chosen as $FE00_{16}$. The workspace file register could have been chosen to start at 16 words away from $FF66_{16}$. However, since there is plenty of space, it is placed at $FF20_{16}$, leaving the room from $FE00_{16}$ to $FF1E_{16}$ as the space for the program (144 words):

WRITING THE PROGRAM

Refer now to the flowchart in *Figure 3-26* as the basis for writing the program. To help in the organization of the program, a form shown in *Table 3-1* will be used. Note that it has a column for addresses, for machine code, for a label, for the assembly language statement and for comments. Each of these columns will be filled in as needed as the program is developed. Not all columns will have an entry when the program is complete. The machine code will be the last column completed. Of particular importance, especially for later references, or reference by another programmer, will be the comments column. Keep referring to *Table 3-1* after each program step to note the comments and see the program develop.

Figure 3-26 indicates that the first step in the program is to be an initializing statement. The location of the file register (workspace) used must be identified by loading the workspace pointer with the address $FF20_{16}$. The program must at all times know where the file registers are in memory for it will use these registers for obtaining data or addresses.

Reference to Chapter 5 and 6 shows there is a load instruction for the workspace pointer, LWPI, *Load Workspace Pointer Immediate.* Recall that the immediate addressing requires two words. Therefore, *Step 1* of the program at address $FE00_{16}$ is shown as:

Step	*A*	*MC*	*L*	*ASSY LANG.*
1	FE00			LWPI >FF20

and *Step 2* has the operand to be loaded. The greater than (>) sign identifies the data as hexadecimal.

The program must be able to branch to the subroutine WAIT when that routine is called by the program. Therefore, the starting address of the WAIT subroutine must be loaded into a file register which then will be referenced when the address is needed. *Step 3* of the program accomplishes this with a *Load Immediate* instruction and register 1 is chosen to hold the address. Note that the program address is incrementing by two. *Step 3* is:

Step	*A*	*MC*	*L*	*ASSY. LANG*
3	FE04			LI 1,>XXXX

Note that the specific address cannot be put in at this time — not until the location is known. *Step 4* is the step for loading the operand.

Recall that a reference needs to be established for the particular 9901 I/O interface unit to be used by the microcomputer. This was referred to as the CRU base address for the chosen 9901. Register 12 of the file register is the one that must contain the CRU base address, therefore, it must be loaded with 0120_{16}, the software base address of the 9901 in the TM990/100M microcomputer. Step 5 of the program is for this purpose.

Step	*A*	*MC*	*L*	*Assy. Lang.*
5	FE08			LI 12, >0120

Again *Step 6* must be added because of the immediate addressing.

All initial conditions are now complete and the flowchart now moves to the start of the light sequence. Light #1 must be turned on. Recall from *Figure 3-25* that light #1 is connected to I/O output 0 (P_0). Therefore, I/O-0 on the 9901 must be set to a "1". This is accomplished with the SBO instruction of *Step 7.* Recall, this instruction was previously discussed in detail. *Step 7* looks like this:

Step	*A*	*MC*	*L*	*Assy. Lang.*
7	FE0C		BEGIN	SBO 0

Note that this instruction is labeled BEGIN. This is done because the program will jump back to this address location after the complete sequence of the first encounter task is completed. The label BEGIN provides an easy reference to this location.

WAIT SUBROUTINE CALL

The first encounter task as defined now requires the light #1 be held on for the time delay represented by the subroutine WAIT. Therefore, the program must be directed to the first address of the subroutine. This first address is contained in the file register 1 (workspace register 1) because *Step 3* and *Step 4* accomplished this.

Recall the discussion on the WAIT subroutine (*Figure 3-28*). The main program must be directed to the subroutine (the main program "calls" the subroutine) but it must also remember where it is in the main program so it can return to the correct location. The *Branch and Link* to register 1 of *Step 8* accomplishes this.

Step	*A*	*MC*	*L*	*Assy. Lang.*
8	FE0E			BL *1

At the same time the address of the next step in the program, *Step 9* is being saved in register 11.

However, note that there is a new symbol in the assembly language instruction. The asterisk (*) means that an indirect addressing mode is used. That means that file register 1 (WR1) does not contain operand information but contains the *address* of an operand to be used for further processing. That is exactly what has been put into register 1 — the address of the first instruction of the WAIT subroutine. Therefore, an indirect addressing mode is used.

Why is that important? When the machine code for an instruction is constructed a little later (this will be done by hand but normally it would be done by a computer under control of a program called an assembler), an identifying code for the addressing mode must be used in the format for each instruction.

Figure 3-31 shows how the 16 bits of the machine code are arranged for the various types of instructions. Much more discussion of these formats is contained in Chapters 5 and 6. For the purpose here, format 6 is the one of particular interest for the *Branch and Link* instruction. Note that for format 6 the first 10 bits are for the operation code, bits 10 and 11 are a T_s field, and bits 12 thru 15 are an S field for identifying the address of the source information. Note that the code for T_s defines the addressing mode for the instruction. 01 will be entered in this field for bits 10 and 11 for the *Branch and Link* instruction because this is the code for indirect addressing. 0001 will be the code for the S field because register 1 contains the source address.

RETURN FROM WAIT SUBROUTINE

The end of the subroutine will return the microcomputer to the main program at *Step 9* because this is the address saved in register 11. *Step 9*, according to the flowchart of *Figure 3-26*, must now turn light #1 off. The instruction is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
9	FE10			SBZ 0

Since I/O port 0 was set to a "1" in order to turn the light on, now it is set to a "0" to turn the light off.

Time delay subroutine WAIT is called for again for the next step and again the *Branch and Link* instruction is used. Thus, *Step 10 is:*

Step	*A*	*MC*	*L*	*Assy. Lang.*
10	FE12			BL *1

Upon return from the WAIT subroutine light #2 is turned on, the WAIT routine occurs, light #2 is turned off, the WAIT routine occurs and the process continues until light #4 is turned off and the time delay is complete. These steps are shown in *Table 3-1* and carry the program through *Step 22.*

The program will return to *Step 23* after the time delay. The flowchart indicates a return to the beginning of the sequence. Recall that this was labeled BEGIN. Therefore, *Step 23* is a jump instruction that jumps the program back to the address of the instruction labeled BEGIN. The assembly language instruction is simple enough:

Step	*A*	*MC*	*L*	*Assy. Lang.*
23	FE2C			JMP BEGIN

This instruction is called an unconditional jump instruction because there are no decisions involved – just the direction to "go to" a specified place. There is no return instruction address saved in register 11 and no testing of status bits.

All the program steps in the flowchart of *Figure 3-26* are now complete. What remains is to define the steps in the subroutine WAIT. *Figure 3-27* is used for this purpose.

WAIT SUBROUTINE

The address at *Step 24,* $FE2E_{16}$, is the one that must be loaded into register 1 at *Step 3* because it is the first instruction of the subroutine. The flowchart identifies this step as a decision block. Is the switch on for a logical "1" or is it off for a logical "0"?
The input line must be tested to determine this. A TB instruction, examining I/O pin P_4, is used for this purpose. This instruction is *Step 24:*

Step	*A*	*MC*	*L*	*Assy. Lang.*
24	FE2E			TB 4

This is the *Test Bit* instruction discussed previously. Recall that when the input line is tested by the instruction it sets the "equals" bit, bit 2 of the status register to the value of the input.

In order to make the decision called for in the flowchart, an instruction that examines bit 2 of the status register must follow. This will be a conditional jump instruction because if the status bit is a "1", the time delay is to be the shortest and the sequence fast. Correspondingly, the sequence would be slow and the time delay long for a status bit 2 of "0". Chapters 5 and 6 identify the jump instructions. JEQ is the one selected which says that the program will jump to a new location if the "equals" bit is set to a "1", otherwise, the program will continue on to the next step. The instruction is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
25	FE30			JEQ TIME

Convenient labels have been placed on the flowchart of *Figure 3-27*. The branch jumped to in *Step 25* is labeled TIME. This branch will be executed in a moment. For now, assume that the "equals" bit is set to "0" and the program continues. The next step is to load a register so that it can be decremented to produce the time delay. In this branch, this must be the largest value for the longest delay and the slowest sequence. Another file register must be selected. Register 3 is chosen and the load instruction is as follows:

Step	*A*	*MC*	*L*	*Assy. Lang.*
26	FE32			LI 3, >FFFF

This is the same as previous *Load Immediate* instructions and another word must be allowed for the value to be loaded. Thus, *Step 27* at FE34.

One must now be subtracted from the value. There is an instruction called *Decrement (by one)* and, of course, it must tell what value to be decremented. In this case, the contents of R3. Thus, *Step 28* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
28	FE3		TIME1	DEC 3

The flowchart shows the decrement as an operation. In addition, as mentioned previously, *the value in register 3 is compared to zero* and the greater than, equal, carry or overflow status bits are set accordingly. This is found in the discussion on the instructions in Chapter 5 and 6.

The decision that follows is made on the basis again of examining the "equals" bit. The flow chart shows that if the "equals" bit is not set, the program will loop back and be decremented again as previously discussed. Therefore, a label, TIME 1, is placed on the instruction at FE36 to tell the program the location of the jump.

The jump occurs this time if the "equals" bit is not set, using the instruction *Jump if Not Equal,* and looks like:

Step	*A*	*MC*	*L*	*Assy. Lang.*
29	FE38			JNE TIME1

When the file register has been decremented to zero, the equals bit will be set and the program is ready to return to the main program. Recall that register 11 contains the address (location) for the return. The branch instruction used for the return is Branch and *Step 30* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
30	FE3A			B *11

Note this again is an indirect addressing mode.

TIME BRANCH

▶3 The only remaining portion of the flowchart that must be programmed is the TIME branch.

In this branch, the time delay is shorter to make the sequence faster. R3, the same register, is loaded with a smaller value, $3FFF_{16}$. Again a *Load Immediate* instruction shown in *Step 31* is used.

Step	*A*	*MC*	*L*	*Assy. Lang.*
31	FE3C		TIME	LI3, >3FFF

This step is labeled with TIME, and will be the location jumped to from *Step 25. Step 32* is the extra word required.

The register must again be decremented, therefore, the instruction is the same type as *Step 28.* However, the label for the location to jump to is now TIME2. *Step 33* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
33	FE40		TIME2	DEC 3

The same jump instruction is used in this branch as for *Step 29* except the label is now TIME 2. Therefore, *Step 34* is:

Step	*A*	*MC*	*L*	*Assy. Lang.*
34	FE42			JNE TIME2

When the equals bit is set, the program must return to the main program as with the other branch. The same return instruction as *Step 30* is used, as shown in *Step 35.*

Step	*A*	*MC*	*L*	*Assy. Lang.*
35	FE44			B *11

The total program is now complete in assembly language. It is shown in *Table 3-1.*

Table 3-1. Assembly Language Program.

(Source Code Statements)

Step	Hex Address	Hex Machine Code	Label	Op Code	Operand	Comments.
1.	FE00			LWPI	>FF20	Load workspace pointer
2.	FE02					with $FF20_{16}$
3.	FE04			LI	1, >FF2E	Load R1
4.	FE06					with 1st Address of WAIT
5.	FE08			LI	12, >0102	Load R12
6.	FE0A					with base address of 9901, 0120_{16}
7.	FE0C		BEGIN	SBO	0	Set I/O P_0 (segment f) equal to one
8.	FE0E			BL	*1	Branch to address in R1 (saves next address in R11)
9.	FE10			SBZ	0	Set I/O P_0 (segment f) equal to zero
10.	FE12			BL	*1	Branch to address in R1 (saves next address in R11)
11.	FE14			SBO	1	Set I/O P_1 (segment b) equal to one
12.	FE16			BL	*1	Branch to address in R1
13.	FE18			SBZ	1	Set I/O P_1 equal to zero
14.	FE1A			BL	*1	Branch to address in R1
15.	FE1C			SBO	2	Set I/O P_2 (segment e) equal to one
16.	FE1E			BL	*1	Branch to address in R1
17.	FE20			SBZ	2	Set I/O P_2 equal to zero
18.	FE22			BL	*1	Branch to address in R1
19.	FE24			SBO	3	Set I/O P_3 (segment c) equal to one
20.	FE26			BL	*1	Branch to address in R1
21.	FE28			SBZ	3	Set I/O P_3 to equal to zero
22.	FE2A			BL	*1	Branch to address in R1
23.	FE2C			JMP	BEGIN	Jump to BEGIN
24.	FE2E		WAIT	TB	4	Test I/O P_4 for a "1" or a "0"
25.	FE30			JEQ	TIME	If equals bit is set ("1"), jump to TIME
26.	FE32			LI	3, >FFFF	Load R3
27.	FE34					with $FFFF_{16}$
28.	FE36		TIME1	DEC	3	Decrement R3
29.	FE38			JNE	TIME1	Jump to TIME 1 if equals bit is not set
30.	FE3A			B	*11	Return to main program (by way of R11)
31.	FE3C		TIME	LI	3, >3FFF	Load R3
32.	FE3E					with $3FFF_{16}$
33.	FE40		TIME2	DEC	3	Decrement R3
34.	FE42			JNE	TIME2	Jump to TIME 2 if equals bit is not set
35.	FE44			B	*11	Return to main program (by way of R11)

WRITING THE MACHINE CODE

Normally the next step in programming (shown in *Table 3-2*) would be done by a computer as mentioned previously. However, in order to demonstrate what an Assembler Program would do and because the program input to the TM990/100M microcomputer is through the microterminal, which requires the machine code, it will be a good exercise to demonstrate how to develop the machine code. If this is of no interest, this portion of the discussion can be bypassed and a jump made to the summary.

As mentioned previously in *Figure 3-31,* there is a set format for the 16 bits of machine code that must be generated for each instruction. The formats used for the first encounter task are shown in *Figure 3-32* for reference. Each instruction has an operation code (OP CODE) and then additional information is required in the various fields of the format. A complete discussion of the format for each instruction can be found in Chapter 6. *Figure 3-33* lists the instructions used in the first encounter.

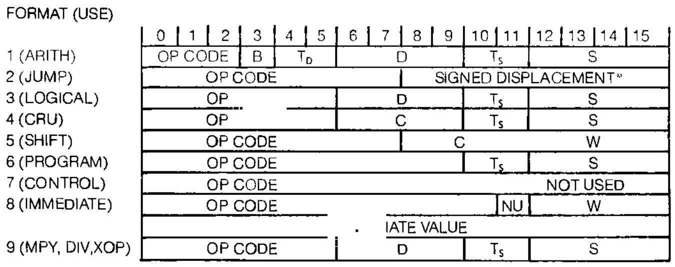
The same programming form will be used as before which is summarized to this point in *Table 3-1.* The machine code will be filled in and several other changes made and the result will be the final program of *Table 3-2.* As before, continue to refer to *Table 3-2* as the machine code is developed.

Immediate Instructions

The coding begins at *Step 1.* LWPI is an immediate instruction. Therefore, the format 8 of *Figure 3-32* is used. There are two words to this instruction; the second one containing the immediate value to be loaded. In the first word, the op code occupies bits 0 through 10; register numbers, where the immediate value is going to be placed, occupy bits 12 thru 15. Bit 11 is not used. The op code is obtained from *Figure 3-33* for the LWPI instruction. The filled out instruction would look like this.

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Binary —	0 0 0 0	0 0 1 0	1 1 1 0	0 0 0 0
Op Code —	0	2	E	0
Machine Code —	0	2	E	0

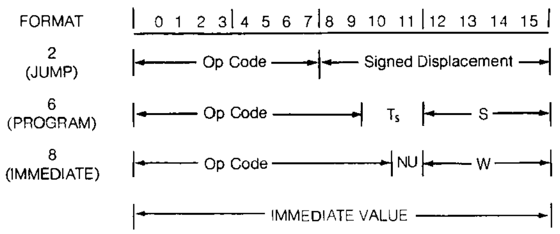
LWPI is a special case of format 8. Bits 11-15 are not used and as such could contain anything. They are don't care conditions. Therefore, the machine code is 02E0. This is entered into *Table 3-2* on the same line as LWPI as *Step 1. Step 2* is the immediate value FF20, therefore, the machine code is $FF20_{16}$.


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

KEY

B = BYTE INDICATOR (1 = BYTE, 0 = WORD)	S = SOURCE ADDR
	C = XFR OR SHIFT LENGTH (COUNT)
T_D = D ADDR. MODIFICATION	W = WORKSPACE REGISTER NO.
D = DESTINATION ADDR.	* = SIGNED DISPLACEMENT OF − 128 TO + 127 WORDS
T_S = S ADDR. MODIFICATION	NU = NOT USED

Figure 3-31. Instruction Formats



NOTES:

T_s = SOURCE ADDRESS MODIFICATION
S = SOURCE ADDRESS
W = WORKSPACE (FILE) REGISTER NO.
NU = NOT USED

SIGNED DISPLACEMENT CAN BE
− 128 TO + 127 WORDS

CODES FOR T_S FIELD	**ADDRESSING MODE**
00	REGISTER
01	INDIRECT
10	INDEXED (S OR D≠0)
10	SYMBOLIC (DIRECT, S OR D = 0)
11	INDIRECT WITH AUTO INCREMENT

Figure 3-32. Formats used for First Encounter

In like fashion, the instructions at *Step 3* and *Step 5* are immediate instructions, use the same format, and are coded with the appropriate register numbers. *Step 4* and *Step 6* are the immediate values to be loaded.

Note, however, that when the program was first prepared, the first address of the WAIT subroutine was not known. Now, it is known. It is substituted for the XXXX in *Table 3-1* at Step 3. Thus, the address of *Step 24,* FE2E is placed after the "greater than" symbol.

The op code for LI is 0200_{16} and since register 1 is used for *Step 3,* the machine code is 0201_{16} while for *Step 5* it is 020C because register 12 is being loaded. The machine code for *Step 4* is the value $FE2E_{16}$ and for *Step 6* it is 0120_{16}.

INSTRUCTIONS SBO, SBZ

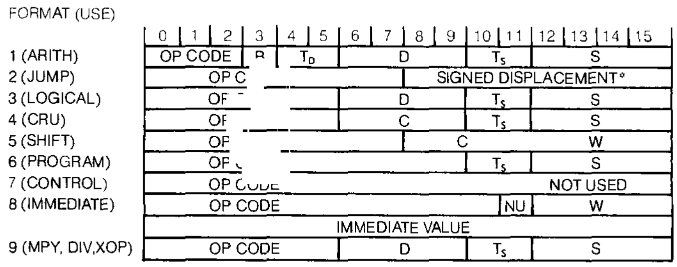
The instruction SBO at *Step 7* uses a different format. This is format 2 in *Figure 3-32.* It has the op code in bits 0 through 7 and the signed displacement that was discussed previously when the 9901 I/O unit program was examined. Recall that the CRU base address was arranged so that the bit number is the value that is put in for the signed displacement.

The op code for SBO from *Figure 3-33* is $1D00_{16}$ and with the first bit being zero, the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Binary —	0 0 0 1	1 1 0 1	0 0 0 0	0 0 0 0
Op Code —	1	D	0	0
Machine Code —	1	D	0	0

MNEMONIC	HEX OP CODE	FORMAT	RESULT COMPARE TO ZERO	INSTRUCTION
LWPI	02E0	8	N	LOAD IMMEDIATE TO WORKSPACE POINTER
LI	0200	8	N	LOAD IMMEDIATE
BL	0680	6	N	BRANCH AND LINK (WR11)
B	0440	6	N	BRANCH
DEC	0600	6	Y	DECREMENT (BY ONE)
SBO	1D00	2	N	SET CRU BIT TO ONE
SBZ	1E00	2	N	SET CRU BIT TO ZERO
TB	1F00	2	N	TEST CRU BIT
JEQ	1300	2	N	JUMP EQUAL (ST2 = 1)
JMP	1000	2	N	JUMP UNCONDITIONAL
JNE	1600	2	N	JUMP NOT EQUAL (ST2 = 0)

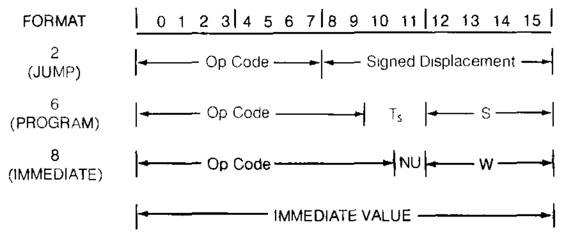
Figure 3-33. Instructions used for First Encounter.



KEY

B = BYTE INDICATOR (1 = BYTE, 0 = WORD)	S = SOURCE ADDR
T_D = D ADDR. MODIFICATION	C = XFR OR SHIFT LENGTH (COUNT)
D = DESTINATION ADDR.	W = WORKSPACE REGISTER NO.
T_S = S ADDR. MODIFICATION	* = SIGNED DISPLACEMENT OF − 128 TO + 127 WORDS
	NU = NOT USED

Figure 3-31. Instruction Formats



NOTES:

T_S	= SOURCE ADDRESS MODIFICATION
S	= SOURCE ADDRESS
W	= WORKSPACE (FILE) REGISTER NO.
NU	= NOT USED

SIGNED DISPLACEMENT CAN BE
− 128 TO + 127 WORDS

CODES FOR T_S FIELD	**ADDRESSING MODE**
00	REGISTER
01	INDIRECT
10	INDEXED (S OR D≠0)
10	SYMBOLIC (DIRECT, S OR D = 0)
11	INDIRECT WITH AUTO INCREMENT

Figure 3-32. Formats used for First Encounter

In like fashion, the instructions at *Step 3* and *Step 5* are immediate instructions, use the same format, and are coded with the appropriate register numbers. *Step 4* and *Step 6* are the immediate values to be loaded.

Note, however, that when the program was first prepared, the first address of the WAIT subroutine was not known. Now, it is known. It is substituted for the XXXX in *Table 3-1* at Step 3. Thus, the address of *Step 24,* FE2E is placed after the "greater than" symbol.

The op code for LI is 0200$_{16}$ and since register 1 is used for *Step 3,* the machine code is 0201$_{16}$ while for *Step 5* it is 020C because register 12 is being loaded. The machine code for *Step 4* is the value FE2E$_{16}$ and for *Step 6* it is 0120$_{16}$.

Instructions SBO, SBZ

The instruction SBO at *Step 7* uses a different format. This is format 2 in *Figure 3-32.* It has the op code in bits 0 through 7 and the signed displacement that was discussed previously when the 9901 I/O unit program was examined. Recall that the CRU base address was arranged so that the bit number is the value that is put in for the signed displacement.

The op code for SBO from *Figure 3-33* is 1D00$_{16}$ and with the first bit being zero, the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Binary —	0 0 0 1	1 1 0 1	0 0 0 0	0 0 0 0
Op Code —	1	D	0	0
Machine Code —	1	D	0	0

MNEMONIC	HEX OP COOE	FORMAT	RESULT COMPARE TO ZERO	INSTRUCTION
LWPI	02E0	8	N	LOAO IMMEOIATE TO WORKSPACE POINTER
LI	0200	8	N	LOAO IMMEDIATE
BL	0680	6	N	BRANCH ANO LINK (WR11)
B	0440	6	N	BRANCH
OEC	0600	6	Y	OECREMENT (BY ONE)
SBO	1000	2	N	SET CRU BIT TO ONE
SBZ	1E00	2	N	SET CRU BIT TO ZERO
TB	1F00	2	N	TEST CRU BIT
JEQ	1300	2	N	JUMP EQUAL (ST2 = 1)
JMP	1000	2	N	JUMP UNCONDITIONAL
JNE	1600	2	N	JUMP NOT EQUAL (ST2 = 0)

Figure 3-33. Instructions used for First Encounter.

The other SBO instructions can be machine coded accordingly using the appropriate bit number. Therefore, *Step 11* is $1D01_{16}$, *Step 15* is $1D02_{16}$ and *Step 19* is $1D03_{16}$.

Similarly, using the op code of $1E00_{16}$ for the SBZ instructions and the appropriate bit number, *Step 9* is $1E00_{16}$, *Step 13* is $1E01_{16}$, *Step 17* is $1E02_{16}$, and *Step 21* is $1E03_{16}$.

Instruction BL

Now *Step 8* brings in another new format. For the BL instruction, it is format 6. Bits 0 thru 9 contain the op code. Bits 12 through 15 are the address of the source data. Ts is a field that modifies the source address and it contains the two bits that code the addressing mode that is being used. Recall BL *1 uses indirect addressing. Therefore, from *Figure 3-32* Ts would be 01 for these 2 bits. It's important to remember that this modifies the op code into a different number for the machine code as shown below.

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Op Code	—	0				6				8				0			
Binary	—	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0
T_S	—											0	1				
S	—													0	0	0	1
Machine Code—(Binary)	—	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	1
Machine Code—(Hex)	—	0				6				9				1			

Thus, the machine code is 0691_{16} and can be placed in *Step 8, 10, 12, 14, 16, 18, 20* and *22,* since register 1 is used in each case.

Miscellaneous Instructions

Because the jump instructions fall into a class that needs special discussion, the remaining instructions will be coded first.

Step 26 and *Step 31* are LI instructions like Step 3 and Step 5 — the code is 0203_{16} in this case because register 3 is being used. Don't forget the values of $FFFF_{16}$ for *Step 27* and $3FFF_{16}$ for *Step 32.*

The TB instruction has an op code of $1F00_{16}$ and a format 2. It is just like the SBO and SBZ so that the bit must be used for the displacement. Bit 4 causes a displacement of 4, therefore, the machine code is $1F04_{16}$. This is *Step 24.*

A branch instruction similar to BL, but does not save the next address in register 11, is the instruction B. It is using the contents of register 11 for a return to the main program. The op code for B is 0440_{16}. It uses an indirect addressing mode so $T_S = 01$ and S is 1011 for register 11. The machine code results as follows:

		0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code	—	0	4	4	0
		0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0
T_S	—			0 1	
S	—				1 0 1 1
Machine Code (Binary)	—	0 0 0 0	0 1 0 0	0 1 0 1	1 0 1 1
Machine Code (Hex)	—	0	4	5	B

It is entered at *Step 30* and *35*.

The only remaining instruction besides the jump instructions is the decrement instruction DEC. From *Figure 3-33* the op code is 0600_{16} and the format is 6. Register 3 is being used, therefore, S is 0011. The addressing mode is a register mode so T_S is 00 and there is no modification of the op code. The machine code is then 0603_{16} for *Step 28* and *33*.

Jump Instructions

Jump instructions use format 2 of *Figure 3-32* which has an op code for bits 0 through 7 and a signed displacement in bits 8 through 15. The signed displacement means the number of program addresses that the program must move to arrive at the required address. For example, let

A_J = present address of jump instruction
A_D = destination address of jump instruction

then,

1.) $A_J + 2\ \text{DISP} = A_D$

since the program moves by increments of 2.

However, for the 9900 microprocessor in the TM990/100M microcomputer, the jump instruction signed displacement must be calculated from the address following the address of the jump instruction or $A_J + 2$. Therefore, equation (1) becomes,

2.) $(A_J + 2) + 2\ \text{DISP} = A_D$

Solving for DISP, gives

3.) $\dfrac{A_D - (A_J + 2)}{2} = \text{DISP}$

Recall that in preparing the program of *Table 3-1* labels were used for instructions so that easy reference could be made to the desired destination address for a jump instruction. *Step 23* at $FE2C_{16}$ is the first jump instruction. The destination is the label BEGIN which is located at address $FE0C_{16}$. Applying equation (3) gives (in Hex)

4.) $\text{DISP} = \dfrac{FE0C - (FE2C + 2)}{2}$

5.) $\text{DISP} = \dfrac{FE0C - FE2E}{2}$

Now,

$$\begin{array}{r} FE0C \\ -\underline{FE2E} \\ -0022_{16} \end{array}$$

Therefore,

6.) $\text{DISP} = -\dfrac{22}{2} = -11_{16}$

This means that in the jump instruction the program moves back 11_{16} steps or 17 decimal steps.

Now, since this is a negative number, a two's complement must be used for the code, thus

	−0011
COMPLEMENT	FFEE
ADD ONE	+0001
2'S COMPLEMENT	FFEF

Now, only the 8 least significant bits are used along with the op code of *Figure 3-33.* JMP of *Step 23* has an op code of 1000_{16}. Therefore, the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	0	0	0
Displacement —			E	F
Machine Code —	1	0	E	F

This machine code is entered at *Step 23.*

Step 25 has a JEQ instruction. A_J is $FE30_{16}$. The instruction says to jump to TIME which has an address of FE3C at *Step 31,* therefore, $A_D = FE3C$. Applying equation (3) gives, again in hexadecimal;

$$DISP = \frac{FE3C - FE32}{2} = \frac{10}{2} = 5_{16}$$

▶3

JEQ has an op code of 1300 and the machine code then becomes:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	3	0	0
Displacement —			0	5
Machine Code—	1	3	0	5

Step 25 then has 1305 as the machine code.

The remaining jump instructions, JNE at *Steps 29* and *34* have an op code of 1600_{16}. Calculating the displacement from *Step 29* to *Step 28* and from *Step 34* to *Step 33,* obviously is -02_{16}. The complement of -02 is FFFD and the twos complement is FFFE. Thus the machine code is:

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Op Code —	1	6	0	0
Displacement —			F	E
Machine Code—	1	6	F	E

Even though the labels jumped to for *Steps 29* and *34* are different, the displacement is the same and, therefore, the machine code entered at these steps is the same, $16FE_{16}$.

TABLE 3-2

Every step is now coded and the program is complete. This is the program that was entered into the microcomputer via the terminal to accomplish the first encounter task.

Only one comment remains. If the *Table 3-1* program were run on a computer under the direction of an assembler program, certain symbols used for directives to the assembler would have to be used. The $ symbol could have been used to indicate the fact that a displacement is to be made from the jump instruction address which was identified in equation (3) as A_J. The instruction then would contain the $ symbol followed by the necessary displacement in hexadecimal. For this reason the instructions at *Step 23, 25, 29,* and *34* would have looked as follows:

Step	*A*	*MC*	*L*	*Assy. Lang.*
23	FE2C	10EF		JMP $–17
25	FE30	1305		JEQ $+5
29	FE38	16FE		JNE $–2
34	FE42	16FE		JNE $–2

SUMMARY

It has been a long discussion. However, a great deal of material has been covered and many basic concepts developed. The facts and procedures presented should provide a solid foundation for expanding an understanding of the 9900 Family of microprocessors and microcomputer component peripherals and the microcomputers which use it. Hopefully, enough examples have been presented with the first encounter task that with a minimum of effort, new real applications of the TM990/100M board can be implemented. A few simple ones that can be implemented immediately with the present setup would be:

A. Wire-up the necessary drivers and resistors to drive all seven-segments of the display and write a new program to make the numbers 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 come up in sequence.

B. Write a program that uses the 7 segment display numbers so that they spell a word when read up-side down.

Maybe more memory will be required, but that is easy to add to the TM990/100M.

The next step is to implement the logic levels at the output pins into real applications of controlling dc and ac voltages for control applications. An extended application in Chapter 9 using this same TM990/100M board setup shows how this can be done. Persons interested can follow right into this application example to gain more insight into the details of the 9900 family of components explained in detail in the following chapters.

Table 3-2. Assembly Language Program.

(With Machine Code)

Step	Hex Address	Hex Machine Code	Label	Op Code	Operand	Comments.
1.	FE00	02E0		LWPI	>FF20	Load workspace pointer
2.	FE02	FF20				with $FF20_{16}$
3.	FE04	0201		LI	1, >FE2E	Load R1
4.	FE06	FE2E				with 1st address of WAIT
5.	FE08	020C		LI	12, >0102	Load 12
6.	FE0A	0120				Will base address of 9901, 0120_{16}
7.	FE0C	1D00	BEGIN	SBO	0	Set I/O P_0 (segment f) equal to one
8.	FE0E	0691		BL	*1	Branch to address in R1, (saves next address in R11)
9.	FE10	1E00		SBZ	0	Set I/O P_0 (segment f) equal to zero
10.	FE12	0691		BL	*1	Branch to address in R1 (saves next address in R11)
11.	FE14	1D01		SBO	1	Set I/O P_1 (segment b) equal to one
12.	FE16	0691		BL	*1	Branch to address in R1
13.	FE18	1E01		SBZ	1	Set I/O P_1 equal to zero
14.	FE1A	0691		BL	*1	Branch to address in R1
15.	FE1C	1D02		SBO	2	Set I/O P_2 (segment e) to one
16.	FE1E	0691		BL	*1	Branch to address in R1
17.	FE20	1E02		SBZ	2	Set I/O P_2 equal to zero
18.	FE22	0691		BL	*1	Branch to address in R1
19	FE24	1D03		SBO	3	Set I/O P_3 (segment c) equal to one
20.	FE26	0691		BL	*1	Branch to address in R1
21.	FE28	1E03		SBZ	3	Set I/O P_3 equal to zero
22.	FE2A	0691		BL	*1	Branch to address in R1
23.	FE2C	10EF		JMP	BEGIN	Jump to BEGIN
24.	FE2E	1F04	WAIT	TB	4	Test I/O P_4 for a "1" or a "0"
25.	FE30	1305		JEQ	TIME	If equals bit is set ("1"), jump to TIME
26.	FE32	0203		LI	3, >FFFF	Load R3
27.	FE34	FFFF				with $FFFF_{16}$
28.	FE36	0603	TIME1	DEC	3	Decrement R3
29.	FE38	16FE		JNE	TIME1	Jump to TIME1 if equals bit is not set
30.	FE3A	045B		B	*11	Return to main program (by way of 11)
31.	FE3C	0203	TIME	LI	3, >3FFF	Load R3
32.	FE3E	3FFF				with $3FFF_{16}$
33.	FE40	0603	TIME2	DEC	3	Decrement R3
34.	FE42	16FE		JNE	TIME2	Jump to TIME2 if equals bit is not set
35.	FE44	045B		B	*11	Return to main program (by way of R11)