CHAPTER 9
Applications

This chapter is devoted to examples of applications of the 9900 family of components. Throughout this book many details of the 9900 family of CPU's, peripherals, microcomputer modules, software and software development system support have been discussed. However, these have been somewhat isolated general discussions and not directed to a particular application. This chapter has solutions of specific problems — from the beginning concept to the final machine code — to give you examples of how someone else has approached the problem and to help you understand the concepts behind the approach and the details of the solution.

Three applications are included. They are:

1. A SIMULATED INDUSTRIAL CONTROL APPLICATION

   A 9900 microprocessor based microcomputer is used in a system simulating the control of industrial manufacturing processes. Solutions to the problems of interfacing between industrial power levels and computer logic levels, both at the input and the output, are demonstrated, as well as basic concepts of computer control.

2. A LOW-COST DATA TERMINAL

   Direct comparison is made showing how the characteristics of the 9940 single chip 16-bit microcomputer are used to significantly reduce the package count of an intelligent terminal designed with an 8080 8-bit processor. At the same time the performance-cost ratio of the end equipment is improved.

3. A FLOPPY DISK CONTROLLER

   The design of a complex system used for the control of a floppy diskette memory is described. All the details of how a 9900 family microprocessor is used to arrive at a problem solution are included.

A Simulated Industrial Control Application

9

INTRODUCTION

Controlling motors, relays, solenoids, actuators; sensing limit switches, photo-electric outputs, push-button switches are real world problems encountered in controlling industrial manufacturing. This application simulates such conditions. It develops the application of a TMS9900 microprocessor (using the 990/100M microcomputer module of Chapter 3) and interconnecting hardware to automating industrial control requirements. This example includes the description of interface hardware to couple industrial power levels to and from the microcomputer system. It illustrates the use of an EIA/TTY terminal for interactive program entry and control, a line-by-line assembler for inexpensive program assembly, and the techniques of interrupt driven processing.

No motors, actuators, or solenoids are actually being controlled, but by sensing switches for logical voltage inputs and by turning lights on and off, the industrial control inputs and loads are simulated and the means demonstrated to accomplish the control.

As a logical extension of the first encounter application of Chapter 3, this application is written for "hands-on" operation to develop basic concepts and show that the 9900 family of microprocessors is ideally suited for industrial control applications. Each program step is described as the subprograms are developed and the total program is assembled into machine code.

Excitement comes from actually getting a microprocessor system doing useful things. This application is designed for that purpose. Let it demonstrate how easy it is to begin applying the 9900 family of microprocessors.

INITIAL SYSTEM SETUP WITH AN EIA TERMINAL

To begin, look at *Figures 1* and *6.* The system uses the same TM990/100M-1 microcomputer module shown in *Figure 3-12* and interconnected in *Figure 3-14.* It is a complete microcomputer with 256 16-bit words of RAM, 1024 16-bit words of ROM, and interface circuits to handle parallel and serial I/O. In *Figure 3-14* it has power supplied to it through P1, the 100 pin edge connector as specified in *Figure 3-17.* P2 interconnects the TM990/301 microterminal which is being used as an input terminal for programming, editing, and debugging. The output board *(Figure 3-9)* with a 7 segment LED display is connected to the microcomputer through P4. The program *(Table 3-2)* sequenced the elements f, b, e and c of the LED display on and off, either fast or slow, depending on the position of the control switch.

Table 3-2 was "assembled-by-hand." In the examples that follow, a ROM resident "line-by-line" assembler will be used. This is a low-cost, effective way of providing machine code. However, a different terminal is required so that print out of the code can be obtained. Therefore, in this application the microterminal attached to the TM990/100M microcomputer is replaced with a keyboard terminal with EIA/TTY interconnection. Refer to *Figure 1.*

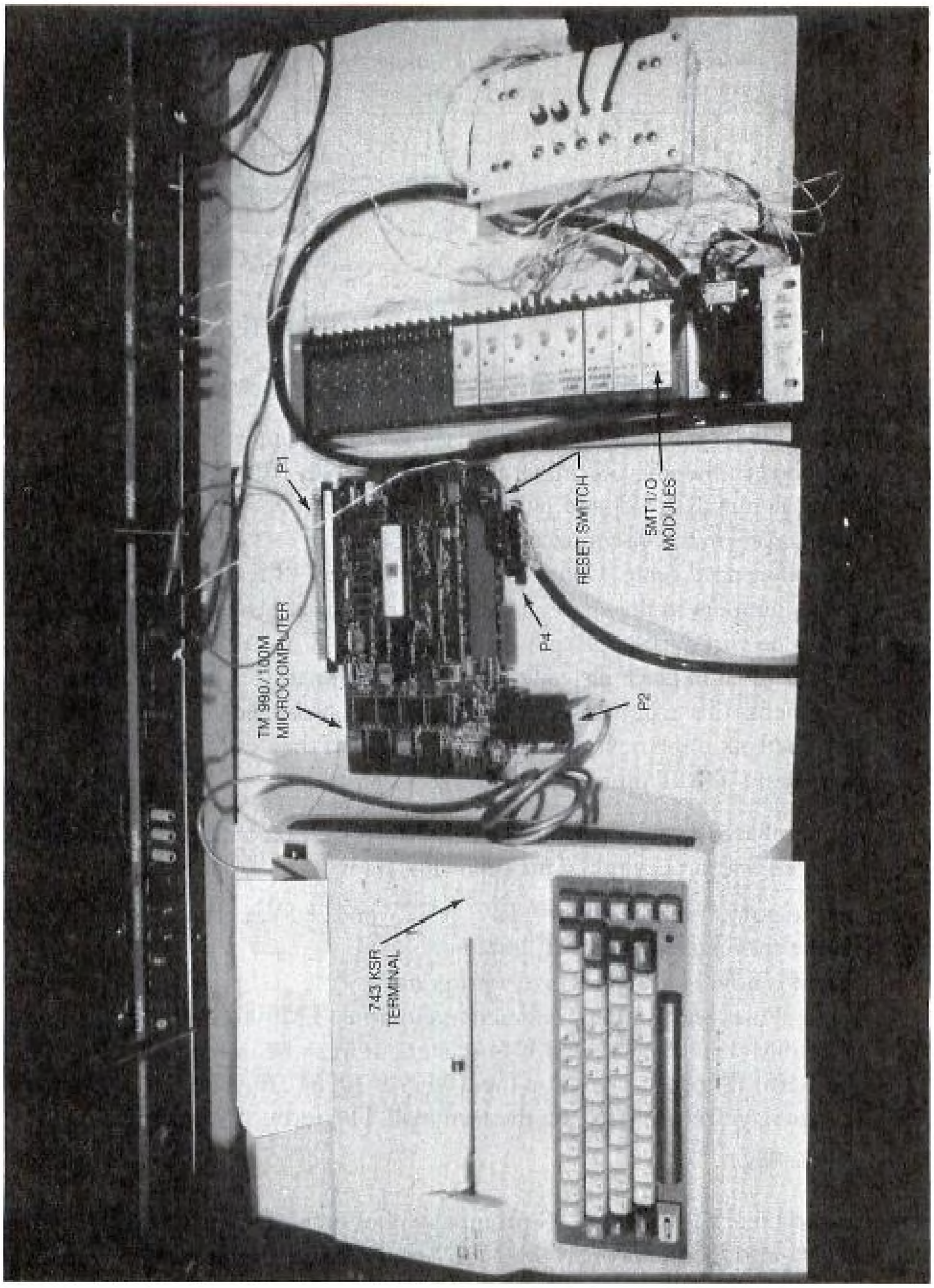


Figure 1. Picture of System Set-up

A 743 KSR terminal is chosen for this purpose. A special cable is required to interface the terminal to the microcomputer through P2. The cable connections are as follows:

TM 990/100M-1 P2 Pin	*743 Terminal P1 Pin*	*Description*
1	9	Protective Gnd
2	13	Transmit data
3	12	Receive data
7	1	Signal Gnd
8	11	Request to send
20	15	Data Terminal Ready

If a preassembled cable is desired, a TM990/503 can be purchased for the purpose.

If the TM990/100M-1 microcomputer was used for the Chapter 3 First Encounter, power was supplied to the microterminal from the TM990/100M module by jumpers installed across the pins J13, J14 and J15 *(Figure 3-12* and *3-13).* These should now be removed; the microterminal disconnected from P2; and the 743 KSR terminal connected to P2 with the referenced cable. Connect ac power to the 743 terminal with a separate cord. Return the jumpers to the spare positions on the board J16, J17, and J18 *(Figure 3-13).* If P1 is to be wired to supply power, use *Figure 3-17* for the connections. *Figure 1* shows the 743 terminal in place instead of the microterminal. It also shows the I/O interface components that will be used for this application connected to P4. If familiar with a 743 terminal, skip the next discussion and go on to the description of the I/O interface components (5MT interface modules).

For those not familiar with the operation of a 743 terminal, reconnect the output board of *Figure 3-9* to P4 and proceed thru the following steps:

1. Turn on the power supplies, the − 12V, + 12V and + 5V, in that order.
2. Turn on the terminal and place it "on line."
3. The system is now ready to receive a program.
4. The terminal uses the TIBUG interactive monitor (TM990/401-1) resident on the TM990/100M-1 in the U42 and U44 sockets. It must be initialized. To do this, press the RESET toggle switch on the TM990/100M *(Figure 1)* and the character "A" or a carriage return (CR) on the terminal. The terminal responds:

```
TIBUG REV.A
?
```

5. The question mark is the TIBUG prompt symbol saying "what's next?" To enter code or data into memory, press the M (Memory Inspect and Change) command key followed by the address in Memory where the program or routine is to start followed by a (CR). The terminal printout looks like this:

```
?M  FE00 (CR)
```

6. TIBUG responds with the address and the data located at that address such as:

   ```
   FE00=ABCD
   ```

   If the data is not correct and is to be changed, type in the correct data and press either of these options:

   A. (CR) to return to TIBUG
   B. The space bar to increment to the next memory word location.
   C. A minus (−) character to return to the previous word location.

The complete sequence is illustrated here:

```
?M FE00  [CR]
FE00=ABCD        02E0  [Space]
FE02=3D04        FF20  [Space]
FE04=FC36          —   [minus]*
FE02=FF20              [Space]
FE04=FC36        0201  [Space]
FE06=0032                 [CR]
?                      *requires pressing "NUM" key
```

7. After an M and the starting address FE00 and a (CR), the total program of *Table 3-2*, should be entered by entering the correct machine code at each address and then pressing the space bar. At the end of the program, exit the memory inspect and change mode by pressing (CR). The terminal responds with the familiar "?". If an error occurs, press (CR), then M and the address at which the error occurred; then repeat the input code.

8. Now the program is ready to run. However, the workspace pointer and the program counter may have to be set; at least the program counter, because it controls where the program starts. The register inspect and change command R is pressed. TIBUG responds with the contents of the workspace pointer. Press the space bar and TIBUG comes back with the program counter contents. Either of these can be changed in the same manner as memory.

   Change the contents of the PC to the first address of the program to be run, then type a (CR) and the program is ready to be executed. The total routine looks like this:

   ```
   ?R
   W=0020 [Space]
   P=0846 FE00 [CR]
   ?
   ```

The program counter is now set at the starting address of the program of *Table 3-2*, FE00. Usually as the program proceeds, it will set the workspace pointer as needed; thus, no change is made to W in the above routine.

9◀

9. The Execute Command, E, runs the program:.

```
?E
```

It runs until the RESET switch is pressed. After RESET, the program counter must be reset to FE00. This is done with a (CR), then R, then (Space), then FE00, then (CR), then E to start again.

The necessary details of interfacing and operating the 743 KSR have now been covered. Further information on commands may be obtained by referring to the TM990/100M user's guide. Operation with a 745 KSR acoustical terminal is possible but an EIA/ auxiliary coupler cable kit (Part #983856) must be obtained from a TI Digital Systems Division distributor.

SIMULATING CONTROL OF AN ASSEMBLY LINE

Coupling the KSR-745 terminal to the TM990/100M microcomputer provides a more interactive terminal than the 301 microterminal so that the hardware can be expanded to simulate general kinds of input and output requirements encountered in light-manufacturing assembly lines. In addition, the "assembling" of the program is made easier by using a "line-by-line" assembler, which requires an EIA compatible terminal for this interaction.

Now, obviously, the output board shown in *Figure 3-9,* which contained only simple logic level inverters and an LED display, will not be adequate to provide the reaction power levels that are required for the simulated application. Therefore, new interface modules are needed.

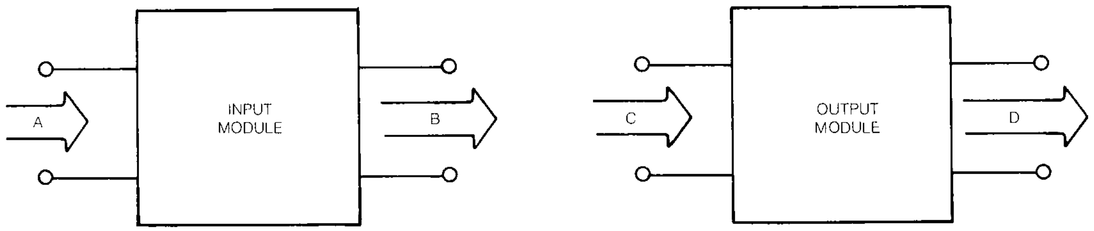
5MT Interface Modules

A means must be provided in the system to change input signals from push buttons, limit switches, cam switches, or transducers that are at voltage levels of 90-132 volts ac or 3 to 28 volts dc to standard TTL low-level logic signals between 0 and +5 volts.

In like fashion, means must also be provided in the hardware system to change the low-level logic output signals into power signals up to 28 volts dc or 90 to 132 volts ac. The concept is shown in *Figure 2.*

Texas Instruments supplies modules which meet these requirements. They are called the 5MT I/O modules that are part of a 5TI Control system. A simplified set of specifications for the basic modules is contained in *Table I.*

▶9 The I/O modules are solid-state devices incorporating optical coupler isolation between input and output of 1500 volts for excellent noise immunity. Internal protection is provided to guard against external voltage transients. Each module has an LED status indicator located at the low-level logic side of the module to help in set-up and troubleshooting. The I/O modules operate from 0-60° C and are designed for 100 million operations. The modules are shown in *Figure 3* with a 5MT43 mounting base which accepts 16 plug-in modules and provides all of the wiring terminals. A logic interface module which mounts on the 5MT mounting base is also shown in *Figure 3.* It provides a serial interface between the 5MT mounting base and a 5TI sequencer. It is not necessary for this application, but is very necessary if other 5TI components are interconnected in the system.



INPUT A	OUTPUT B
90-132VAC	LOW-LEVEL LOGIC
3-28VDC	LOW-LEVEL LOGIC

INPUT C	OUTPUT D
LOW-LEVEL LOGIC	90-132VAC
LOW-LEVEL LOGIC	3-28VDC

Figure 2. Input/Output Modules

CATALOG NO.	TYPE OF DEVICE	RATING		TURN ON TIME (ms)	TURN OFF TIME (ms)
		VOLTAGE	CURRENT		
5MT11-A05L	AC Input	90-132 Vac Input Voltage	35 mA Max	8 Typ. 8.3 Max	12 Typ. 8.3 Max
5MT12-40AL	AC Output	90-132 Vac Output Voltage	3 Amps Continuous (40° C)	4 Max	4 Max
5MT13-D03L	DC Input	3-28 Vdc Input Voltage	30 mA Max	2 Max	2 Max
5MT14-30CL	DC Output	10-28 Vdc Output Voltage	1 Amp Continuous (60° C)		
5MT43	Mounting Base Holds Up to 16 Modules				

Table 1. *5MT Module Selection Table*

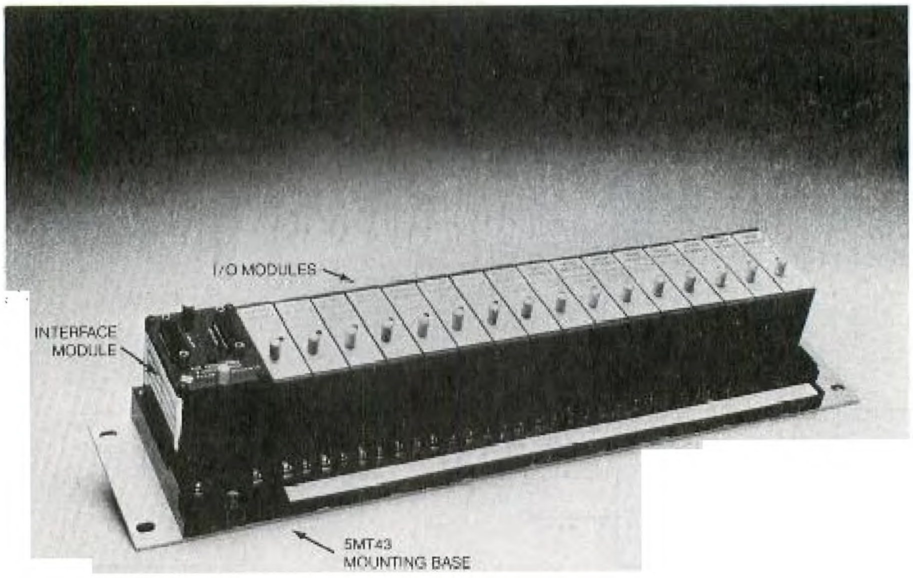


Figure 3. I/O Modules and Mounting Base

The 5MT43 mounting base interfaces with the TM 990/100M-1 microcomputer with a cable to P4, the same 40 pin edge connector that was used for the output board of *Figure 3-9.* The cable connections and hardware required are shown in *Figure 4.* This cable may be wired from scratch or a TM 990/507 cable can be purchased for the purpose. With this cable in place (J1 to the 5MT43 base and J4 to P4 on the TM 990/100M microcomputer module), the major components will be ready to simulate the industrial application. Of course, the additional parts must be purchased:

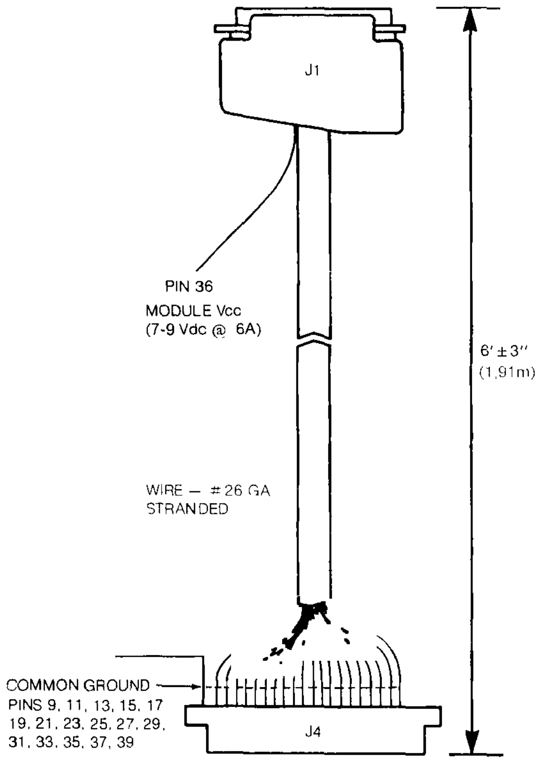
1 — 5MT43 Mounting Base
2 — 5MT11-A05L Input Modules
2 — 5MT12-40AL Output Modules
2 — 5MT13-D03L Input Modules
2 — 5MT14-30CL Output Modules
1 — TM990/507 Cable (or this can be fabricated as per *Figure 4*)

(Equivalent circuits of 5MT modules are provided in *Figure 5* in case these are to be simulated.)

WIRE LIST

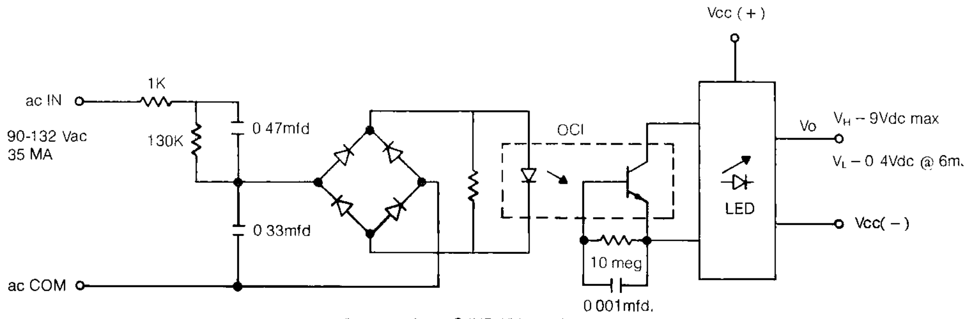
J1	J4	SIGNAL
1	10	MODULE 5
2	18	MODULE 4
3	14	MODULE 2
4	20	MODULE 0
5	24	MODULE 7
6	28	MODULE 9
7	32	MODULE 11
8	36	MODULE 13
9	40	MODULE 15
10	13	GROUND 2
11	19	GROUND 0
12	9	GROUND 5
13	23	GROUND 7
14	27	GROUND 9
15	31	GROUND 11
16	35	GROUND 13
17	39	GROUND 15
21	16	MODULE 3
22	22	MODULE 1
23	12	MODULE 6
24	26	MODULE 8
25	30	MODULE 10
26	34	MODULE 12
27	38	MODULE 14
28	15	GROUND 3
29	21	GROUND 1
30	17	GROUND 4
31	11	GROUND 6
32	25	GROUND 8
33	29	GROUND 10
34	33	GROUND 12
35	37	GROUND 14
36	#24 GA, STRANDED FOR MODULE V_{CC} (7 ~ 9V_{dc}@.6A) TERMINATION CAN BE #6 SPADE LUG, BANANA PLUG, ETC	

J1
37 PIN "D" TYPE CONNECTOR, FEMALE TYPE
AMP 205-209-1
TRW 6 INCH DC375

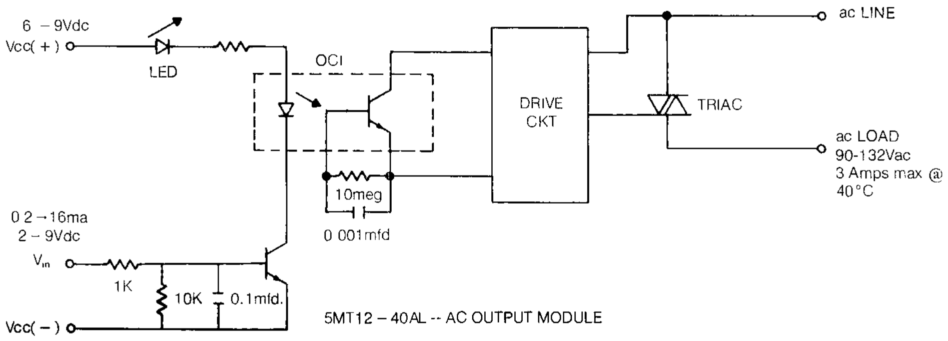


J4
40 PIN
0.100 C-C, PCB EDGE CONNECTOR
TI 421121-50 (WIRE WRAP)
TI 421111-50 (SOLDER TAIL)
VIKING 3VH20/1JN5

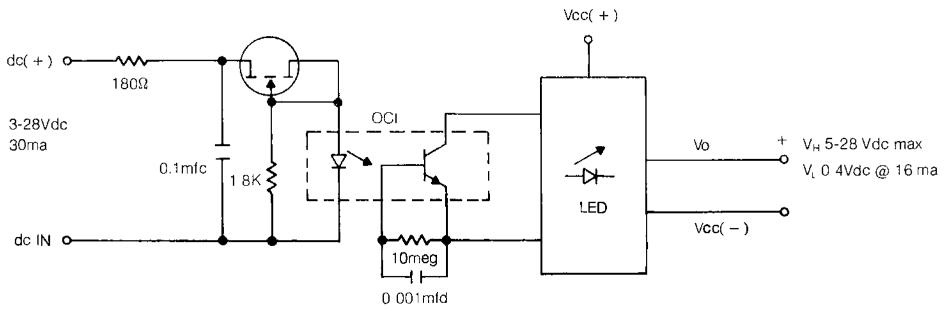
Figure 4. 5MT Interface Cable



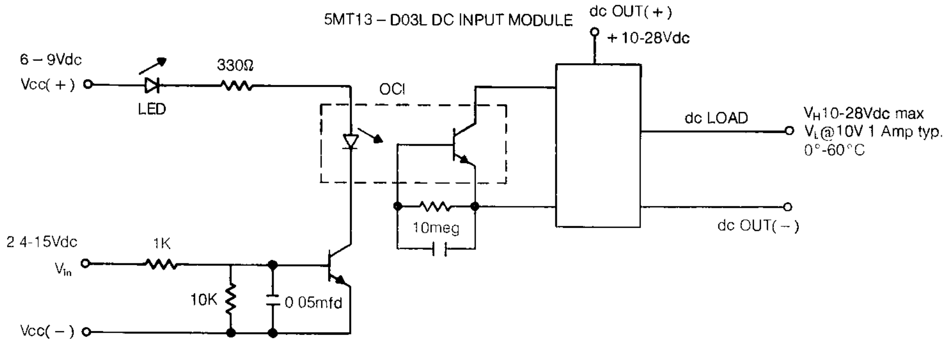
5MT11 – A05L – AC INPUT MODULE



5MT12 – 40AL -- AC OUTPUT MODULE



5MT13 – D03L DC INPUT MODULE



5MT14 – 30CL DC OUTPUT MODULE

Figure 5. Equivalent Circuits for 5MT Modules

Demonstration Example

The industrial control example, shown in concept form in the block diagram of *Figure 6* is intended to give the reader an insight into the use of a microcomputer based system. Even though no motors, actuators, solenoids, positioning valves, etc. are actually energized, the application demonstrates the means to do it. It also uses real world control voltages in its operation. There will be three modes of operation. To add interest, the system will be programmed so that the user can select the mode of operation.

In the first mode of operation *(Figure 6),* the system is to be programmed to accept inputs and switch a corresponding output according to the state of the input. Switches are going to apply input industrial level dc voltages to the dc input modules and input industrial level ac voltages to the ac input modules. Output lights powered by industrial level dc and ac voltages will be activated corresponding to the state of the input signal. Such a mode of operation simulates switch closures on the assembly line requesting an output reaction.

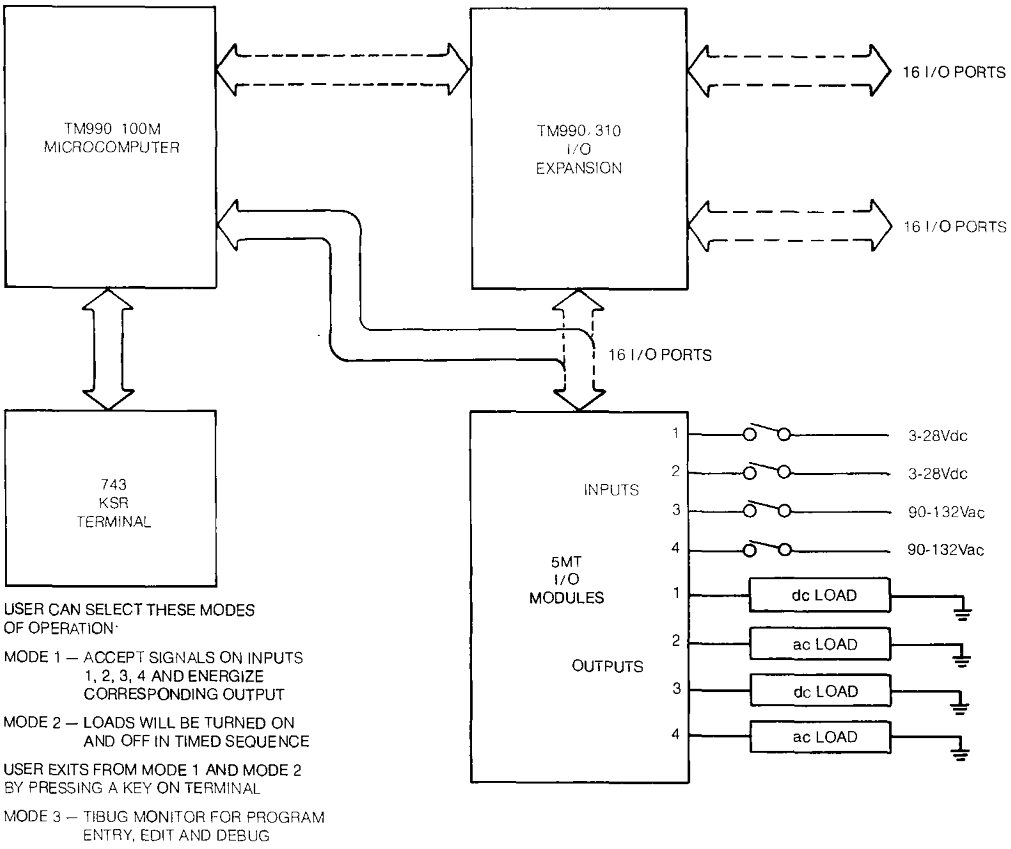

TM990 100M MICROCOMPUTER
TM990, 310 I/O EXPANSION
16 I/O PORTS
16 I/O PORTS
16 I/O PORTS
743 KSR TERMINAL
INPUTS
5MT I/O MODULES
OUTPUTS
1
2
3
4
1
2
3
4
3-28Vdc
3-28Vdc
90-132Vac
90-132Vac
dc LOAD
ac LOAD
dc LOAD
ac LOAD
USER CAN SELECT THESE MODES OF OPERATION·
MODE 1 — ACCEPT SIGNALS ON INPUTS 1, 2, 3, 4 AND ENERGIZE CORRESPONDING OUTPUT
MODE 2 — LOADS WILL BE TURNED ON AND OFF IN TIMED SEQUENCE
USER EXITS FROM MODE 1 AND MODE 2 BY PRESSING A KEY ON TERMINAL
MODE 3 — TIBUG MONITOR FOR PROGRAM ENTRY, EDIT AND DEBUG

Figure 6. *Application Block Diagram*

9◂

The second mode of operation is very similar to the light sequence of Chapter 3. However, with the 5MT modules controlling either + 12Vdc light bulbs or 110Vac light bulbs, it demonstrates a different means of timed sequence control. It uses the real time clock in the TMS9901 in the microcomputer module for a much greater precision. The system is to be programmed so the time can be varied easily. There is to be an added feature in the first and second mode. The system has a routine that allows the user to choose the mode of operation by selecting a key on the keyboard.

A third mode returns the system to the TIBUG interactive monitor. In this mode, the program can be edited, debugged or added to and initial conditions can be changed.

Lets see how this can be accomplished.

THE TM990/100M MICROCOMPUTER MODULE

Figure 7 is a much more detailed block diagram of the TM990/100M microcomputer. Four areas are of particular interest:

1. More details on the TMS9901;
2. Details on the TMS9902—this device was not discussed at all in Chapter 3;
3. The addition of a TM990/310 module to the system to obtain I/O expansion; and
4. Expansion of resident RAM and ROM.

Note in particular that the TM990/100M-1 comes populated with 256 words of RAM and 1K words of ROM (which is the TIBUG EPROM resident monitor). Also note the address bus goes to the I/O interface units. Thus, I/O is selected with addresses in the same fashion as memory words. In addition, the four busses—address, control, data and CRU are available for off-board expansion. This is the way I/O expansion through the TM990/310 module is controlled. 512 words of RAM can be provided on the board. Further expansion is possible with off-board memory. Additional ROM, expandable on the board to 4K, will be used when the line-by-line assembler (LBLA) is used.

TMS9901

The TMS9901, programmable system interface, shown in *Figure 7* was previously shown in the block diagram of *Figure 3-17.* Only one portion of it was used to control output signals and detect an input signal. Now all of the functions will be examined in more detail.

9 The block diagram of the TMS9901 in *Figure 8* will be used to identify the major functions.

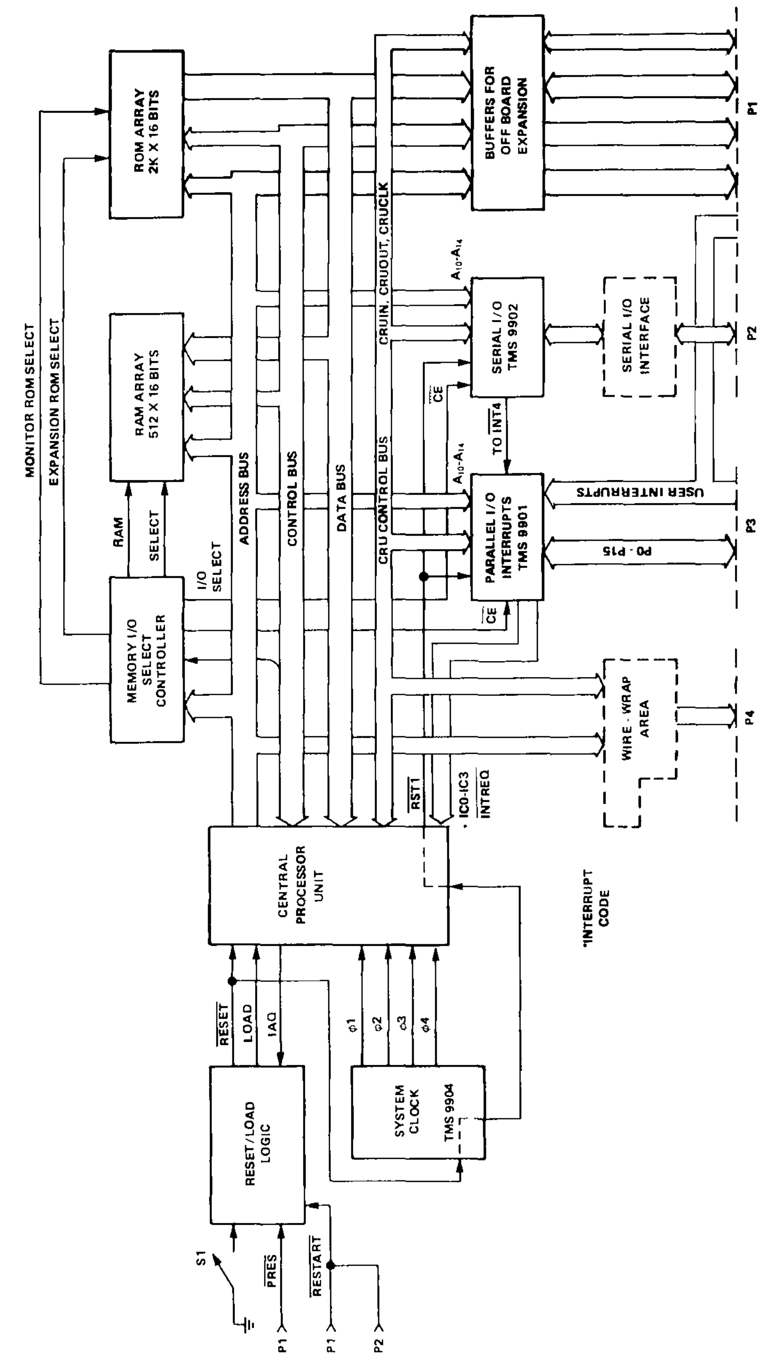


Figure 7. TM 990/100M Block Diagram

First of all, since the TMS9901 is a programmable systems interface, as shown in *Figure 7,* it is designed to handle parallel input and output signals. The input signals are either data inputs or special signals called interrupts. Interrupts are special signals because they interrupt the main program routine of the microcomputer and ask for service from the microcomputer to do some selected priority subroutine or subprogram. In *Figure 8,* the data output paths and input paths and the interrupt paths are identified. The 22 pins are programmable and divide into three groups as follows:

Table 1. Programmable Pin Functions

GROUP	NAME	IN	OUT	INT	COMMENT
1.	$\overline{\text{INT}}$ 1	X		X	Principally inputs but may be used as interrupts
	$\overline{\text{INT}}$ 2	X		X	
	$\overline{\text{INT}}$ 3	X		X	
	$\overline{\text{INT}}$ 4	X		X	
	$\overline{\text{INT}}$ 5	X		X	
	$\overline{\text{INT}}$ 6	X		X	
2.	$\overline{\text{INT}}$ 7/P15	X	X	X	Fully programmable as inputs, outputs or interrupts
	$\overline{\text{INT}}$ 8/P14	X	X	X	
	$\overline{\text{INT}}$ 9/P13	X	X	X	
	$\overline{\text{INT}}$ 10/P12	X	X	X	
	$\overline{\text{INT}}$ 11/P11	X	X	X	
	$\overline{\text{INT}}$ 12/P10	X	X	X	
	$\overline{\text{INT}}$ 13/P9	X	X	X	
	$\overline{\text{INT}}$ 14/P8	X	X	X	
	$\overline{\text{INT}}$ 15/P7	X	X	X	
3.	P6	X	X		Programmable as inputs or outputs.
	P5	X	X		
	P4	X	X		
	P3	X	X		
	P2	X	X		
	P1	X	X		
	P0	X	X		

In addition to the input/output function, the TMS9901 also has incorporated a clock function. This was identified in *Figure 8,* but is further detailed in *Figure 9.* This real time clock will be used in this application as an interval timer for the Mode 2 light sequence. To provide this function, the clock register is loaded with a value, (just like in Chapter 3); however, now the register automatically decrements after it is loaded. When it has decremented to zero, an interrupt signal is sent out to be processed by the interrupt path of the TMS9901. It won't be used for this application, but an elapsed time counter can be implemented by reading the value of the clock read register (*Figure 9*) periodically to determine how much time has elapsed from an established start.

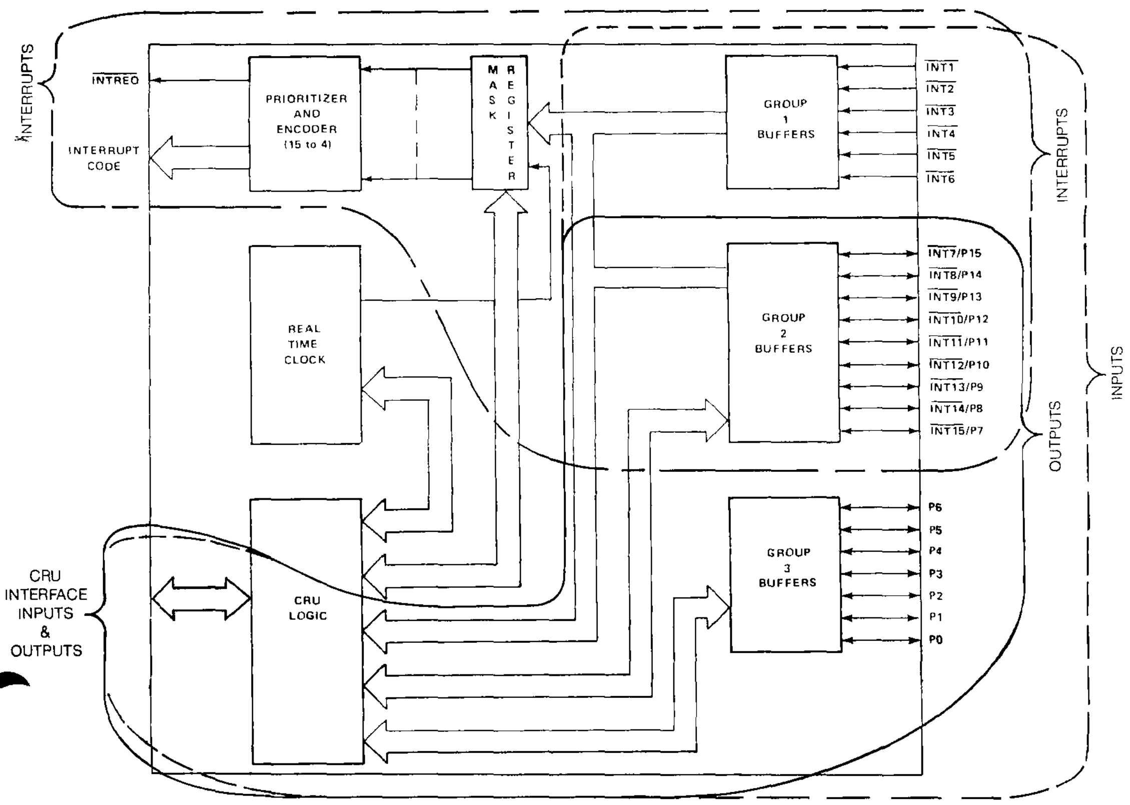


Figure 8. TMS 9901 Block Diagram

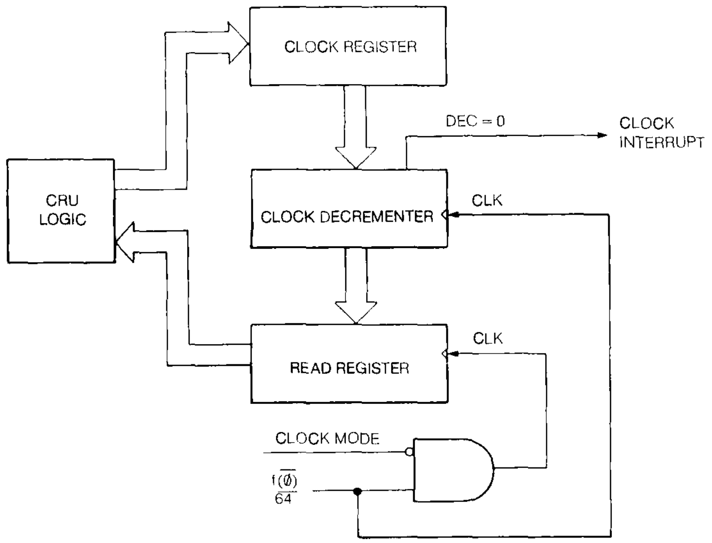


Figure 9. Real Time Clock

Interface with the 9900

It is important to understand the communications channels between the TMS 9901 and the 9900 microprocessor in the microcomputer. Basic concepts need to developed to understand how the algorithm for this application is programmed.

The communications channels are shown in *Figure 10.* They are presented in somewhat different form than shown previously in Chapter 3.

The main data link between the 9900 and the 9901 and subsequent inputs and outputs is via a serial data link. The line CRUIN transfers data from the 9901 to the 9900 in serial format. Again in serial format, the line CRUOUT transfers data from the 9900 to the 9901. The transfer of data out is synchronized by the signal CRUCLK, which comes from the 9900 and specifies that data is valid on the CRUOUT line. Remember that CRU means Communications Register Unit.

In order to manipulate data from the CRU to and from the inputs and outputs and the real time clock of the 9901, five CRU instructions are included in the instruction set. They are:

1.	SBO	Set bit to one
2.	SBZ	Set bit to zero
3.	TB	Test bit
4.	LDCR	Load CRU Register
5.	STCR	Store CRU Register

In Chapter 3, it was demonstrated how individual bits could be selected and set to a "1" or a "0" by using the SBO and SBZ instructions. If this hasn't been reviewed, it would be helpful to do so.

Not only can individual bits be manipulated, but data can also be transferred in blocks of from one to 16 bits. The multiple bit instructions LDCR, "Load CRU Register", and STCR, "Store CRU Register", are used for this purpose. Since this application requires the use of these multiple-bit instructions, further time will be spent explaining them in more detail.

Basic Concepts

Figure 11 summarizes the basic concept of the programmable input-output capability of the 9900 family. In this example, a microcomputer, the TM990/100M, which contains a 9901, and a TM990/310 module, which contains 3 additional 9901's are used. Such an arrangement expands the I/O capabilities by 48 inputs or outputs.

Industrial control applications like the one that is being simulated normally require many inputs and outputs. Much more capability is available because I/O could be expanded to 4096 ports by adding more units and continuing the example of *Figure 11.*

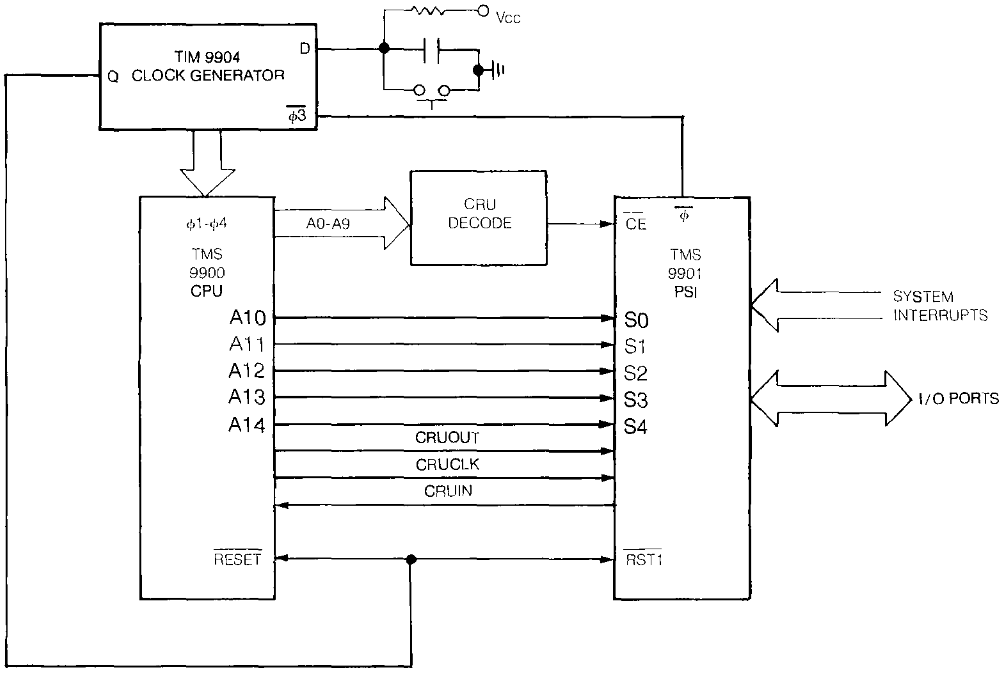


Figure 10. TMS 9900-TMS 9901 Interface

As shown, the data moves over CRUIN and CRUOUT in a serial format from the 9900 to the 9901, or vice versa. When the instruction LDCR is used, the data is flowing from the 9900 to the 9901 over CRUOUT. The first bit to arrive serially (the least significant bit) is latched in the zero bit position of the 9901 determined by the CRU select bit, subsequent bits that arrive are then placed in bits, 1, 2, 3-12, 13, 14, 15 at each CRUCLK pulse. Such is the case if 16-bits are being processed. Any number of bits from 1 to 16 may be processed at the user's discretion. When flowing out on CRUOUT, the transfer rate is determined by CRUCLK. When flowing in on CRUIN, the 9900 microprocessor transfers the data present on the inputs during ϕ_1 of clock cycle 2 of the machine cycles.

What determines where the bit position starts? The select bits on S_0-S_4 in the 9901 (*Figure 10* and *11*) are distributed as A_{10} thru A_{14} from the 9900. Since this address is distributed to each 9901 shown, and since CRUOUT goes to each 9901, the data out would tend to be latched in each 9901. This is prevented by the chip enable (CE) signal. The only CE that is active low is the one decoded from the corresponding base address for the correct 9901. Bits A_0 thru A_9 provide the additional address information. For example, if in *Figure 11* the 9901 on the TM990/100M board is to be used for the I/O, then hardware base address 0080_{16} is used. If the second 9901 on the TM990/310 module is used, the hardware base address is 0140_{16}.

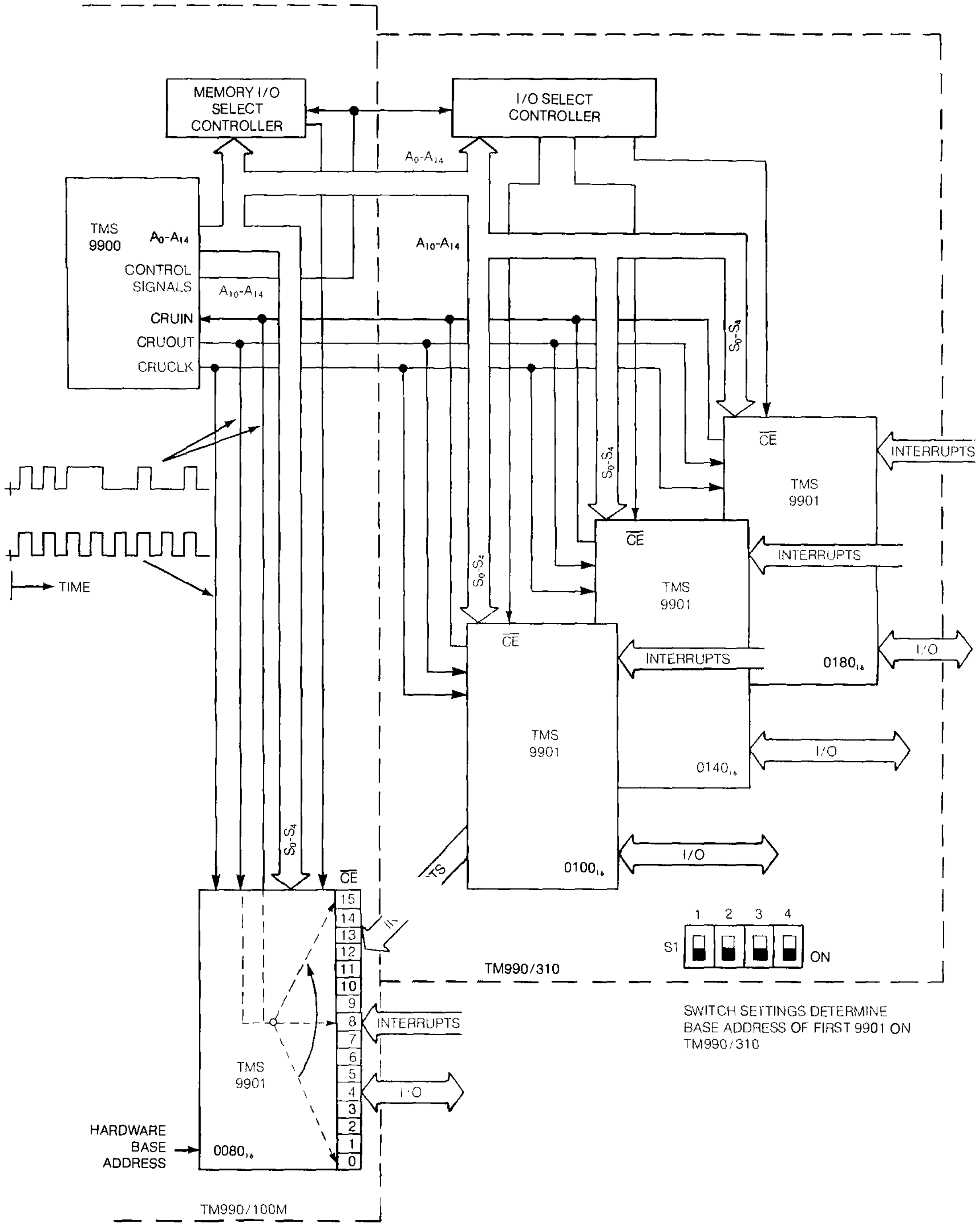


Figure 11. Basic Concept of Programmable I/O

In *Figure 3-23,* for the single bit instructions SBO, SBZ, and TB, the effective CRU bit address is obtained by adding a signed displacement to the 9901 base address. For the multiple bit instructions, the effective CRU bit address is computed in the same way; however, the base address is the address of the first bit. From there, the address is incremented by the number of multiple bits to be transferred. The LDCR instruction format contains a C field which specifies the number of multiple bits to be transferred. For example:

```
        LDCR    R1,9
```

would instruct the microcomputer to send out (output) the 9 least significant bits of register R1. The 9 would be in the C field of the instruction format. Before the LDCR instruction in the program, there is an instruction that loaded the software base address of the particular 9901 to be used into the correct workspace register 12. Recall that WR12 is the register where the software base address is always located for a CRU instruction. This will become clearer as a specific example is discussed later. What is important is that the software base address for the 9901 must be loaded into workspace register 12. However, this is not completely straightforward. For example, if the 9901 on the TM990/100M microcomputer is to be addressed with a LDCR or STCR instruction, the 0080_{16} hardware base address must be displaced to the software base address 0100_{16} when it is loaded into WR12. This is necessary because bit 15 of WR12 is not used in the calculation of the effective CRU bit address. The concept, described in *Figure 3-23,* is shown again in *Figure 12.*

It is probably obvious that the STCR instruction operates in the reverse of the LDCR. The data from the input pins on the selected 9901 is incremented bit by bit and sent to the CRU in the 9900 over CRUIN. The final result of a STCR instruction is that the 9900 processor stores the input data in RAM in a specified location called out in the instruction. In like fashion, when LDCR is used the data transferred to the output is obtained from a RAM location called out in the instruction. This is a distinct advantage in that it need not be a register. The specifics on the data transfers are shown in *Figure 13.*

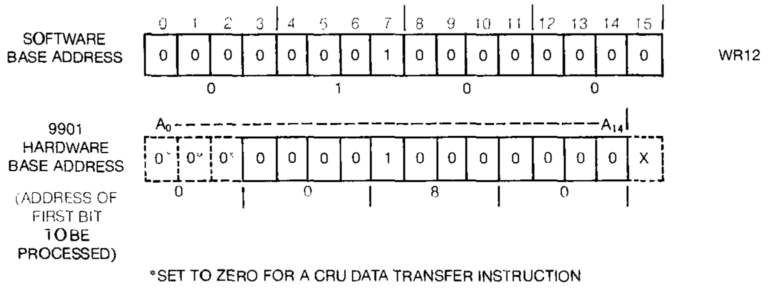

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
SOFTWARE BASE ADDRESS
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
WR12
0 1 0 0
9901 HARDWARE BASE ADDRESS
A0
A14
0* 0* 0* 0 0 0 0 1 0 0 0 0 0 0 0 X
0 0 8 0
(ADDRESS OF FIRST BIT TO BE PROCESSED)
*SET TO ZERO FOR A CRU DATA TRANSFER INSTRUCTION

Figure 12. 9901 Base Address

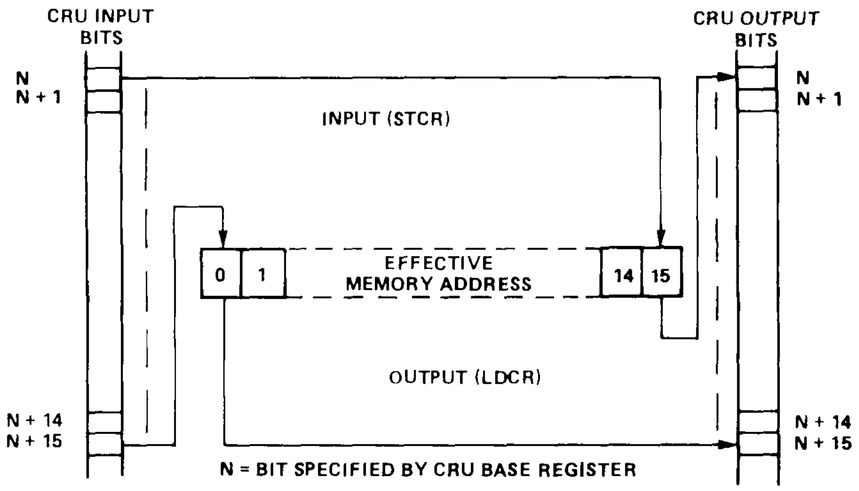


Figure 13. LDCR / STCR Data Transfers

Interrupts

Another form of input is the special one called interrupt, so named because it asks the microcomputer to interrupt the program routine presently in process.

In *Figure 8,* it was pointed out that there are only certain lines on which an interrupt is accepted. Group 1 of the 9901 pins may be used for 6 interrupts. Up to 15 interrupt signals can be programmed by using Group 2 pins.

What value do interrupts have? First, they allow external events to interrupt the current program so that the program can provide service to an external device. In so doing certain pieces of data must be saved in order to return to the same point in the program that was interrupted. This allows the program to continue correctly after the interrupt has been serviced. Secondly, interrupts provide quick response. Third, they provide a priority to be established for time critical events. Certain interrupts are more important than others. The user decides the priority. To set up priorities for interrupt signals, a means is provided to honor the priority established. In the 9900 system family, this is called enabling a valid interrupt through a "masking" of interrupts.

Masking means to enable or disable. *Figure 14* shows that the TM990/100M microcomputer module has two levels of masking. One mask must be enabled to pass the interrupt signals through the 9901 and another must be enabled at the 9900 microprocessor. The value in bits 12, 13, 14 and 15 of the status register set the priority level of the interrupt mask in the 9900. Any interrupt equal to or higher than the priority level is enabled and allowed to interrupt the microcomputer.

Masking

Figure 15 is a block diagram of the 9901 control logic illustrating how the masking is accomplished. In order to enable an interrupt, MASK must equal 1 for the particular interrupt pin. When several interrupts are present at the same time, the control logic encodes the enabled interrupt inputs and sends to the 9900 microprocessor a code that represents the highest level of interrupt that has been enabled. $\overline{\text{INT}}$ 1 is the highest level, $\overline{\text{INT}}$ 2 is next and so on down to 15. In addition, an $\overline{\text{INTREQ}}$ active low signal is also sent to the 9900. The code sent on lines IC0 through IC3 is shown in *Table 2.* Level zero is used by $\overline{\text{RESET}}$ and will be covered later.

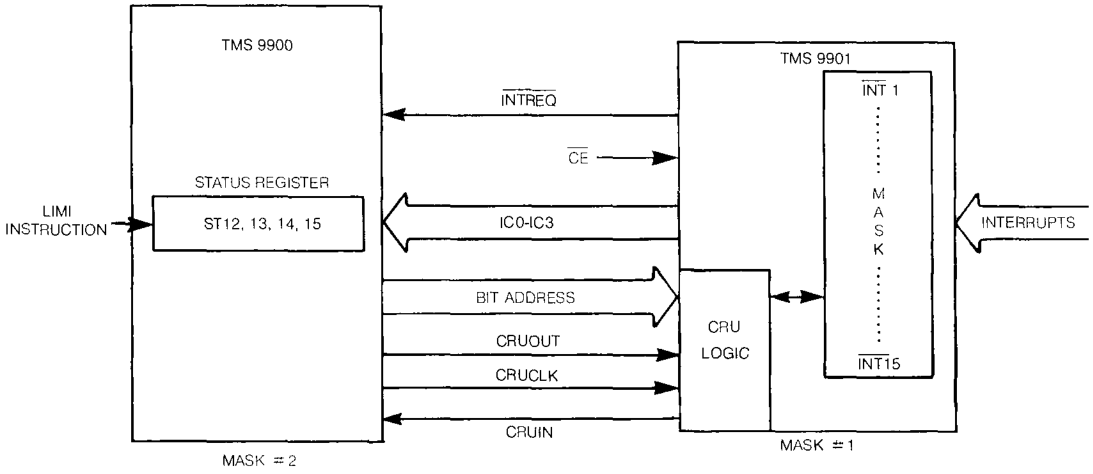


Figure 14. Interrupt Masking

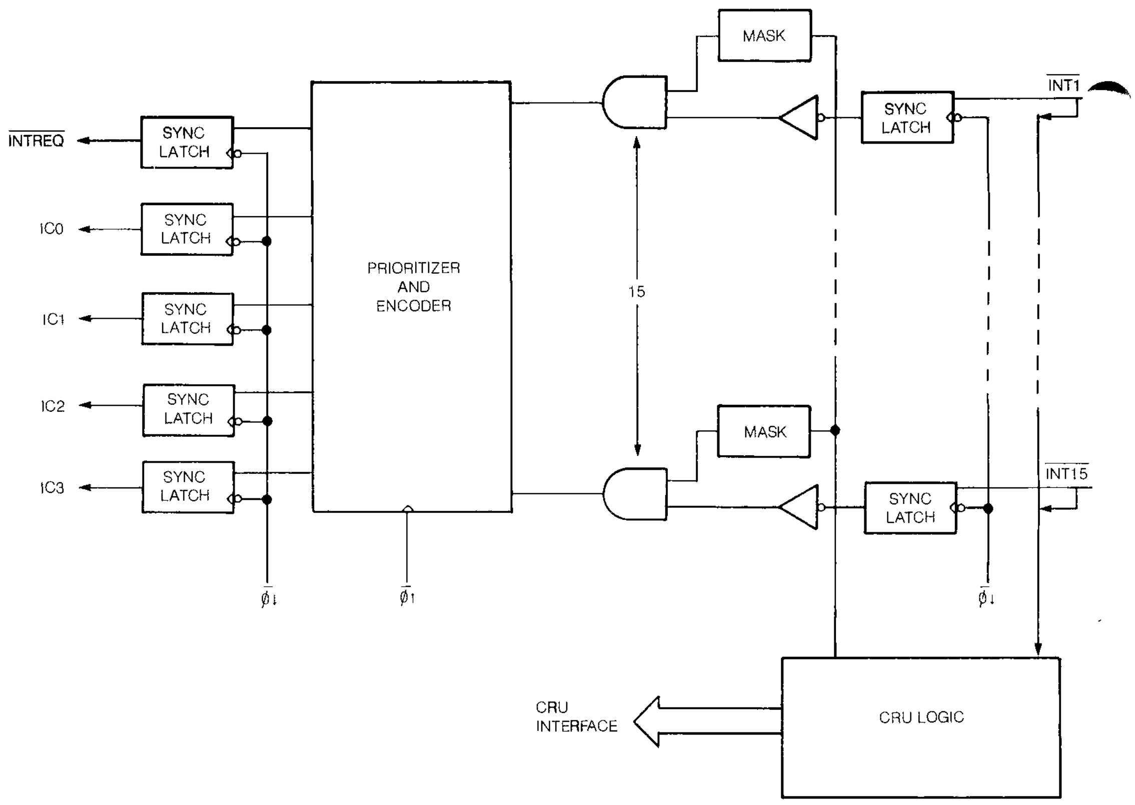


Figure 15. Interrupt Control Logic

The code on IC0 thru IC3 is compared to the status bits ST12, 13, 14 and 15 in the status register of the 9900. The priority level loaded into the interrupt mask of the 9900 enables that level and all higher priority levels as well. If the interrupt level set up in ST12, 13, 14 and 15 is higher than the interrupt level received, the interrupt is not enabled. If the interrupt received is higher in level than the priority level, then the interrupt is enabled and all higher level interrupts as well. This is shown in *Figure 16.*

The code on IC0-IC3 is as follows:

Table 2. Interrupt Code Generation

INTERRUPT/STATE	*PRIORITY*	*IC0*	*IC1*	*IC2*	*IC3*	*$\overline{INTREQ}$*
$\overline{\text{INT}}$ 1	1 (HIGHEST)	0	0	0	1	0
$\overline{\text{INT}}$ 2	2	0	0	1	0	0
$\overline{\text{INT}}$ 3/CLOCK	3	0	0	1	1	0
$\overline{\text{INT}}$ 4	4	0	1	0	0	0
$\overline{\text{INT}}$ 5	5	0	1	0	1	0
$\overline{\text{INT}}$ 6	6	0	1	1	0	0
$\overline{\text{INT}}$ 7	7	0	1	1	1	0
$\overline{\text{INT}}$ 8	8	1	0	0	0	0
$\overline{\text{INT}}$ 9	9	1	0	0	1	0
$\overline{\text{INT}}$ 10	10	1	0	1	0	0
$\overline{\text{INT}}$ 11	11	1	0	1	1	0
$\overline{\text{INT}}$ 12	12	1	1	0	0	0
$\overline{\text{INT}}$ 13	13	1	1	0	1	0
$\overline{\text{INT}}$ 14	14	1	1	1	0	0
$\overline{\text{INT}}$ 15	15 (LOWEST)	1	1	1	1	0
NO INTERRUPT		1	1	1	1	1

The output signals will remain valid until the corresponding interrupt input is removed, or an interrupt service routine disables (MASK = 0), or a higher priority enabled interrupt becomes active. When the highest priority enabled interrupt is removed, the code corresponding to the next highest priority enabled interrupt is output. If no enabled interrupt is active, all CPU interface lines ($\overline{\text{INTREQ}}$, IC0-IC3) are held high.

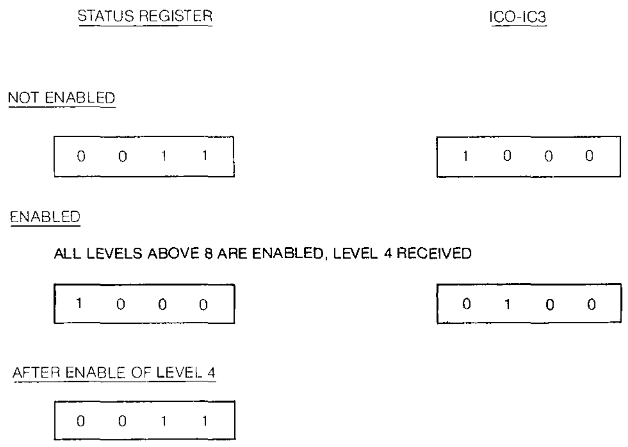


Figure 16. Interrupt Mask at 9900

9◀

Remember to enable an interrupt, say $\overline{\text{INT}}$ 1, a "1" must be placed in the latch (MASK = 1) for the CRU bit (pin) associated with that interrupt. Likewise, to disable an interrupt, a "0" must be placed in the latch (MASK = 0) associated with the pin receiving the particular interrupt.

To mask any of the interrupts from 1 through 15, the 9901 must be in the interrupt mode. The zero select bit of the 9901 is the control bit for this. As shown in *Figure 23,* if this control bit is a zero, the 9901 is in the interrupt mode. If it is a "1", the 9901 is in the clock mode.

Enabling or disabling the mask in the 9901 for the interrupts may be accomplished by individual bit instructions SBO and SBZ or by a multiple bit LDCR instruction.

All masks can be disabled simultaneously by performing a hardware ($\overline{\text{RESET}}$) or software ($\overline{\text{RST}}$ 2) reset.

Signals appearing on the inputs to the 9901 will be accepted as interrupt signals by the 9901 if the masks are enabled. The priority code for the highest priority level interrupt simultaneously received will be sent to the 9900 via the code lines, IC0-IC3, as well as the signal $\overline{\text{INTREQ}}$. If the interrupt mask in the 9900 has the level enabled, the interrupt is accepted and serviced.

Saving Items on Interrupt

When an interrupt occurs, data pertinent to the "state of the machine" must be saved. This provides a return to the interrupted program so that the program can continue to execute properly. For example, when an interrupt occurs, the CPU suspends its current program routine to do the subroutine called for by the interrupt. How does it do this? As any program executes, the "state of the machine" at any time is determined by the value in the program counter, the value in the workspace pointer, the value in the status register, and the contents of the registers in the workspace register file. Each of these is saved through a "context switch" when an interrupt occurs. Full details are available in Chapter 4. A brief summary will be covered here for convenience.

Interrupt Vectors — Context Switching

To execute an interrupt, here's what happens. There are special places in memory reserved for the address that contains a new workspace pointer for a given interrupt. In addition, in the next word following there is a new program counter value. These special places in memory are called interrupt vector traps and the two addresses — one for workspace and the other for the program counter — have the name "interrupt vector."

Figure 17 illustrated the process. A valid interrupt is received and its level points to its vector. The vector contains a new workspace pointer and a new program counter value. The program shifts and points to the new workspace. In the new workspace, the microprocessor stores the old workspace pointer in R13, the old program counter in R14 and the old status register in R15. These old contents are always put in the same place in the new workspace — R13, R14 and R15.

After all this occurs, the program counter with its new value executes the interrupt subroutine. The last instruction in this subroutine, RTWP, is an instruction to return to the interrupted routine. RTWP — "Return with Workspace Pointer" — returns to the interrupted routine by loading the contents of R13 into the workspace pointer (R13→WP), R14 into the program counter (R14→PC), and R15 into the status register (R15→ST) and then executes the instruction pointed to by the program counter. In so doing, the system has returned to the interrupted program at the point of interruption and begins execution using the old workspace. This is illustrated in *Figure 18.*

Note: When the interrupt priority level comes into the 9900 and the interrupt is enabled, a number one less than the interrupt level received is placed in the interrupt mask in the status register as shown in *Figure 16* to prevent lower level interrupts from occurring during the servicing of the present interrupt. If a higher priority interrupt occurs, a second interrupt context switch takes place after at least one instruction is executed for the first interrupt routine. This means that an interrupt service routine may begin with a LIMI instruction which can load an interrupt mask in the 9900 which disables other interrupts. Completion of the second interrupt passes control back to the first interrupt using the RTWP instruction.

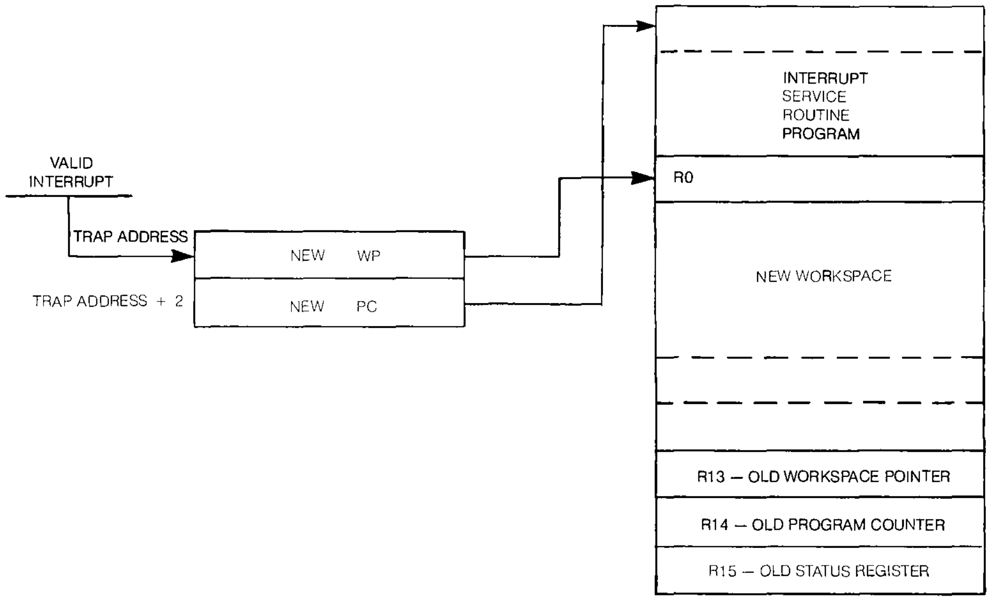


Figure 17. Interrupt Context Switch — New Workspace and Saving Old WP, PC, and ST Data

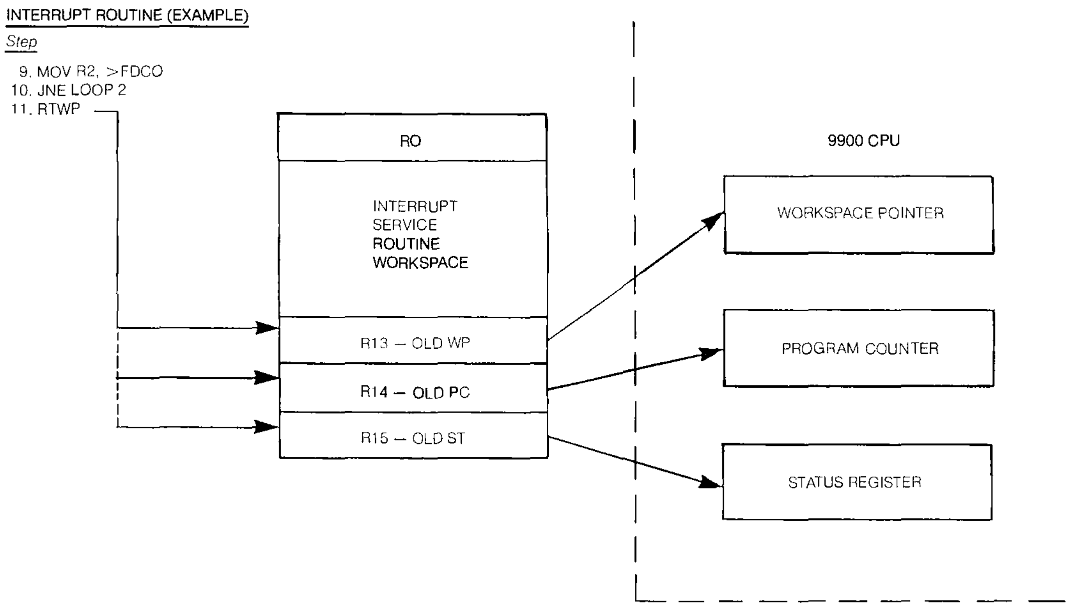


Figure 18. Interrupt Context Switch Returning to Interrupted Program

Memory Map and Interrupt Vectors

In *Figure 19,* the memory map of the TM990/100M microcomputer module is shown. Note that the first words of memory from hexadecimal addresses 0000_{16} to $07FE_{16}$ are dedicated memory. Addresses 0000_{16} to $003E_{16}$ are reserved for the 16 interrupt transfer vectors. These are detailed further in Figure 20. Each interrupt vector has two words of memory — one for the workspace pointer, one for the program counter.

There are two interrupt vectors, $\overline{\text{INT}}$ 3 and $\overline{\text{INT}}$ 4 that will be of particular interest for they have important use in the program for this application.

Notice that interrupt 0 in *Figure 20* is used for $\overline{\text{RESET}}$ and that values have already been placed in the vector locations for interrupt 3 and interrupt 4.

When an $\overline{\text{INT}}$ 3 level is received, it points to the interrupt 3 vector. The context switch occurs and at $000C_{16}$ it obtains the value $FF68_{16}$ for the workspace pointer and at $000E_{16}$ the value $FF88_{16}$ for the program counter. The context switch operations store the old context registers in the new workspace pointed to by $FF68_{16}$. Then the interrupt service routine begins by executing the instruction pointed to by $FF88_{16}$. Since there are valid reserved locations for only two memory words at the $FF88_{16}$ location, the instruction pointed to by $FF88_{16}$ and $FF8A_{16}$ must branch to another section of memory where the remaining interrupt service routine is located.

A similar sequence of events occurs when an $\overline{\text{INT}}$ 4 level interrupt signal is received, except that the workspace pointer value is $FF8C_{16}$ and the program counter value is $FFAC_{16}$.

The remaining interrupt vectors do not have values. These would be programmed into EPROM locations by the user as the need arises.

For the interrupt 3 and 4 service routines, 16-word workspaces are provided, pointed to by $FF68_{16}$ and $FF8C_{16}$. These are reserved and must be noted by the programmer.

The microcomputer must always start from initial conditions. These are usually started by a reset. The vector space required for the initial value of the workspace pointer and the program counter resides in the reserved memory spaces 0000_{16} for WP and 0002_{16} for PC, as shown in *Figure 20.* The 16 interrupt vectors at 0000_{16} to $003E_{16}$ are in read only memory and cannot be changed unless the read only memory is reprogrammed.

As the extended application program is written, it must be remembered that the TIBUG monitor needs workspaces. The space from $FFB0_{16}$ to $FFFB_{16}$ is reserved for this purpose. This is noted because this space cannot be used for data or program memory in the application.

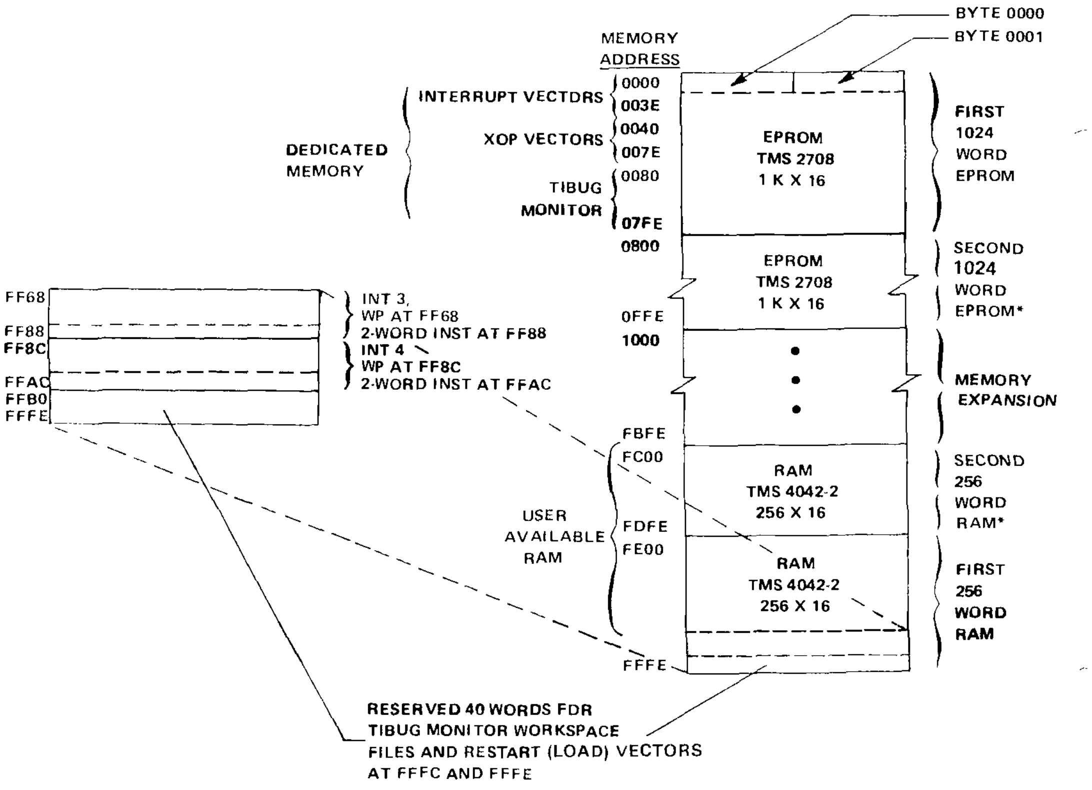


Figure 19. Memory Map

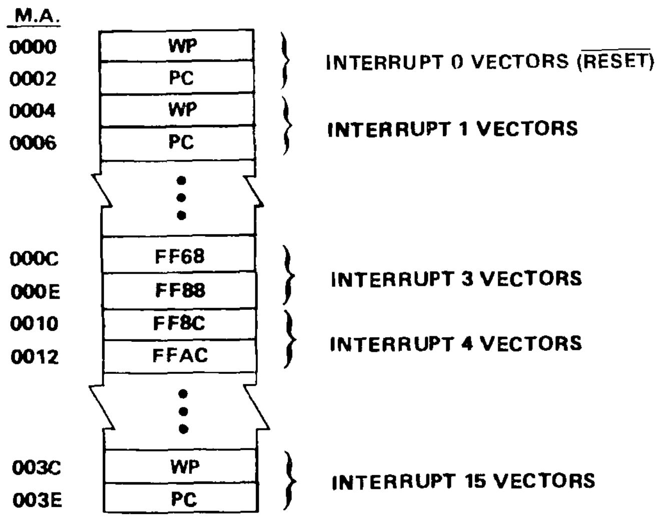


Figure 20. Interrupt Trap Locations

▸9

Extended Operations (XOP's)

Refer to *Figure 19* which shows the read-only memory space reserved for software interrupt vectors. Memory words from 0040_{16} to $007E_{16}$ are XOP vectors. As with interrupts, each XOP vector has a word containing a workspace pointer value and a next word containing a program counter value.

XOP instructions point to XOP vectors which point to new workspace pointer and program counter values in a similar way to what was just described for interrupts.

An instruction calling for an XOP (extended operation) is a means of switching from the main program to a subroutine. It has a special calling sequence and it functions as though the routine were a single instruction added to the 9900 set of operation codes, hence the name "extended operation".

For example, the TIBUG monitor in the microcomputer contains seven XOP routines that perform input/output functions with the terminal. These are as follows:

XOP	*Description*
8	Write one hexadecimal character to terminal
9	Read hexadecimal word from terminal
10	Write 4 hexadecimal characters to terminal
11	Echo character
12	Write one character to terminal
13	Read one character from terminal
14	Write message to terminal

Two of these XOPs are used in the extended application example. XOP 11 is used to read a character from the terminal and at the same time print it at the terminal. XOP 14 is used to print out instructions to explain how the program operates. Some of these XOPs call other XOPs. Further detail on XOPs can be obtained in Chapter 5 and 6.

Printing a Message

A message at the beginning of the program which will be developed for this application tells the user to select the mode of operation. XOP 14 is used to write the message. The instruction

```
XOP @MSG1,14
```

is used. XOP 14 identifies that the subtask is "Write message to terminal". A context switch takes place. The vector at location 14 of the reserved XOP vector memory space provides the WP and the PC values. The PC value provides the first subtask instruction and the subroutine continues until the subtask is complete and the program returns to the main program.

Suppose the message identified with the label MSG1 is "THIS IS A SAMPLE." Its coding would look like the following:

LINE	ADDRESS	CDDE	MESSAGE	ASCI I CDDE	
0	MSG1	5448	$THIS IS A SAMPLE.	A	41
				E	45
1		4953		H	48
				I	49
2		2049		L	4C
				M	4D
3		5320		P	50
				S	53
4		4120		T	54
5		5341			
6		4D50		SPACE (SP)	20
7		4C45		LINE FEED (LF)	0A
8		2E20			
9		0D0A	+>0D0A	CARRIAGE RETURN (CR)	0D
10		0000	+>0000	• (PERIDD)	2E

Note that line 9 contains a carriage return and a line feed and has the code 0D0A. The message beginning at location MSG1 is preceded by a dollar sign and terminated with a byte containing all binary zeroes. The + > 0D0A is a code recognized by the line-by-line assembler that is loaded directly into memory. It is initiated by typing the (+) before the desired number. The dollar sign indicates that a comment is being entered. Such XOPs are very useful in calling subroutines prepared to accomplish specific terminal functions.

Selecting a Mode

XOP 11 will be used to make the choice of the mode of operation. ECHO CHARACTER means that whatever key is pressed on the terminal will be read into a designated workspace register and then sent back from the register and printed on the terminal.

The one instruction,

```
XOP R5,11
```

9 accomplishes this. If a key is pressed, the terminal reads the character, places it in workspace register 5 and then prints the character on the terminal. The XOP subroutine was provided by the TIBUG monitor but it all was accomplished with one instruction — thus, the "extended operation."

TMS9902

The TMS9902, asynchronous communications controller provides an interface between the EIA terminal (serial asynchronous communications channel) and the 9900 in the TM990/100M microcomputer module. The block diagram of the microcomputer was shown in *Figure 7.* A simplified one is shown in *Figure 21a.* Note that the interface to the CPU (TMS9900) is the same as for the 9901. Note also the line $\overline{\text{INT}}$ 4 going from the 9902 to the 9901; this interrupt line will be important in this application.

All of the discussion that pertained to the 9901 and the addressing of the I/O bits also applies to the 9902. It has the same address bits A_{10}—A_{14} used for addressing the CRU bits inside the 9902 through S_0—S_4. It has the same CRU control bus signals for communication over the CRU serial data link.

A base address and $\overline{\text{CE}}$ select the 9902 over other I/O units that might be available in the system (in this case, only 9901s are present). The hardware base address 0040_{16} identifies the 9902 contained in the microcomputer. The software base address of 0080_{16} is loaded into WR12. This is added to the appropriate displacement to arrive at the effective CRU bit address desired as described for the 9901.

In this extended application, pressing a key on the terminal while the system is in mode 1 or mode 2 will switch the system back to the command mode. The user then selects a new mode of operation. This is a common way to use a terminal and the 9902 must be programmed to accomplish it. The arrangement is as shown in *Figure 21a.*

First, the 9902 must recognize that a character has been generated by the terminal and received by the 9902. Second, the output signal line $\overline{\text{INT}}$ from the 9902 must be enabled so it can pass the signal to the 9901 input $\overline{\text{INT}}$ 4. Since the 9901 receives this signal as an interrupt, then interrupt masks at the 9901 and the 9900 must be enabled. With these steps accomplished, the main program of the processor is interrupted and the operation mode is shifted.

Figure 21b shows that $\overline{\text{INT}}$ will be active in the receive mode if RBRL = 1 and RIENB = 1. RBRL will be a "1" when the Receive Buffer Register has received a character and stored it. This happens when a key is pressed. The 9902 is enabled by making RIENB (Receiver Interrupt Enable) a "1". *Figure 22* identifies that CRU bit 18 must be made a "1" to make RIENB = 1. A CRU SBO instruction with a displacement of 18 will set CRU bit 18 to a "1" if the software base address has previously been loaded in WR12.

Since $\overline{\text{INT}}$4 is the desired interrupt level, it is enabled in the 9900 by placing this level in its interrupt mask. This is accomplished with an instruction LIMI 4 which loads the value 4 into the status register.

With the $\overline{\text{INT}}$4 enabled at the 9901 by placing a "1" in the CRU bit mask corresponding to the input for $\overline{\text{INT}}$4, the 9901 sends the interrupt code to the 9900 over IC0-IC3 when the $\overline{\text{INT}}$ signal is received from the 9902. Since $\overline{\text{INT}}$4 is enabled in the 9900, the signal path is complete and the operating mode shifts.

$\overline{\text{INT}}$4 executes a context switch and finds its new workspace pointer is $FF8C_{16}$ and its new PC is $FFAC_{16}$.

In all the discussion, only the enabling of interrupts has been covered. It must be stressed that similar instructions in many cases must be included in the programming to disable an interrupt once it has been enabled.

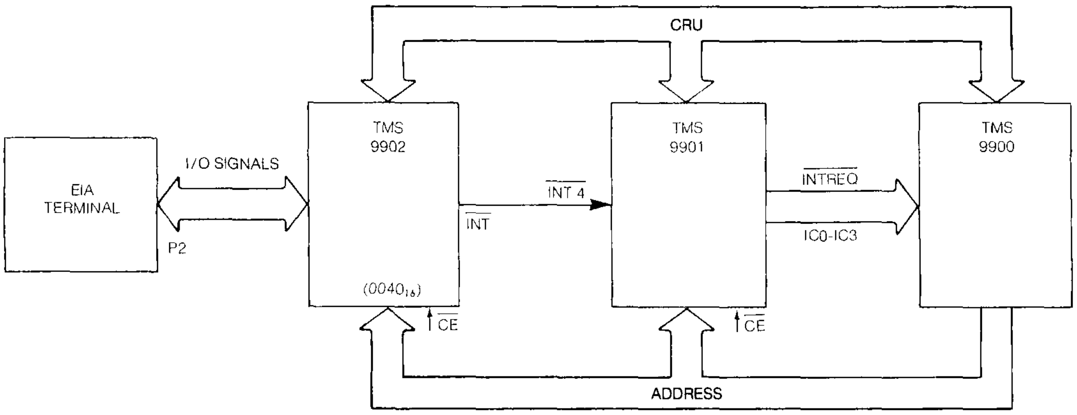


Figure 21a. Simplified Block Diagram Showing TMS 9902 Interface

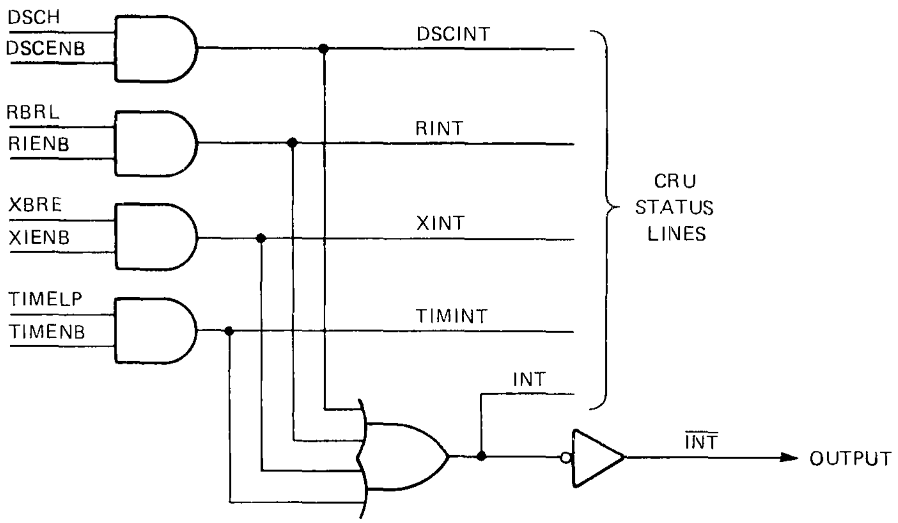


Figure 21b. $\overline{\text{INT}}$ Output Generation

ADDRESS$_2$ S0 S1 S2 S3 S4	ADDRESS$_{10}$	NAME	DESCRIPTION
1 1 1 1 1	31	RESET	Reset device.
	30-22		Not used.
1 0 1 0 1	21	DSCENB	Data Set Status Change Interrupt Enable.
1 0 1 0 0	20	TIMENB	Timer Interrupt Enable
1 0 0 1 1	19	XBIENB	Transmitter Interrupt Enable
1 0 0 1 0	18	RIENB	Receiver Interrupt Enable
1 0 0 0 1	17	BRKON	Break On
1 0 0 0 0	16	RTSON	Request to Send On
0 1 1 1 1	15	TSTMD	Test Mode
0 1 1 1 0	14	LDCTRL	Load Control Register
0 1 1 0 1	13	LDIR	Load Interval Register
0 1 1 0 0	12	LRDR	Load Receiver Data Rate Register
0 1 0 1 1	11	LXDR	Load Transmit Data Rate Register
	10-0		Control, Interval, Receive Data Rate, Transmit Data Rate, and Transmit Buffer Registers

Figure 22. TMS 9902 ACC Output Bit Address Assignments

PROGRAMMING THE 9901 I/O

The discussion, previously quite general, now gets more specific, focusing on how the program will have to be written to satisfy the requirements of the application. Since all input and output signals must go through the 9901, let's begin there. Refer to *Figure 23.*

Note that there are multiple functions for the pins on the 9901. The pins are referenced to establish the link between Group 1, Group 2 and Group 3 which were mentioned previously in the text. Note that all the functions are referenced to a select bit number from 0 to 31. Select bit zero is addressed when the 9901 base address is called. For example, the instruction:

```
SBO 0
```

addresses select bit zero in the 9901 and will set this bit, called the control bit, to a "1". Because it was bit zero, there was no additional displacement value added to the base address. However, as was done in Chapter 3, 10_{16} will be added to the 9901 hardware base address in the microcomputer when P_0 thru P_{15} are being used as data inputs and data outputs. This makes the base address point to select bit 16 as indicated in *Figure 23.* It makes the assignment of I/O bit 0 correspond to P_0, bit 1 to P_1, bit 2 to P_2, etc.

Figure 23 shows how select bit zero, the control bit, controls the mode of the 9901. When it is a "0", the 9901 is in the interrupt mode; when it is a "1", the 9901 is in the clock mode. The 9901 must be in the interrupt mode to mask interrupt inputs; it must be in the clock mode to use the internal clock.

NOTES	SELECT BIT	S0 S1 S2 S3 S4	PIN NO.	PIN FUNCTION WHEN BEING READ BY A CRU INSTRUCTION		PIN FUNCTION WHEN BEING SET OR "WRITTEN TO" BY A CRU INSTRUCTION	
9901	MODE			INTERRUPT	CLOCK	INTERRUPT	CLOCK
Base Address	0 (control bit)	0 0 0 0 0		0	1	0	1
	1	0 0 0 0 1	17	$\overline{\text{INT}}$ 1	CLK 1	MASK 1	CLK 1
	2	0 0 0 1 0	18	$\overline{\text{INT}}$ 2	CLK 2	MASK 2	CLK 2
	3	0 0 0 1 1	9	$\overline{\text{INT}}$ 3	CLK 3	MASK 3	CLK 3
	4	0 0 1 0 0	8	$\overline{\text{INT}}$ 4	CLK 4	MASK 4	CLK 4
	5	0 0 1 0 1	7	$\overline{\text{INT}}$ 5	CLK 5	MASK 5	CLK 5
	6	0 0 1 1 0	6	$\overline{\text{INT}}$ 6	CLK 6	MASK 6	CLK 6
	7	0 0 1 1 1	*34	$\overline{\text{INT}}$ 7	CLK 7	MASK 7	CLK 7
	8	0 1 0 0 0	*33	$\overline{\text{INT}}$ 8	CLK 8	MASK 8	CLK 8
	9	0 1 0 0 1	*32	$\overline{\text{INT}}$ 9	CLK 9	MASK 9	CLK 9
	10	0 1 0 1 0	*31	$\overline{\text{INT}}$ 10	CLK 10	MASK 10	CLK 10
	11	0 1 0 1 1	*30	$\overline{\text{INT}}$ 11	CLK 11	MASK 11	CLK 11
	12	0 1 1 0 0	*29	$\overline{\text{INT}}$ 12	CLK 12	MASK 12	CLK 12
	13	0 1 1 0 1	*28	$\overline{\text{INT}}$ 13	CLK 13	MASK 13	CLK 13
	14	0 1 1 1 0	*27	$\overline{\text{INT}}$ 14	CLK 14	MASK 14	CLK 14
	15	0 1 1 1 1	*23	$\overline{\text{INT}}$ 15	ΔINTREQ	MASK 15	$\overline{\text{RST}}$ 2
I/O Ports → Address	16	1 0 0 0 0	38	P0 INPUT		P0 OUTPUT	
	17	1 0 0 0 1	37	P1 INPUT		P1 OUTPUT	
	18	1 0 0 1 0	26	P2 INPUT		P2 OUTPUT	
	19	1 0 0 1 1	22	P3 INPUT		P3 OUTPUT	
	20	1 0 1 0 0	21	P4 INPUT		P4 OUTPUT	
	21	1 0 1 0 1	20	P5 INPUT		P5 OUTPUT	
	22	1 0 1 1 0	19	P6 INPUT		P6 OUTPUT	
	23	1 0 1 1 1	*23	P7 INPUT		P7 OUTPUT	
	24	1 1 0 0 0	*27	P8 INPUT		P8 OUTPUT	
	25	1 1 0 0 1	*28	P9 INPUT		P9 OUTPUT	
	26	1 1 0 1 0	*29	P10 INPUT		P10 OUTPUT	
	27	1 1 0 1 1	*30	P11 INPUT		P11 OUTPUT	
	28	1 1 1 0 0	*31	P12 INPUT		P12 OUTPUT	
	29	1 1 1 0 1	*32	P13 INPUT		P13 OUTPUT	
	30	1 1 1 1 0	*33	P14 INPUT		P14 OUTPUT	
	31	1 1 1 1 1	*34	P15 INPUT		P15 OUTPUT	

*COMMON
Δ INVERTED FROM INTREQ

Figure 23. 9901 Select Bit Assignments

Interrupt Mode

Select bit outputs 1 through 15 become MASK bits 1 through 15 when writing to these bits to enable (MASK = 1) or disable (MASK = 0) interrupts. Enabled interrupts received on the inputs will be decoded by the prioritizer and encoder of *Figure 15.*

Clock Mode

To set or read the self-contained clock, the 9901 must be in the clock mode. Using the CRU, the clock is set to a total count by writing a value to select bits 1 through 14.

Reading the clock is accomplished by a CRU instruction to read select bits 1 through 14. Another read instruction without switching the 9901 out of the clock mode will read the same value.

The clock is reset by writing a zero value to the clock or by a system reset.

In the clock mode, select bits 1 through 14 become CLK bits 1 through 14.

Data Inputs and Outputs

Select bits 16 through 31 are used for data inputs and outputs. All I/O pins are set to the input mode by a reset. To set a select bit as an output, just write data to that pin. The data will be latched and can be read with a CRU read instruction without affecting the data. Once an I/O port is programmed to be an output, it can only be programmed as an input by a hardware or software reset. This can be done two ways.

1. Receiving a hardware reset, $\overline{\text{RESET}}$.
   (Operating the RESET switch on the microcomputer.)
2. Writing a "0" to select bit 15 of the 9901 while in the clock mode will cause a software $\overline{\text{RST}}$2 and force all I/O ports to the input mode.

The status of the 9901 can be evaluated by checking (reading) the control bit. Testing select bit 15 in the interrupt mode can indicate if an interrupt has been received. If one has, INTREQ will be high because $\overline{\text{INTREQ}}$ is low.

After a hardware $\overline{\text{RESET}}$, or a software reset $\overline{\text{RST}}$2, all interrupts $\overline{\text{INT1}}$ through $\overline{\text{INT15}}$ are disabled, all I/O ports will be in the input mode, the code on IC0-IC3 will be 0000, $\overline{\text{INTREQ}}$ will be high and the 9901 will be in the interrupt mode.

Examples of Programming

Setting the Control Bit

If the interrupt and clock modes of the 9901 are to be controlled, load the base address in WR12 (100_{16} for 9901 on microcomputer board) and set select bit zero to the respective value:

```
LI   R12,>100          LOADS>100 INTO WR12
SBZ  0                 9901 TO INTERRUPT MODE
SBO  0                 9901 TD CLOCK MODE
```

Enabling or Disabling Interrupt Level

Interrupt levels are enabled or disabled by setting the MASK to a "1" or a "0" value, respectively. As an example, after a reset, the 9901 would be in the interrupt mode. Now interrupts 2, 5, 6 and 8 are to be enabled. The instruction:

```
LDCR R2,9
```

will do this as shown in *Figure 24.* The contents of workspace register 2, 0164_{16} from bit 15 thru 7 are read into select bits 0 thru 8 to enable interrupt levels 2, 5, 6 and 8. Of course, WR12 had to be loaded with the software base address using a

```
LI     R12,>100
```

instruction, as an example, and WR2 would have been loaded in a similar fashion.

In like fashion, the same levels could be disabled by writing "0" to bits 2, 5, 6 and 8 with an LDCR instruction, or programming a software $\overline{\text{RST2}}$, or by using the single bit CRU instructions.

▶9

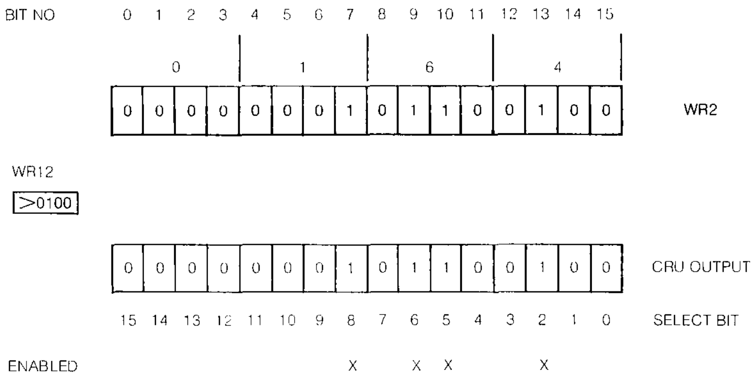


Figure 24. Enabling Interrupt Levels 2, 5, 6 and 8 with an LDCR Instruction

For example,

SBZ 2
SBZ 5
SBZ 6
SBZ 8

would set each bit to a "0". Previously WR12 was set to 0100_{16} to reference the 9901 on the microcomputer module.

Setting the Output Bits

Similar single bit or LDCR CRU instructions can be used to set the output bits.

LDCR R2, 0 would read out the value of WR2 to the output pins P_0 through P_{15} (the 0 in the LDCR R2, 0 means all 16 bits will be written to the output). WR12 has previously been loaded with 0120_{16}. This is shown in *Figure 25.*

A routine of loading 9901 I/O INPUTS and storing 9901 I/O OUTPUTS with a 743 KSR terminal would look like the following, after pressing the RESET toggle switch on the microcomputer module and a carriage return on the terminal:

```
TIBUG REV A
?M FE00 (CR)

 ADDRESS          OP CODE           MNEMONIC           COMMENT
FE00=XXXX        02E0    (SP)     LWPI  >FF20        ;WP=>FF20
FE02=XXXX        FF20    (SP)
FE04=XXXX        020C    (SP)     LI  R12,>120       ;9901 SOFTWARE BASE
                                                      ADDRESS = >120
FE06=XXXX        0120    (SP)
FE08=XXXX        0200    (SP)     LI  R0,>F0F0       ;CRU DATA
FE0A=XXXX        F0F0    (SP)
FE0C=XXXX        3000    (SP)     LDCR  R0,0          LOAD 9901 I/O
                                                      PORTS WITH R0
FE0E=XXXX        3400    (SP)     STCR  R0,0          STORE 9901 I/O
                                                      PORTS IN R0
FE10=XXXX        0460    (SP)     B  @>80            ;RETURN TO TIBUG
FE12=XXXX        0080    (CR)
?
```

The XXXX shown are don't care contents at the respective memory addresses which are changed as the op codes are entered. (SP) is a space bar command and (CR) is a carriage return.

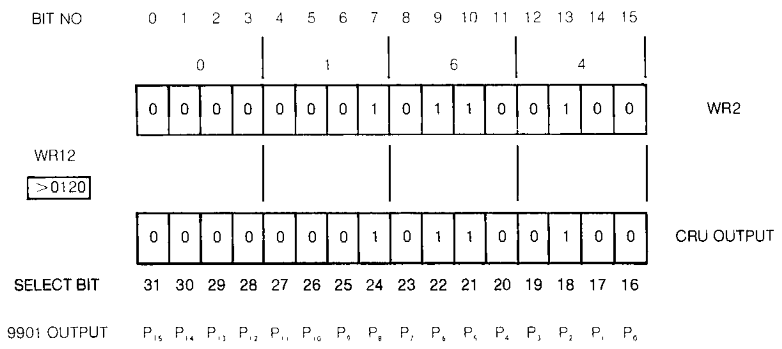


Figure 25. Output From WR 2 with LDCR Instruction

Programming the 9901 Clock

In *Figure 9,* the clock function of the 9901 was described. The clock register must be loaded with a value to set its total count and enable the clock. When the register is decremented to zero, it generates a level 3 interrupt (INT 3) as the elapsed time signal.

Access is gained to the clock by setting select bit zero to a "1" which puts the 9901 in the clock mode. All select bits 1 thru 15 are then in the clock mode and become the access for setting the clock count. CLK bit 15 is used for software reset. Therefore, the clock count is set by the value on select bits 1 through 14. An example is shown in *Figure 26.* The maximum value that can be loaded into 14 bits (all ones) would be 16,383. The rate at which the clock decrements the value is f(ϕ)/64. If f is 3 MHz, then the rate is approximately 46,875 Hz. The time interval is equal to the value in the clock register times 1/46,875. With the maximum value, the maximum interval is 349 milliseconds.

If 25 millisecond intervals are required, then the clock register would have to be loaded with 46,875 × 0.025 = 1172. This is equivalent to 0494_{16}. The least significant bit of the register value must be a 1 to set the control bit, therefore 0494_{16} is moved over a bit position and the register is loaded with 0929_{16}. A LDCR instruction is used for loading the value and the sequence of steps is shown in *Figure 26.*

▶9

The software is as follows:

```
LI  R12,>0100         ;SET 9901 ON MODULE SOFTWARE ADDRESS=>0100
LI  R1,>0929          ;LOAD CLOCK VALUE INTO R1, SET CLOCK MODE
LDCR  R1,15           ;MOVE TIMER VALUE AND CONTROL BIT TO 9901
```

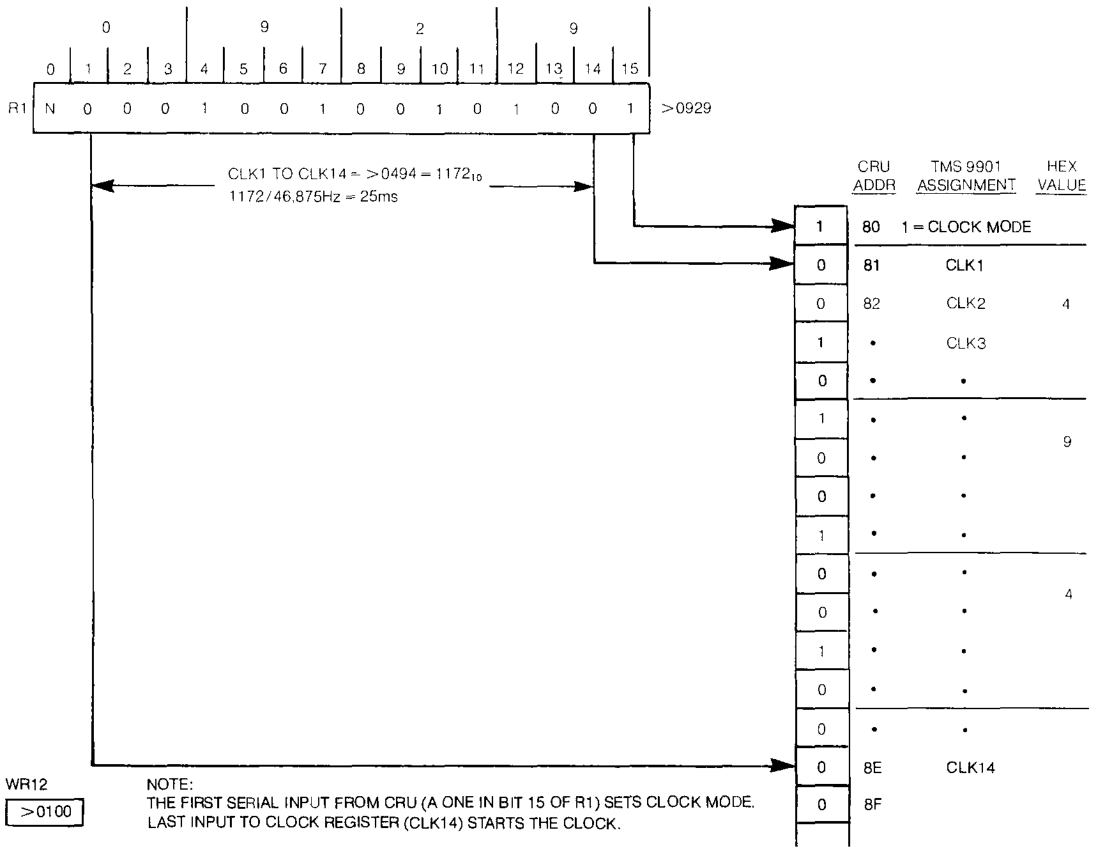

0
9
2
9
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
R1
N 0 0 0 1 0 0 1 0 0 1 0 1 0 0 1
>0929
CLK1 TO CLK14 = >0494 = 1172_{10}
1172/46,875Hz = 25ms
CRU ADDR
TMS 9901 ASSIGNMENT
HEX VALUE
80 1 = CLOCK MODE
81 CLK1
82 CLK2
CLK3
8E CLK14
8F
4
9
4
WR12
>0100
NOTE:
THE FIRST SERIAL INPUT FROM CRU (A ONE IN BIT 15 OF R1) SETS CLOCK MODE.
LAST INPUT TO CLOCK REGISTER (CLK14) STARTS THE CLOCK.

Figure 26. Enabling and Triggering TMS 9901 Interval Timer

Enabling Clock Interrupt

When the clock decrements to zero, a level 3 interrupt is given. The interrupt level 3 mask needs to be enabled on the 9901 and the 9900 CPU. The interrupt mask on the 9901 is enabled by setting the control bit to a logical "0" (interrupt mode) and then setting select bit 3 to a "1" (write a "1" to bit 3). The interrupt mask on the 9900 is enabled by loading the appropriate value (in this case, 3) into the interrupt mask. When 3 is loaded into the 9900 with a LIMI 3 instruction, all higher priority levels are also enabled.

The software is:

```
LI  R12,>0100          ;SET BASE ADDRESS TO 9901 ON BOARD, >0100
SBZ  0                 ;9901 TO INTERRUPT MODE
SBO  3                 ;ENABLE INTERRUPT 3 AT 9901
LIMI  3                ;LOAD 9900 INTERRUPT MASK
```

Putting Some Pieces Together

Some of the pieces can now be combined to provide a larger program. It looks like this:

```
        LI  R12,>0100        ;SET SOFTWARE BASE ADDRESS OF 9901=0100
        CLR  R0              ;INITIALIZE INTERRUPT INDICATOR, R0 SET TO ZERO
        LI  R1,>0929         ;CLOCK COUNT 0494 AND CLOCK MODE IN R1
        LDCR  R1,15          ;SET CLOCK COUNT ENABLE TIMER
        SBZ  0               ;9901 TO INTERRUPT MODE
        SBD  3               ;ENABLE INT 3 AT 9901
        LIMI  3              ;LOAD 9900 INTERRUPT MASK
LODP 2  CI  R0, >FFFF        ;HAS INT 3 OCCURED?
        JNE  LOOP 2          ;IF NO, GO TO LOOP 2
```

When the timer gives an interrupt 3, a context switch occurs; the interrupt 3 vector PC points to $FF88_{16}$ which contains an instruction to get to the interrupt routine:

```
        B  @CLKINT           ;BRANCH TO INTERRUPT ROUTINE IDENTIFIED BY CLK INT
```

The branch then takes the program to:

```
CLKINT  LI  R12,>0100        ;SET SOFTWARE BASE ADDRESS OF 9901=0100
        SBZ  3               ;DISABLE INTERRUPT 3
        SETO  *R13           ;SET PREVIOUS R0 TO FFFF
        RTWP                 ;RETURN TO PROGRAM
```

Thus, if an interrupt 3 has not occured, the program remains in Loop 2 until it does. When $\overline{\text{INT}}$ 3 occurs a context switch to the interrupt subroutine causes R0 to be changed from all zeros to all ones. R0 will now equal $FFFF_{16}$ and the program proceeds to the step after JNE Loop 2, which, as will be seen later, is a count down.

FROM BASIC CONCEPTS TO PROGRAM

As with the Chapter 3 application, converting the idea to program starts with solidifying the basic concept, then developing acceptable flow charts, and then programming the algorithm for the problem solution. As with hard-wired logic design, the place to start is with a block diagram. The one used in *Figure 6* will be expanded with a bit more detail and will be the concept diagram *(Figure 27)*.

The terminal, the microcomputer module and the interface modules with their respective inputs and outputs will constitute the system. Later on the TM900/310 module will be added to show the I/O expansion capability. This will only involve plugging the interface modules into one of the additional 9901 outputs on the 310 board (P4 in this case) and changing the CRU base address to select the chosen 9901. It will be assumed that the power and all interconnections have also been made through P1 to the microcomputer and 310 module as shown in *Figure 27.* There is a special power supply required for supplying the interface modules. This is the +8V shown in *Figure 27.* 110Vac is supplied separately for the terminal and the industrial level voltages of 12 volts dc and 110Vac are supplied separately, as they would be in a user facility.

The physical arrangement of the interface modules is important to the program for the problem solution. Therefore, I/O positions 0 thru 7 are identified. Positions 0 thru 3 are input positions; positions 4 thru 7 are output positions. Signals received on input position 0 will cause reaction at output position 4. Correspondingly for input 1 and output 5, input 2 and output 6, and input 3 and output 7. Thus, the program will be written to sense input 1 and set output 5 to correspond.

Switches S1 through S4 represent industrial level input voltages, either dc or ac. Lights L1 and L3 represent industrial dc loads; L2 and L4 represent industrial ac loads.

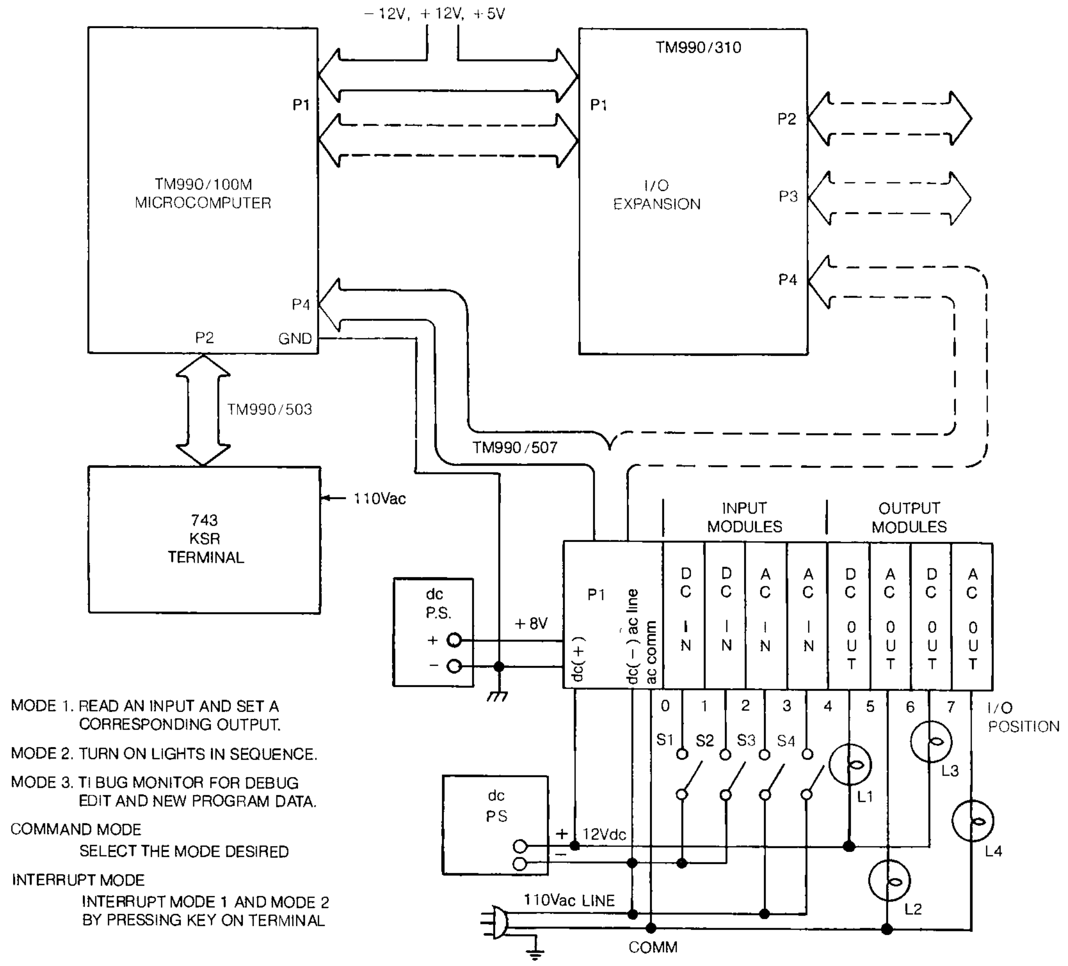


Figure 27. Concept Flow Diagram

Flow Charts for the Program

Software design is really little different from hardware design in the execution of good engineering practice.

The task from overall concept stage is divided into subsystems — in the case of software, subprograms or subroutines. *Figure 28* identifies subprograms for the extended application which are detailed in flow charts so that basic functions can be identified.

The flow charts are separated according to the functions that are to be implemented. Operation in Mode 1 simulates sensing four industrial level inputs 0 through 3 and reacting to these inputs by providing output voltages to four corresponding loads, 4 through 7. The flow chart identifies that inputs will be sensed and a corresponding output will be set to match the input state or value.

The four output loads, in this case light bulbs, will be turned on and off in sequence and held in each of these states for a set time (variable by the program). This is Mode 2 operation. The flow chart shows the major functions. After all four lights are turned off and on, the sequence starts over. The clock in the 9901 will be used to provide the time interval.

There is an operating Mode 3 but it will be contained in the mode called the COMMAND Mode. In Mode 3 the operation of the system is under the control of the TIBUG Monitor which is contained in the 1K words of EPROM resident in the microcomputer. It is used for inputting the original program and editing and changing the program as the need may be.

The flow chart for the Command Mode starts with initial setup of the system. Certain registers and certain locations in memory are loaded with data used throughout the program. A print-out of general information and specific instructions follows. Since the user will make a choice, instructions identify that a one (1) key is to be pressed on the terminal to operate in Mode 1; a two (2) key to operate in Mode 2; and a Q for Mode 3. The character pressed by the user is then examined and the appropriate operating mode selected. If none of the operating mode characters are received the system waits in the command mode until one is received.

On the flow chart for the COMMAND mode A and B connect with the respective points on the MODE 1 and MODE 2 flow charts.

Recall that the system is to have a provision for the user to command an escape from the continuous operation in Mode 1 or Mode 2. This happens by interrupting Mode 1 or Mode 2 operation by pressing a key on the terminal. The first blocks in the flowcharts of MODE 1 and MODE 2 provide the means for accomplishing the interrupt. When a key is pressed on the terminal, this initiates an interrupt signal output from the 9902. This interrupt must be enabled to pass to the 9901 and the 9900 so that it will cause the return to the COMMAND MODE. The generation of the signal in the 9902 is flowcharted under the heading INTERRUPT MODE of *Figure 28*.

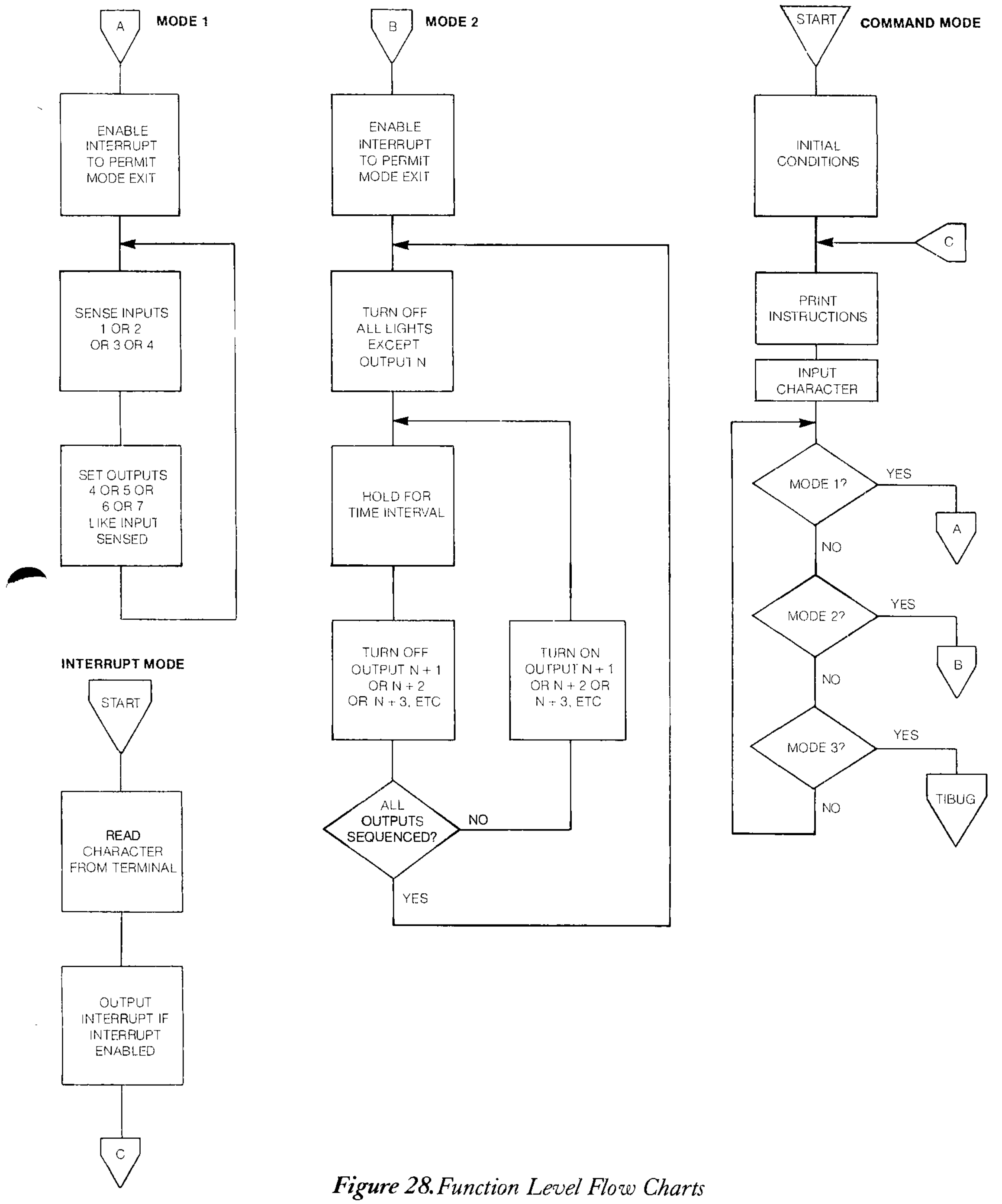

MODE 1
A
ENABLE INTERRUPT TO PERMIT MODE EXIT
SENSE INPUTS 1 OR 2 OR 3 OR 4
SET OUTPUTS 4 OR 5 OR 6 OR 7 LIKE INPUT SENSED
INTERRUPT MODE
START
READ CHARACTER FROM TERMINAL
OUTPUT INTERRUPT IF INTERRUPT ENABLED
C
MODE 2
B
ENABLE INTERRUPT TO PERMIT MODE EXIT
TURN OFF ALL LIGHTS EXCEPT OUTPUT N
HOLD FOR TIME INTERVAL
TURN OFF OUTPUT N + 1 OR N + 2 OR N + 3, ETC
TURN ON OUTPUT N + 1 OR N + 2 OR N + 3, ETC
ALL OUTPUTS SEQUENCED?
NO
YES
COMMAND MODE
START
INITIAL CONDITIONS
C
PRINT INSTRUCTIONS
INPUT CHARACTER
MODE 1?
YES
A
NO
MODE 2?
YES
B
NO
MODE 3?
YES
TIBUG
NO

Figure 28. Function Level Flow Charts

Writing the Program

Memory Space

All elements are now in place to write the program. First, it is necessary to decide what locations are to be used in memory for the program, for the workspace and for data. Refer to *Figure 29.*

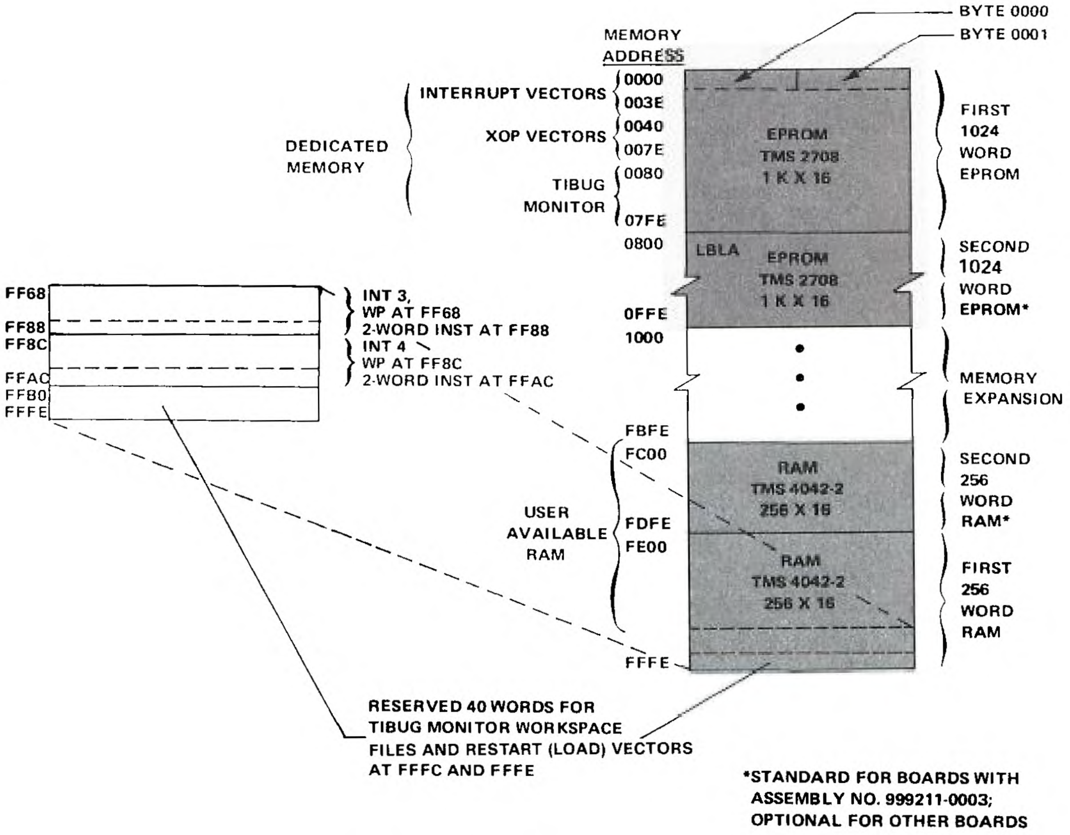
For this application more memory space is required than for Chapter 3's First Encounter. Thus, additional RAM units are installed on the microcomputer board at locations U33, U35, U37 and U39 (4042 Units). This expands the available RAM space to $FC00_{16}$ and this is the location for the start of the program.

Incidentally, while available memory is being discussed, note the address of the TIBUG monitor, 0080_{16}. This memory location must be referenced when returning to the TIBUG Monitor in Mode 3. The TIBUG workspace located at $FFB0_{16}$ has already been discussed. This space must be reserved.

One more point — the second 1K of EPROM starting at location 0800_{16} will be populated with the Line-by-Line Assembler (LBLA) resident in EPROM. This will be used for assembly of the program. The socket locations on the board are U43 and U45 and the product number is TM990/402-1. Normally, the LBLA would start assemblying at address $FE00_{16}$, however, by using a /FC00 command the start location is changed to $FC00_{16}$.

The Command Mode

A more complete flow chart is shown in *Figure 30* for the Command Mode. The program begins with initialization of registers. When writing the first draft of the program, labels are used for ease of writing. For later drafts and when a LBLA is used, the labels are replaced with actual addresses. INPUT1 will be the label for the start of Mode 1. BLINKR will be the label for the start of Mode 2. COMODE labels the message that asks the user to select the mode.



DEDICATED MEMORY

ADDRESS (HEX)	PURPOSE
0000-0003	Level zero interrupt vector ($\overline{\text{RESET}}$)
000C-000F	$\overline{\text{INT3}}$ vectors (TMS 9901 timer)
0010-0013	$\overline{\text{INT4}}$ vectors (TMS9902 timer)
0040-0047	Vectors for XOP's 0 and 1 (Microterminal I/O)
0060-007F	Vectors for XOP's 8 to 15 (TIBUG utilities)
0080-07FF	TIBUG monitor
FFB0-FFFB	Four overlapping monitor workspaces
FFFC-FFFF	Restart (load) vector

Figure 29. Memory Map with Fully Populated 990/100M-1 Module

The Command Mode program is as follows:

```
CDUNT        +>929              ;SET UP 9901 CLDCK
BASE1        +>100              ;SET UP 9901 CRU BASE
BASE2        +>120              ;SET UP 9901 I/D BASE
START        LWPI  >FF20        ;SET WP AT FF20
             LI  R1,>1E00       ;SBZ DP CDDE TD R1
             LI  R2,>1D00       ;SBD DP CDDE TD R2
             LI  R3,>1F00       ;TB OP CODE TO R3
             XDP  @MSG1, 14     ;PRINT HEADER @MSG1
CDMDDE       XDP  @MSG2, 14     ;ASK FDR MDDE WITH MSG2
             XDP  R7, 11        ;READ CHAR FRDM TER TD R7
             CI  R7,>3100       ;IS CHAR A 1?
             JEQ  INPUT1        ;IF YES GD TD MDDE 1
             CI  R7,>3200       ;IS CHAR A 2?
             JEQ  BLINKR        ;IF YES TD TD MDDE 2
             CI  R7,>5100       ;IS CHAR A Q?
             JNE  CDMDDE        ;IF ND KEEP LDDPING
             B  @>80            ;IF YES GO TO TIBUG
```

To initialize registers, the values for the TMS 9901 clock interval, TMS 9901 CRU software base address and TMS 9901 I/O software base address are loaded directly into memory spaces by using a (+) in front of the data. 0929_{16} is placed in the 9901 for a 25ms interval. Recall that the module 9901 has a base address of 0100_{16} for select bit zero and 0120_{16} so that select bit 16 activates P0 when input or output bit 0 is addressed, as discussed previously. Note that the workspace is set up at $FF20_{16}$.

The machine codes for SBZ, SBO and TB are loaded into workspace registers one, two and three, respectively. As discussed previously, an XOP 14 is used to print the header and instructions for use of the program. The messages are labeled with MSG1 and MSG2 and are located at the end of the program and will be discussed later. Next an XOP is used to read a character from the terminal and load the ASCII code into R7. This is then compared with the ASCII codes for the number one, two and the letter Q to determine the character. Depending on what character is received, the program jumps to the proper area in memory to execute the correct mode of operation. The entry point to the TIBUG monitor is 0080_{16} and a branch to this location will execute the monitor.

Mode 1 Operation

Figure 31 shows the flow chart for Mode 1 Operation. The label INPUT1 begins the operation. The first function sets up the system so that the 9902 will generate an interrupt when a received character fills the receiver buffer (RBRL = 1). Recall that the interrupt generated by the 9902 must be enabled by making RIENB = 1. This is accomplished by making the 9902 select bit 18 equal to "1". The enabled interrupt from the 9902 is wired to the $\overline{\text{INT}}4$ input of the 9901. Thus, as previously discussed, level 4 interrupts must be enabled both at the 9901 and the 9900.

The software looks like this:

```
INPUT1    RSET               ; PUT 9901 INTO INPUT MODE
          LIMI  4            ; ENABLE 9900 INT1-INT4
          LI  R12,>80        ; LOAD R12 W/9902 BASE ADDR
          STCR  R7,0         ; CLEAR 9902 RCV BUFFER
          SBO  18            ; ENABLE 9902 RCV INT
          MOV  @BASE1,R12    ; SET 9901 BASE ADDR TO  >100
          SBO  4             ; ENABLE 9902 INT AT 9901
```

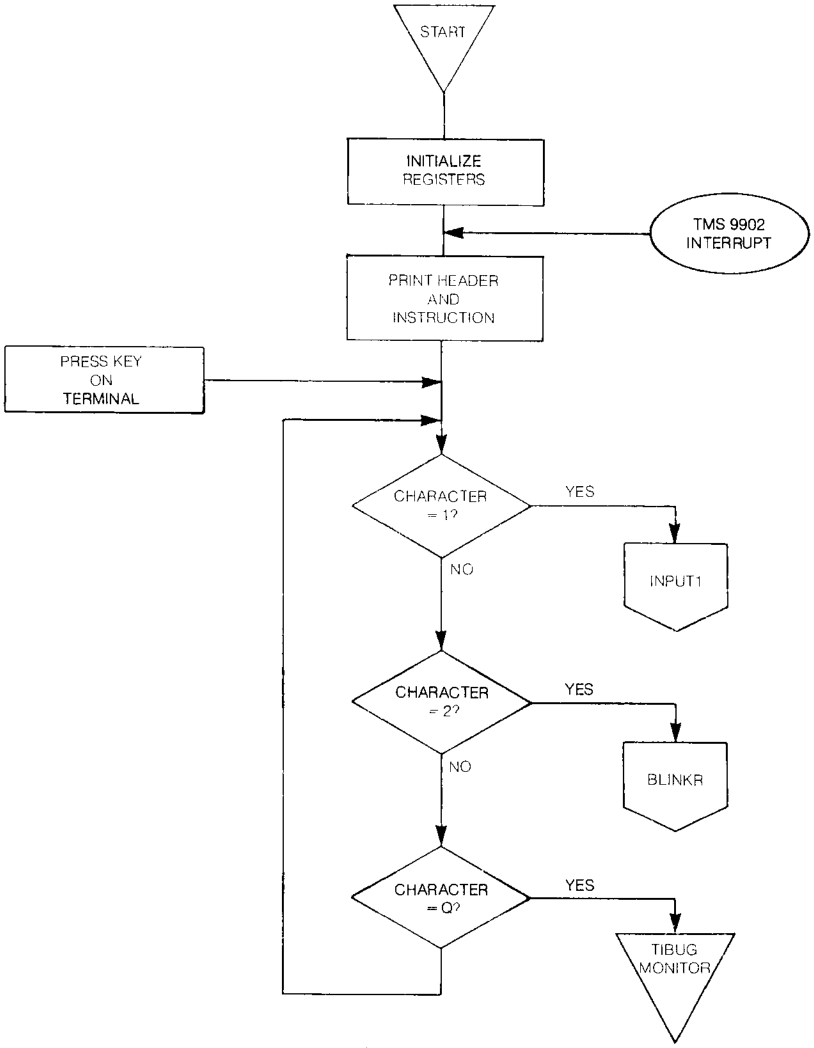


Figure 30. Command Mode

First the 9901 is reset to put it into the input mode. Then the 9900 interrupt mask is set to 4 to allow interrupts 1 thru 4 to be acknowledged. To enable select bit 18 of the 9902, the software base address is loaded into WR12 and an SBO 18 instruction sets the bit to "1" for the enable. The 9902 receiver buffer is read into R7 with the STCR instruction which resets the buffer for receipt of a character. WR12 is set with the software base address for the 9901, and then select bit 4 is set to a "1". These steps enable the 9901 interrupt level 4 to clear the complete path for generating an interrupt when a character is received from the terminal.

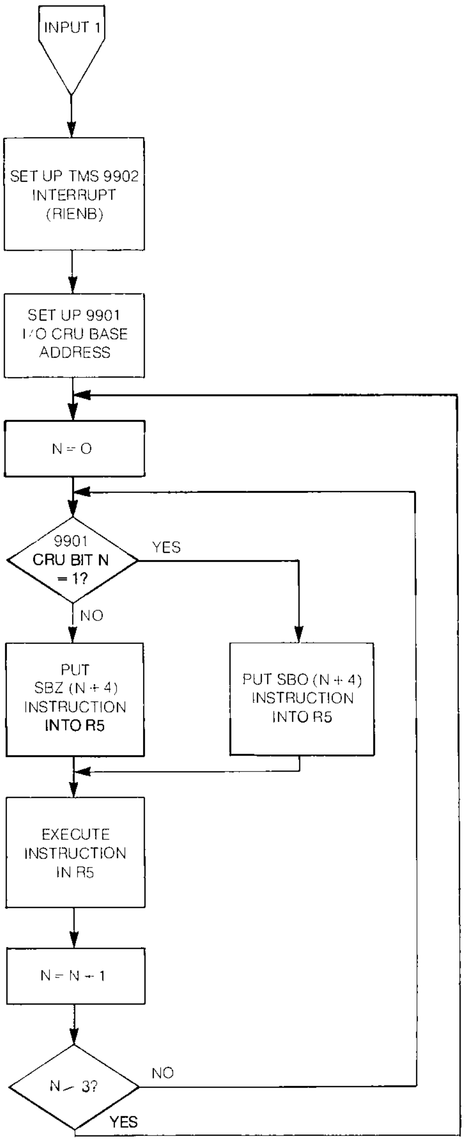

INPUT 1
SET UP TMS 9902 INTERRUPT (RIENB)
SET UP 9901 I/O CRU BASE ADDRESS
N = O
9901 CRU BIT N = 1?
YES
NO
PUT SBZ (N + 4) INSTRUCTION INTO R5
PUT SBO (N + 4) INSTRUCTION INTO R5
EXECUTE INSTRUCTION IN R5
N = N + 1
N > 3?
NO
YES

Figure 31. Mode 1 Operation

Checking the Inputs—Setting the Outputs

Figure 31 shows that with N = 0 the CRU is testing the zero input bit of the 9901. If it is a "1", then the N + 4 bit (I/O bit 4) will be set to a "1" to correspond. One will be added to N (0 + 1 = 1), which will be less than 3 and the cycle is repeated: the second time with N = 1, the next time with N = 2 and the next with N = 3. With N = 3, N + 1 will be greater than three and everything is reinitialized and the sequence starts over with input bit zero again. So the procedure is to check each input bit and set the corresponding output bit. The software is as follows:

```
          MOV  @BASE2,R12      ; SET 9901 BASE AOOR TO >120
INIT1     CLR  R4              ; R4 CONTAINS CRU BIT TO BE
                               ; TESTEO
INOEX1    MOV  R4,R5           ; MOVE CRU BIT TO R5
          SOC  R3,R4           ; R4 CONTAINS TB INST (R3)
          X  R4                ; EXECUTE TB SPECIFIEO BY R4
          JEQ  HIGH            ; IF CRU BIT=1 GO TO HIGH
LOW       MOV  R5,R4           ; RELOAO CRU BIT INTO R4
          AI  R5,>4            ; SHIFT CRU BIT OVER BY 4
          SOC  R1,R5           ; R5 CONTAINS SBZ OP COOE (R1)
XECUTE    X  R5                ; EXECUTE OP COOE SPECIFIEO BY R5
          INC  R4              ; INCREMENT TO NEXT CRU BIT
          CI  R4,>3            ; IS CRU BIT >3?
          JGT  INIT1           ; IF YES REINITIALIZE
          JMP  INOEX1          ; START TESTING NEXT CRU BIT
HIGH      MOV  R5,R4           ; RELOAO CRU BIT INTO R4
          AI  R5,>4            ; SHIFT CRU BIT OVER 4
          SOC  R2,R5           ; R5 CONTAINS SBO OP COOE (R2)
          JMP  XECUTE          ; GO EXECUTE SBO INST
```

Input bits 0-3 correspond to output bits 4-7 respectively. R4 contains the value of the select bit to be tested (the program starts with bit zero). R4 is moved to R5 to preserve the contents of R4. R3 contains the machine code for TB. Actually it contains the machine code for the instruction TB 0 (Test bit 0). By doing a set ones correspondence (SOC) between R3 and R4, the machine code for the TB instruction is combined with the value of the select bit to be tested so that R4 contains the instruction — "test the select bit previously specified by R4." More specifically, R4 = TB (R4).

An X of R4 will execute this instruction. Using this procedure allows R4 to contain the bit position separate from the TB instruction which is in R3. The bit position in R4 or R5 can also be combined with the SBO and SBZ op codes located in R2 and R1 to allow execution of the SBO or SBZ instructions on the select bits specified by R4 or R5. The procedure is the same as for the TB instruction.

If the bit tested is a zero, R4 is reloaded from R5 with the original value of the select bit to be tested, which is still in R5. R5 plus 4 is combined with R1 using a SOC R1, R5 instruction. The selected output bit will be set to zero when the resulting SBZ instruction in R5 is executed. Thus, an N + 4 output is set to zero, if the corresponding N bit was a zero.

R4 is incremented to the next bit and is tested to determine if its value is greater than 3. When it is not, the program jumps to label INDEX1 and tests the next bit in the same sequence as the first and sets the corresponding output bit. Now, suppose this bit is a "1" instead of a "0" as for the preceding bit. The program jumps to the label HIGH, reloads R4, adds 4 to R5, and now executes an SOC R2, R5 to set the N + 4 output to one when the SBO instruction in R5 is executed.

When input bit 3 is tested, the test of R4 + 1 will show its value is greater than 3 and the program is reinitialized and the procedure starts over. To exit the loop, any key on the keyboard is pressed which produces a level 4 interrupt. The level 4 interrupt comes from the 9902 and the system enters the command mode as shown in *Figure 30.*

Mode 2 Operation *(Figure 32)*

Mode 2 operation sequences the loads simulated by light bulbs. The flowchart is shown in *Figure 32.* It has a time interval of 25ms set up by the 9901 real time clock. A program loop multiples the 25ms times R6 to obtain the total time interval; with R6 = 4, each total time interval is 100ms. The time interval can also be varied by changing the initial value 0929_{16} set into the clock register of the 9901. The value in R4 determines the number of light bulbs (loads) that are going to be turned on, held for 100ms, turned off, and started through the sequence again. As with mode 1, pressing a key on the terminal causes a return to the Command Mode.

It is worthy to note, even though the 9901 is in the input mode when reset, outputs 4,5,6 and 7 are such that all light bulbs are on. Thus, the function of turning off outputs 5, 6 and 7 and leaving 4 on starts the program after the CRU base address is set. In actual industrial applications it may be necessary to put additional inverters between the output of the microcomputer and the 5MT modules so that the reset condition has all loads off.

Recall that when the 9901 clock register is decremented to zero it puts out a $\overline{\text{INT}}3$ signal. This interrupt causes a context switch to occur and sets the old workspace R0 to $FFFF_{16}$. When this happens the time interval has ended.

Interrupt 4 from TMS 9902

The software for Mode 2 starts as follows to set up the interrupt 4 from the 9902:

```
BLINKR      RSET          ; SET 9901 TO THE INPUT MODE
            LIMI  4       ; ENABLE 9900 INT1-INT4
            LI  R12,>80   ; SET UP 9902 BASE ADDR
            STCR  R7,0    ; CLEAR 9902 RCV BUFFER
            SBO  18       ; ENABLE 9902 RCV INT
```

The reset at BLINKR sets the 9901 to the input mode and turns on the loads on outputs 4, 5, 6 and 7.

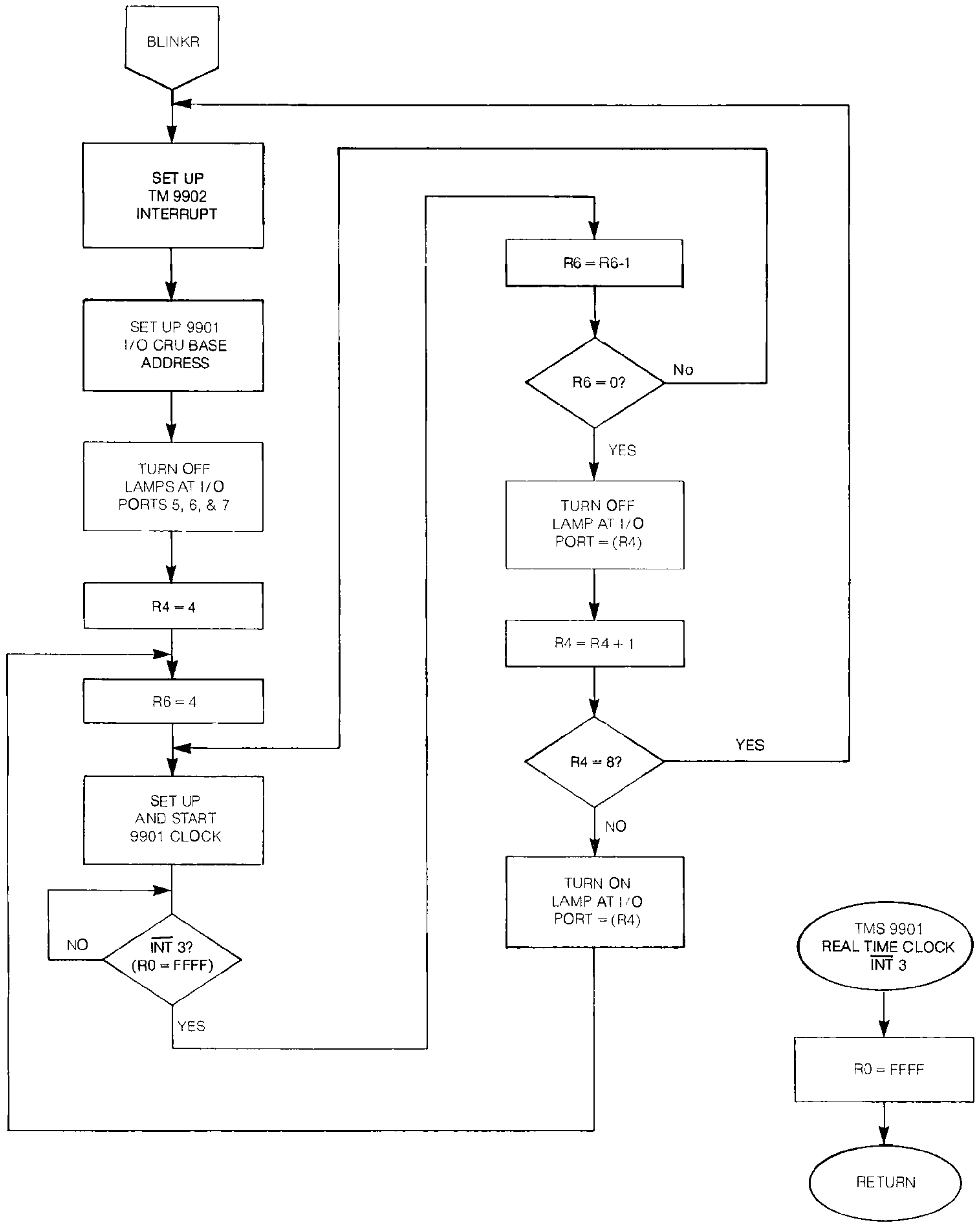

BLINKR
SET UP TM 9902 INTERRUPT
SET UP 9901 I/O CRU BASE ADDRESS
TURN OFF LAMPS AT I/O PORTS 5, 6, & 7
R4 = 4
R6 = 4
SET UP AND START 9901 CLOCK
INT 3? (R0 = FFFF)
NO
YES
R6 = R6-1
R6 = 0?
No
YES
TURN OFF LAMP AT I/O PORT = (R4)
R4 = R4 + 1
R4 = 8?
YES
NO
TURN ON LAMP AT I/O PORT = (R4)
TMS 9901 REAL TIME CLOCK INT 3
R0 = FFFF
RETURN

Figure 32. Mode 2 Operation

The next 7 instructions after the 9901 software base address is set at 0120_{16} *are concerned* with turning off outputs 5, 6 and 7. These start with INT2 and continue through the next 6 instructions after LOOP 1.

```
                  MOV   @BASE2, R12        ; SET 9901 BASE ADDR=>120
INT2              LI   R4,>5               ; R4 CONTAINS CRU BIT POS 5
LOOP1             MOV   R4,R5              ; MOV POS 5 TO R5
                  SOC   R1,R5              ; R5 CONTAINS SBZ OP CODE (R1)
                  X   R5                   ; EXECUTE SBZ SPECIFIED BY (R5)
                  INC   R4                 ; R4=R4+1
                  CI   R4,>8               ; HAS CRU BIT 7 BEEN SET=0?
                  JNE   LOOP1              ; IF NO GO TO LOOP 1
```

Lamp 4 remains on.

Register 4 must now be loaded with the output position from which the sequence starts—in this case 4.

```
                  LI   R4,>4               ; SET OUTPUT BASE BIT
```

Timing Loop

R6 is set equal to 4 so that the overall time interval is 100ms. This starts the timing loop at INDEX2. The 5 instructions following TIMER set up the 9901 clock to count a 25ms interval and then cause a level 3 interrupt. Note that the 9901 must be put into the interrupt mode and the level 3 interrupt enabled. Since the 9902 interrupt signal comes in on interrupt level 4, it is convenient to enable it at this same time. The loop is such that it loops 4 times. Each loop is controlled by the interval timer of the TMS9901. The TMS9901 timer is set and started when loaded with the value at the label COUNT. The clock decrements until it hits zero and then it gives a level 3 interrupt. The interrupt service routine begins at $FF88_{16}$ as directed by the level 3 vector. It sets R0 to $FFFF_{16}$ and returns to the program. The program will be in a continuous loop (Loop 2) checking R0 for an indication that an interrupt has occured. When the time interval is complete, the I/O bit dictated by R4 is turned off. R4 is incremented and checked to see if it is equal to 8. If not, the I/O bit position of the new R4 is turned on and the sequence restarts. If $R4 + 1 = 8$, then the program jumps back to BLINKR and starts over causing R4 to be reset to 4 and to restart the sequence.

The software looks like this:

```
INDEX 2     LI   R6,>4             ; OVERALL LOOP COUNT=100ms
TIMER       MOV  @BASE1,R12        ; SET CRU BASE ADDR OF 9901=>100
            CLR  RO                ; INITIALIZE INT3 INDICATOR
            LDCR @COUNT,15         ; LOAD TIMER AND START COUNT
            SBZ  0                 ; 9901 TO INTERRUPT MODE
            SBO  3                 ; ENABLE INT3 AT 9901
            SBO  4                 ; ENABLE 9902 INT AT 9901
LOOP2       CI   RO,>FFFF          ; HAS INT3 OCCURRED?
            JNE  LOOP2             ; IF NO GO TO LOOP2
            DEC  R6                ; R6=R6-1
            JNE  TIMER             ; IF R6=0 GO TO TIMER
            MOV  @BASE2,R12        ; SET 9901 BASE ADDR=>120
            MOV  R4,R5             ; MOV CRU BIT TO R5
            SOC  R1,R5             ; (R5)=SBZ (R5)
            X    R5                ; EXECUTE SBZ SPECIFIED BY (R5)
            INC  R4                ; R4=R4+1
            CI   R4,>9             ; IS R4=9?
            JEQ  BLINKR            ; IF YES RESTART SEQUENCE
            MOV  R4,R5             ; R4=R5
            SOC  R2,R5             ; (R5)=SBO (R5)
            X    R5                ; EXECUTE SBO SPECIFIED BY (R5)
            JMP  INDEX2            ; RESTART TIMING CYCLE AT INDEX 2
```

9902 Interrupt Service Routing

This interrupt service routine is the one resulting from a level 4 interrupt generated by the 9902. It starts at INTREC. As discussed previously, when the interrupt occurs, the program counter points to $FFAC_{16}$, the reserved space, where it finds an instruction directing it to INTREC. This instruction looks like this:

```
ADDRESS     INSTRUCTION
FFAC        B @INTREC              ; GO TO INT4 SERVICE ROUTINE
```

The routine first disables the 9901 timer interrupt level 3, then disables the 9902 interrupt at the 9902 (Set select bit 18 = 0) and finally loads the address of COMODE into the old PC, so that when an RTWP (return with workspace pointer) is executed, the program returns to the command mode. The software is as follows:

```
INTREC      MOV  @BASE1,R12        ; SET 9901 BASE ADDR=>100
            SBZ  3                 ; DISABLE INT3 AT 9901
            SRL  R12, 1            ; SET BASE ADDR=>80 FOR 9902
            SBZ  18                ; DISABLE 9902 INT
            STOR R7, 0             ; READ 9902 RCV BUFFER (CLEARS)
            LI   R14, COMODE       ; LOAD ADDR OF COMODE INTO PC
            RTWP                   ; RETURN TO 5MT ROUTINE
```

9901 Clock Interrupt Service Routine

When the clock decrements to zero it generates a level 3 interrupt. The routine to service this interrupt starts at CLKINT. The level 3 interrupt context switch provides a new PC at $FF88_{16}$ which directs the program to CLKINT. This instruction looks like this:

```
ADDRESS   INSTRUCTION

FF88      B   @CLKINT             ; GO TO INT3 SERVICE ROUTINE
```

Here, after setting the software base address of the 9901 to 0100_{16}, $\overline{\text{INT3}}$ is disabled and R0 of the previous workspace is set to $FFFF_{16}$. A RTWP instruction then returns the processor to the interrupted routine.

The software is as follows:

```
CLKINT    LI   R12,>100           ; SET 9901 BASE ADDR
          SBZ   3                 ; DISABLE INT3 AT 9901
          SETO  *R13              ; SET PREVIOUS R0=>FFFF
          RTWP                    ; RETURN TO INTERRUPTED ROUTINE
```

Message Routines

The remaining routines that must be included in the program are the messages at MSG1 and MSG2. In order to program the message, a $ sign is used at the beginning of each line and each message is terminated with a zero byte. The ASCII code for a carriage return — line feed is $0D0A_{16}$ and is included in the instruction format.

Each character must be coded with the appropriate ASCII code and placed into bytes of memory. A typical example is shown; however, the individual character codes have not been listed. This can be seen on the LBLA listing.

```
MSGI      $5MT I/O DEMONSTRATION ROUTINE
          +>0D0A
          $MODE 1 — INPUTS 0-3 SWITCH OUTPUTS
          $4-7 RESPECTIVELY
          +>0D0A
          $MODE 2 — OUTPUTS 4-7 ARE SWITCHED SEQUENTIALLY
          +>000A
          $A Q RETURNS CONTROL TO THE TIBUG MONITOR
          +>0D0A
          $A CARRIAGE RETURN DURING MODE 1 OR 2
          $OPERATION RETURNS THE USER TO THE
          +>0D0A
          $CONTROL MODE
          +>0D0A
          +>0000
          +>0D0A
          $SELECT MODE 1, 2 or Q
          +>0D0A
          +>0000
```

▶9

SYSTEM OPERATION

With program in hand, it is time to connect the hardware to prove out the complete program. Refer to the block diagram of *Figure 27.*

The terminal and its cable have been previously connected to P2 of the microcomputer module. P1 has the same power supply connections as for Chapter 3 supplying −12V, +12V, +5V and ground. The full connections will be added to P1 to interface with P1 on the TM990/310 I/O expansion board. However, for now, operations will be only with the microcomputer and the 5MT I/O modules. Connection to the modules is made through the cable of *Figure 4* and P4 on the microcomputer and P1 on the 5MT43 module base. There is a separate wire from the J1 connector to provide +8 volts to the 5MT modules. This +8 volts must supply 0.6A worst case if all the positions in the 5MT43 base are populated. *This supply ground must be common with the microcomputer module ground and isolated from the +12V industrial control voltage supply ground.*

The +12V for the industrial control level voltages must supply 200mA. *This must have a minus terminal free of chassis ground, otherwise its case will be at ac line voltage when the 5MT I/O module ac power cord is connected.*

Light bulbs that are rated at 80 mA at 14 Vdc are used for the dc loads. Standard 110 Vac light bulbs and sockets are used for the ac loads. A separate ac power cord is connected to the 5MT43 base for the ac power. The industrial level power *(both dc and ac) is and must be isolated from the dc power for the microcomputer module and low-level logic +8V power source of the 5MT interface modules.*

A summary of the parts list and power supply requirements follows:

System Parts List

- TM990/100M-1 board
- TM990/310 48 I/O board (optional)
- 5MT43 base*
- 2 – 5MT11-A05L AC input modules*
- 2 – 5MT12-40AL AC output modules*
- 2 – 5MT13-D03L DC input modules*
- 2 – 5MT14-30CL DC output modules*
- 5MT interface cable-TM990/507
- 743 KSR terminal
- TM 990/503 cable assembly for Terminal
- 4 – TMS 4042-2 (or 2111-1) 256 x 4 RAM's

*In case your local distributor does not have these parts, the address from which they can be ordered is:

Industrial Controls Order Entry
M/S 12-38
34 Forrest St.
Attleboro, Mass. 02703
Phone: (617) 222-2800

- Line-by-line assembler TM990/402-1 (in two TMS 2708 EPROM's)
- Power supplies for Microcomputer and I/O Expansion (TM990/518)

Voltage	*REG*	*/100M Current*	*w/310 Module Current*
+5V	±3%	1.3A	2.1A
+12V	±3%	0.1A	0.1A
−12V	±3%	0.2A	0.2A

- Industrial Control Level Power Supplies

Voltage	*REG*	*Current*
+8Vdc	±5%	0.6A
+12Vdc	±5%	0.2A
110Vac		1A

- 4 Toggle switches, SPST
- 2 dc lamps and sockets (14V — 80mA)
- 2 ac lamps and sockets (130V — 30 W)
- Power cord
- 14 and 18 AWG insulated stranded wire

Equipment Hookup

Follow these steps in making the system interconnections;

Step 1 — Verify that the power supply connections to P1 are correct for −12V, +12V and +5V. Refer to *Figure 3-11* or to the TM990/100M user's guide *Figure 2-1.* Don't turn on any power supplies. It may be desirable to make all the connections from P1 of the TM990/100M to P1 of the TM990/310 at this time. Refer to *Table 6* for these connections. Some reprogramming because of power shutdown will be required if this is not done.

Step 2 — Verify that the 743 KSR terminal is connected to P2 with the TM990/503 cable. AC power is supplied to the terminal with a separate cord.

Step 3 — Special connections must now be made at the jumpers on the TM990/100M microcomputer. The jumper positions are shown in Chapter 3, *Figures 12* and *13.* Make sure of the following jumper connections.

JUMPERS	*INTERCONNECTION*	*COMMENT*
J15	Disconnected	Power for TM990/301
J14	Disconnected	Microterminal, not
J13	Disconnected	required for 743 KSR
J12	N.A.	For multiple boards
J11	Disconnected	For ASR 745
J10,9,8	N.A.	For multiple boards
J7	EIA position	
J6,5	N.A.	For multiple boards
J4,3,2	In 08, or 2708 Position	For 2708 EPROMS
J1	9902	This will likely need to be positioned

▶9

Step 4 — As mentioned previously, the RAM on the TM990/100M should be fully populated for this example. Make sure that 4-TMS 4042-2's have been inserted in U33, U35, U37 and U39 with the #1 pin towards the TMS 9900. The LBLA which is in two TMS 2708 EPROM's should have also been inserted in U43 and U45 with the #1 pin towards the TMS 9900. The higher order byte (bits 0-7) must be in U45. It is quite difficult to insert these packages in the sockets the first time so it must be done carefully. Rocking the packages will help.

Step 5 — Install the 5MT modules in the 5MT43 base as shown in *Figure 33.* Be sure modules are in the proper order. This arrangement will show dc input controlling dc output, dc input—ac output, ac input—dc output and ac input — ac output. Connect the wiring as shown. Be sure to use heavy gage (14 AWG) insulated wire for the ac connections. 18 AWG can be used for dc power connections. *NOTE THAT AC LINE IS CONNECTED TO DC COMMON.* Two screw connections on the base are available for each module as shown in *Figure 33.* All connections to the 5MT modules are to the right-hand leads when facing the terminals and P1 is on the left. Be sure to screw down the locking screw to ensure good connections.

Step 6 — Connect J1 of the cable of *Figure 4* to the 5MT43 base. Connect the +8V lead to the power supply and its ground to the common ground lead on J4 of the cable of *Figure 4. DO NOT CONNECT THIS GROUND TO THE DC COMMON OF THE INDUSTRIAL CONTROL LEVEL POWER SUPPLY OF FIGURE 33.*

Step 7 — Connect the +12Vdc industrial power supply. Don't plug in the 110Vac power cord.

Step 8 — Turn on the +8V and +12V supply and verify that the dc input and output 5MT modules are connected correctly. Use J4 for test voltages.

Step 9 — Plug-in the ac power cord for the 5MT modules and verify that the ac input and output modules are interconnected correctly. The LED's on the modules will be useful for this.

Step 10 — Unplug ac cord, turn off +12V and +8V supplies.

Step 11 — Connect J4 of the cable from the 5MT43 base to P4 on the TM990/100M module.

Step 12 — Turn on the power supplies for the microcomputer *in this order:* −12V, +12V, +5V.

A simulated industrial control application

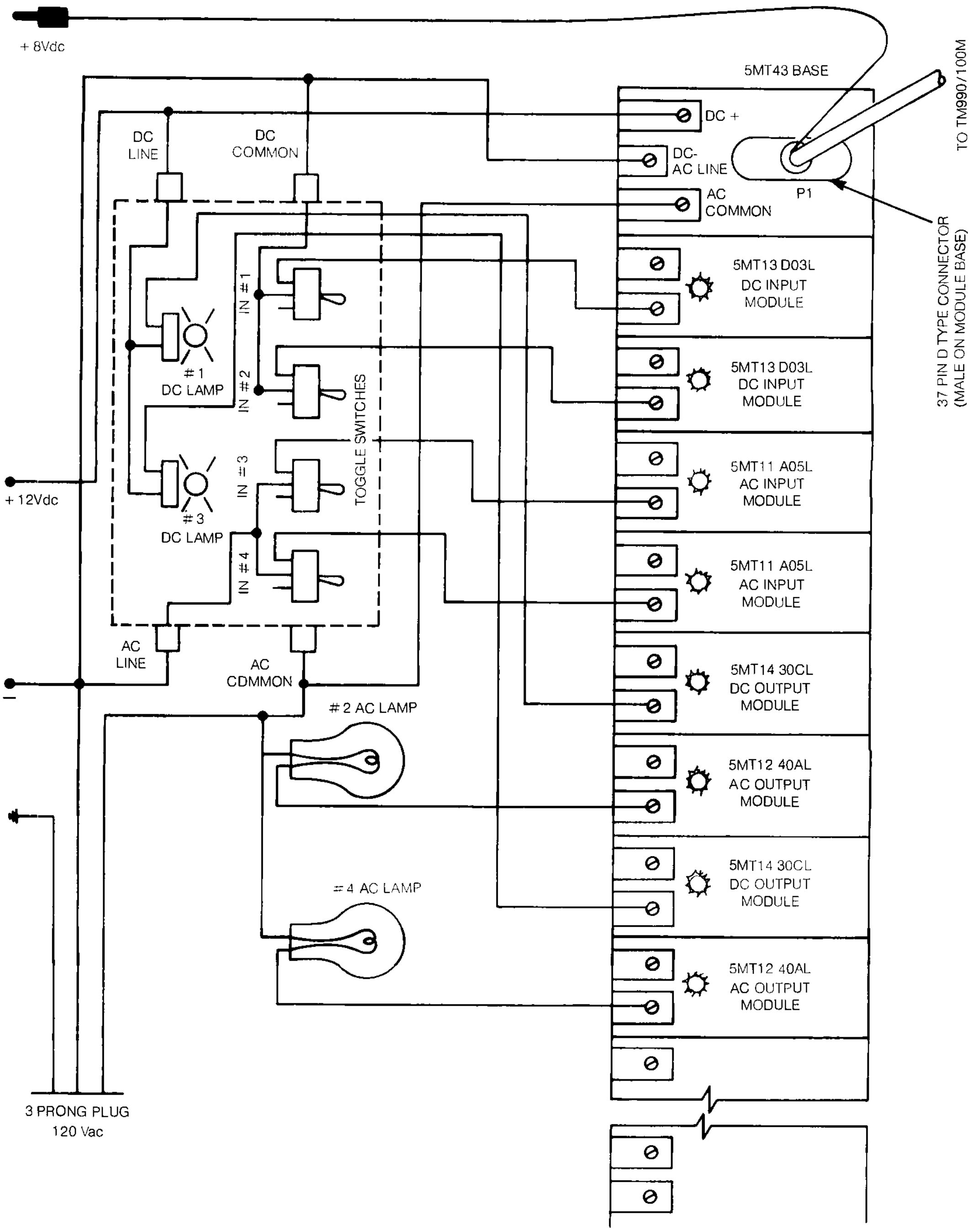

+ 8Vdc
5MT43 BASE
TO TM990/100M CABLE (TM990/507)
DC +
DC- AC LINE
AC COMMON
P1
37 PIN D TYPE CONNECTOR (MALE ON MODULE BASE)
DC LINE
DC COMMON
IN # 1
IN # 2
IN # 3
IN # 4
TOGGLE SWITCHES
# 1 DC LAMP
# 3 DC LAMP
5MT13 D03L DC INPUT MODULE
5MT13 D03L DC INPUT MODULE
5MT11 A05L AC INPUT MODULE
5MT11 A05L AC INPUT MODULE
5MT14 30CL DC OUTPUT MODULE
5MT12 40AL AC OUTPUT MODULE
5MT14 30CL DC OUTPUT MODULE
5MT12 40AL AC OUTPUT MODULE
+ 12Vdc
AC LINE
AC COMMON
# 2 AC LAMP
# 4 AC LAMP
3 PRONG PLUG 120 Vac

Figure 33. 5MT I/O Module Wiring

▸9

Step 13 – Turn on the terminal. Make sure it is "ON LINE."

Step 14 – Turn on the +8V supply for the 5MT modules; then the industrial level +12Vdc and then plug in the power cord for the 110Vac.

Step 15 – Press the RESET switch on the microcomputer. All the light bulbs will be lit since a RESET latches I/O pins on the microcomputer in the "1" state.

The microcomputer system is now ready to be programmed.

Loading the Program

The program as it was developed will now be loaded into RAM in the microcomputer. Instead of assembling the program by hand, the line-by-line assembler contained in EPROM will be used. It works with the EIA terminal and the TIBUG monitor.

The LBLA is a stand alone program that assembles into object code the 69 instructions used by the TM 990/100M microcomputer. To initialize the LBLA, the TIBUG monitor must first be brought up. This is done by switching the reset switch on the TM 990/100M module and pressing the carriage return (CR) on the terminal. The terminal will respond with:

```
TIBUG REV. A.
?
```

The question mark is the TIBUG prompt.

Now an R is typed to inspect/change the WP, PC and ST registers. The LBLA program begins at location $09E6_{16}$* so this is the value that is to be loaded into the PC. After typing an R the terminal prints out the value of the WP. This can be changed by typing the new value and a space or it can be left alone by typing just a space. The terminal will then print the value of the PC. The same procedure as for the WP applies except that ST is printed if a space is typed. A CR after the WP or PC value will cause the TIBUG prompt to be printed, or a space or CR after the ST is printed will do the same.

Loading $09E6_{16}$ into the PC looks like this:

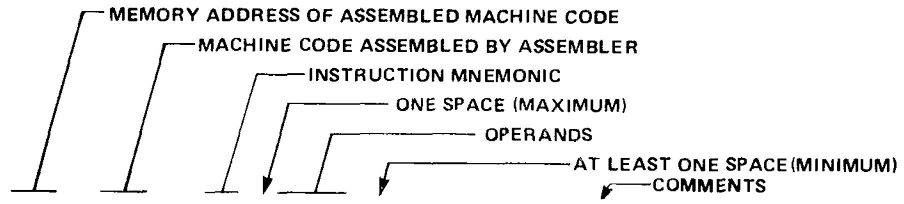
```
?R                     (CR)
W=FFC6                 (SP)
P=01A6      09E6       (CR)
?
```

Once the PC has been loaded, executing the program will initialize the LBLA. Pressing the E key accomplishes this. The LBLA responds with an address. That address can be changed to the starting address of the program by typing a slash (/) and the new address and a CR.

```
?E
FE00                 /FC00              (CR)
FC00
```

*This value may change depending on the version of LBLA. Early versions had $09E8_{16}$ as entry point.

The program can then be entered using the machine instructions. The LBLA accepts assembly language inputs from a terminal. As each instruction is input, the assembler interprets it, places the resulting machine code in an absolute address, and prints the machine code (in hexadecimal) next to its absolute address as shown in Figure 34.


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

```
FE00   02E0   LWPI >FE80             ; SET UP WORKSPACE ADDRESS
FE02   FE80
FE04   0200   LI R0, 10              ; SET UP COUNTER VALUE
FE06   000A
FE08   0201   LI R1,>FEA0            ; ADDRESS OF VALUES IN R1
FE0A   FEA0
FE0C   0202   LI R2,>FE80            ; ADDRESS OF STORAGE AREA IN R2
FE0E   FE80
FE10   CC81   MOV *R1+,*R2+          ; MOVE VALUES TO STORAGE AREA
FE12   0600   DEC R0                 ; DECREMENT COUNTER
FE14   1301   JEQ >FE18              ; EXIT IF COUNTER=ZERO
FE16   10FC   JMP >FE10              ; LOOP BACK UNTIL 10 VALUES MOVED
FE18
```

Figure 34. LBLA Format

Only one space is used between the mnemonic and the operand. If comments are used, use at least one space between the operand and the start of the comment. If no comment is used complete the instruction *with a space and a carriage return.* If a comment is used, only a carriage return is required.

Note that to load a hex value directly into a memory location a (+) is used. (see Start of Program, *Table 4.*) Also a string of characters is preceded by a dollar sign ($) and *terminated with two carriage returns*—CR (Example shown under—Message Routines). To change the address location being loaded, type a slash (/) and the address desired. To exit from the LBLA and return to the TIBUG monitor, press the ESC key on the terminal. The terminal will then give the TIBUG prompt—a question mark.

Labels cannot be used with the LBLA. However, in the program of *Table 4,* the left side is the assembled program with LBLA and the right side is for a comparison to the labels and the comments that were previously used on each of the pieces of the program as it was developed on the preceding pages.

Remember to press the ESC when the last program address location is reached. This returns control to the TIBUG monitor.

Table 4. Final Program

```
              LBLA                            Labels          Comments
?R
W=FFB0
P=0168  09E6
?E
FD00            /FC00
FC00    0929    +>929                         COUNT        ;  SET UP 9901 CLOCK
FC02    0100    +>100                         BASE 1       ;  SET UP 9901 CRU BASE
FC04    0120    +>120                         BASE 2       ;  SET UP 9901 I/O BASE
FC06    02ED    LWPI  >FF20                   START        ;  SET WP AT FF20
FC08    FF20
FC0A    0201    LI  R1,>1E00                               ;  SBZ OP CODE TO R1
FC0C    1E00
FC0E    0202    LI  R2,>1D00                               ;  SBO OP CODE TO R2
FC10    1D00
FC12    0203    LI  R3,>1F00                               ;  TB OP CODE TO R3
FC14    1F00
FC16    2FA0    XOP  @>FCF2,14                             ;  PRINT HEADER @MSG1
FC18    FCF2
FC1A    2FA0    XOP  @>FE00,14                COMODE       ;  ASK FOR MODE WITH MSG2
FC1C    FE00
FC1E    2EC7    XOP  R7,11                                 ;  READ CHAR FROM TER TO R7
FC20    0287    CI  R7,>3100                               ;  IS CHAR A 1?
FC22    3100
FC24    1308    JEQ >FC36                                  ;  IF YES GO TO MODE 1
FC26    0287    CI R7,>3200                                ;  IS CHAR A 2?
FC28    3200
FC2A    1325    JEQ  >FC76                                 ;  IF YES GO TO MODE 2
FC2C    0287    CI  R7,>5100                               ;  IS CHAR A Q?
FC2E    5100
FC30    16F4    JNE  >FC1A                                 ;  IF NO KEEP LOOPING
FC32    0460    B  @>0080                                  ;  IF YES GO TO TIBUG
FC34    0080
FC36    0360    RSET                          INPUT1       ;  PUT 9901 INTO INPUT MODE
FC38    0300    LIMI  4                                    ;  ENABLE 9900 INT1-INT4
FC3A    0004
FC3C    020C    LI  R12,>0080                              ;  LOAD R12 W/9902 BASE ADDR
FC3E    0080
FC40    3407    STCR  R7,0                                 ;  CLEAR 9902 RCV BUFFER
FC42    1D12    SBO  18                                    ;  ENABLE 9902 RCV INT
FC44    C320    MOV  @>FC02,R12                            ;  SET 9901 BASE ADDR TO >100
FC46    FC02
FC48    1D04    SBO  4                                     ;  ENABLE 9902 INT AT 9901
FC4A    C320    MOV  @>FC04,R12                            ;  SET 9901 BASE ADDR TO >120
FC4C    FC04
FC4E    04C4    CLR  R4                       INIT1        ;  R4 CONTAINS CRU BIT TO BE TESTED
FC50    C144    MOV  R4,R5                    INDEX1       ;  MOVE CRU BIT TO R5
FC52    E103    SOC  R3,R4                                 ;  R4 CONTAINS TB INST [R3]
FC54    04B4    X  R4                                      ;  EXECUTE TB SPECIFIED BY R4
FC56    130A    JEQ  >FC6C                                 ;  IF CRU BIT=1 GO TO HIGH
FC58    C105    MOV  R5,R4                    LOW          ;  RELOAD CRU BIT INTO R4
FC5A    0225    AI  R5,>4                                  ;  SHIFT CRU BIT OVER BY 4
FC5C    0004
FC5E    E141    SOC  R1,R5                                 ;  R5 CONTAINS SBZ OP CODE [R1]
FC60    0485    X  R5                         XECUTE       ;  EXECUTE OP CODE SPECIFIED BY R5
FC62    05B4    INC  R4                                    ;  INCREMENT TO NEXT CRU BIT
FC64    02B4    CI  R4,>3                                  ;  IS CRU BIT >3?
FC66    0003
```

```
FC68   15F2   JGT   >FC4E                       ; IF YES REINITIALIZE
FC6A   10F2   JMP   >FC50                       ; START TESTING NEXT CRU BIT
FC6C   C105   MOV   R5,R4             HIGH      ; RELOAD CRU BIT INTO R4
FC6E   0225   AI   R5,>4                        ; SHIFT CRU BIT OVER 4
FC70   0004
FC72   E142   SOC   R2,R5                       ; R5 CONTAINS SBO OP CODE [R2]
FC74   10F5   JMP   >FC60                       ; GO EXECUTE SBO INST
FC76   0360   RSET                    BLINKR    ; SET 9901 TO INPUT MODE
FC78   0300   LIMI  4                           ; ENABLE 9900 INT1-INT4
FC7A   0004
FC7C   020C   LI   R12,>B0                      ; SET UP 9902 BASE ADDR
FC7E   0080
FC80   3407   STCR   R7,0                       ; CLEAR 9902 RCV BUFFER
FC82   1D12   SBO   18                          ; ENABLE 9902 RCV INT
FC84   C320   MOV   @>FC04,R12                  ; SET 9901 BASE ADDR=>120
FC86   FC04
FC88   0204   LI   R4,>5              INT2      ; R4 CONTAINS CRU BIT POS 5
FC8A   0005
FC8C   C144   MOV   R4,R5             LOOP1     ; MOV POS 5 TO R5
FC8E   E141   SOC   R1,R5                       ; R5 CONTAINS SBZ OP CODE [R1]
FC90   0485   X   R5                            ; EXECUTE SBZ SPECIFIED BY [R5]
FC92   0584   INC   R4                          ; R4=R4+1
FC94   0284   CI   R4,>B                        ; HAS CRU BIT 7 BEEN SET=0?
FC96   000B
FC98   16F9   JNE   >FC8C                       ; IF NO GO TO LOOP1
FC9A   0204   LI   R4,>4                        ; SET OUTPUT BASE BIT
FC9C   0004
FC9E   0206   LI   R6,>4              INDEX2    ; OVERALL LOOP COUNT=100MS
FCA0   0004
FCA2   C320   MOV   @>FC02,R12        TIMER     ; SET CRU BASE ADDR OF 9901=>100
FCA4   FC02
FCA6   04C0   CLR   R0                          ; INITIALIZE INT3 INDICATOR
FCA8   33E0   LDCR   @>FC00,15                  ; LOAD TIMER AND START COUNT
FCAA   FC00
FCAC   1E00   SBZ   0                           ; 9901 TO INTERRUPT MODE
FCAE   1D03   SBO   3                           ; ENABLE INT3 AT 9901
FCB0   1D04   SBO   4                           ; ENABLE 9902 INT AT 9901
FCB2   0280   CI   R0,>FFFF          LOOP2     ; HAS INT3 OCCURRED?
FCB4   FFFF
FCB6   16FD   JNE   >FCB2                       ; IF NO GO TO LOOP2
FCB8   0606   DEC   R6                          ; R6=R6-1
FCBA   16F3   JNE   >FCA2                       ; IF R6=0 GO TO TIMER
FCBC   C320   MOV   @>FC04,R12                  ; SET 9901 BASE ADDR=>120
FCBE   FC04
FCC0   C144   MOV   R4,R5                       ; MOV CRU BIT TO R5
FCC2   E141   SOC   R1,R5                       ; [R5]=SBZ [R5]
FCC4   0485   X   R5                            ; EXECUTE SBZ SPECIFIED BY [R5]
FCC6   0584   INC   R4                          ; R4=R4+1
FCC8   0284   CI   R4,>9                        ; IS R4=9?
FCCA   0009
FCCC   13D4   JEQ   >FC76                       ; IF YES RESTART SEQUENCE
FCCE   C144   MOV   R4,R5                       ; R4=R5
FCD0   E142   SOC   R2,R5                       ; [R5]=SBO [R5]
FCD2   0485   X   R5                            ; EXECUTE SBO SPECIFIED BY [R5]
FCD4   10E4   JMP   >FC9E                       ; RESTART TIMING CYCLE AT INDEX2
FCD6   C320   MOV   @>FC02,R12        INTREC    ; SET 9901 BASE ADDR=>100
FCD8   FC02
FCDA   1E03   SBZ   3                           ; DISABLE INT3 AT 9901
FCDC   091C   SRL   R12,1                       ; SET BASE ADDR=>80 FOR 9902
FCDE   1E12   SBZ   18                          ; DISABLE 9902 INT
```

```
FCE0   3407   STCR  R7,0                      ; READ 9902 RCV BUFFER [CLEARS]
FCE2   020E   LI  R14,>FC1A                   ; LOAD ADDR OF COMODE INTO PC
FCE4   FC1A
FCE6   0380   RTWP                            ; RETURN TO 5MT ROUTINE
FCE8   020C   LI  R12,>100        CLKINT      ; SFT 9901 BASE ADDR
FCEA   0100
FCEC   1E03   SBZ  3                          ; DISABLE INT3 AT 9901
FCEE   071D   SETO  *R13                      ; SET PREVIOUS R0=>FFFF
FCF0   0380   RTWP                            ; RETURN TO INTERRUPTED ROUTINE
FCF2          /FF88
FF88   0460   B  @>FCE8                       ; GO TO INT3 SERVICE ROUTINE @CLKINT
FF8A   FCE8
FF8C          /FFAC
FFAC   0460   B  @>FCD6                       ; GO TO INT4 SERVICE ROUTINE @INTREC
FFAE   FCD6
FFB0           /FCF2
FCF2   354D    $5MT I/O DEMONSTRATION ROUTINE
FCF4   5420
FCF6   492F
FCF8   4F20
FCFA   4445
FCFC   4D4F
FCFE   4E53
FD00   5452
FD02   4154
FD04   494F
FD06   4E20
FD08   524F
FD0A   5554
FD0C   494E
FD0E   4520
FD10   0D0A    +>0D0A
FD12   4D4F    $MODE 1 — INPUTS 0-3 SWITCH OUTPUTS
FD14   4445
FD16   2031
FD18   202D
FD1A   2049
FD1C   4E50
FD1E   5554
FD20   5320
FD22   302D
FD24   3320
FD26   5357
FD28   4954
FD2A   4348
FD2C   204F
FD2E   5554
FD30   5055
FD32   5453
FD34   2020
FD36   342D    $4-7 RESPECTIVELY.
FD38   3720
FD3A   5245
FD3C   5350
FD3E   4543
FD40   5449
FD42   5645
FD44   4C59
FD46   2E20
FD48   0D0A    +>0D0A
```

```
FD4A  4D4F   $MODE 2 — OUTPUTS 4-7 ARE SWITCHED SEQUENTIALLY.
FD4C  4445
FD4E  2032
FD50  202D
FD52  204F
FD54  5554
FD56  5055
FD58  5453
FD5A  2034
FD5C  2D37
FD5E  2041
FD60  5245
FD62  2053
FD64  5749
FD66  5443
FD68  4845
FD6A  4420
FD6C  5345
FD6E  5155
FD70  454E
FD72  5449
FD74  414C
FD76  4C59
FD78  2E20
FD7A  0D0A   +>0D0A
FD7C  4120   $A Q RETURNS CONTROL TO THE TIBUG MONITOR
FD7E  5120
FD80  5245
FD82  5455
FD84  524E
FD86  5320
FD88  434F
FD8A  4E54
FD8C  524F
FD8E  4C20
FD90  544F
FD92  2054
FD94  4845
FD96  2054
FD98  4942
FD9A  5547
FD9C  204D
FD9E  4F4E
FDA0  4954
FDA2  4F52
FDA4  0D0A   +>0D0A
FDA6  4120   $A CARRIAGE RETURN DURING MODE 1 OR 2 OPERATION
FDA8  4341
FDAA  5252
FDAC  4941
FDAE  4745
FDB0  2052
FDB2  4554
FDB4  5552
FDB6  4E20
FDB8  4455
FDBA  5249
FDBC  4E47
FDBE  204D
```

9

```
FDC0  4F44
FDC2  4520
FDC4  3120
FDC6  4F52
FDC8  2032
FDCA  204F
FDCC  5045
FDCE  5241
FDD0  5449
FDD2  4F4E
FDD4  5245  $RETURNS THE USER TO THE
FDD6  5455
FDD8  524E
FDDA  5320
FDDC  5448
FDDE  4520
FDE0  5553
FDE2  4552
FDE4  2054
FDE6  4F20
FDE8  5448
FDEA  4520
FDEC  0D0A  +>0D0A
FDEE  434F  $CONTROL MODE.
FDF0  4E54
FDF2  524F
FDF4  4C20
FDF6  4D4F
FDF8  4445
FDFA  2E20
FDFC  0D0A  +>0D0A,
FDFE  0000  +>0000
FED0  0D0A  +>0D0A
FE02  5345  $SELECT MODE 1, 2 OR Q
FED4  4C45
FE06  4354
FE08  204D
FEDA  4F44
FEDC  4520
FE0E  312C
FE10  2032
FE12  204F
FE14  5220
FE16  5120
FE18  0D0A  +>0D0A
FE1A  0000  +>0000
FE1C
```

9◂

RUNNING THE PROGRAM

To execute the program, the PC needs to be set to the starting address. This is done by typing an R to enter the inspect/change mode of TIBUG. The WP will be printed. A space will give the PC and here the new PC should be entered. A CR will return to TIBUG and the prompt will be given. Typing an E will cause the program to begin executing. The following is an example of this:

```
?R
W=FFFE  (SP)
P=006C  FC00  (CR)
?E
```

The program will begin by requesting a mode of operation from the user. Typing a "1" will get mode 1 and the state of outputs can be changed by changing the input toggle switches. Pressing a key will cause a return to the command mode. Pressing a 2, switches to mode 2 and the light sequence. Pressing a key returns to the command mode. Pressing a Q on the terminal returns the system to the TIBUG and specific address locations could be inspected for contents, etc.

Debugging

Because of the hard copy given by the terminal, looking for mistakes is made easier. If the program is stuck in a loop, the reset switch on the TM990/100M board can be switched. When in the LBLA use a slash (/) and a new address to change the address. When in TIBUG use the memory inspect/change (M) command to change the address. The TM990/100M user's guide gives the TIBUG commands and the TM990/402 LBLA user's guide gives the LBLA commands. These are also given in Chapter 7 and on the reference cards in the appendix.

I/O EXPANSION WITH THE TM990/310

What remains now is to show the I/O expansion through the use of the TM990/310 module. As shown in *Figure 35,* there are three additional 9901's on the /310 module. The 9901's signals are connected to edge connections P2, P3, and P4, respectively, and are shown in *Table 5.*

All of the pins on the connector to P1 on the 900/100M-1 microcomputer module must now be connected to P1 on the TM990/310 module (if not made previously). These are shown in *Table 6.* Such a power down requires the program to be re-entered.

Table 5. *9901 Pin-Outs on TM990/310.*

P2, P3, P4 Pin Number	Signature
20	P0
22	P1
14	P2
16	P3
18	P4
10	P5
12	P6
24	INT15/P7
26	INT14/P8
28	INT13/P9
30	INT12/P10
32	INT11/P11
34	INT10/P12
36	INT9/P13
38	INT8/P14
40	INT7/P15
6	Neg. Edge Triggered INT5
8	Pos. Edge Triggered INT6
1	+12V
2	−12V
3	+5V
4	Spare
All remaining pins	Ground

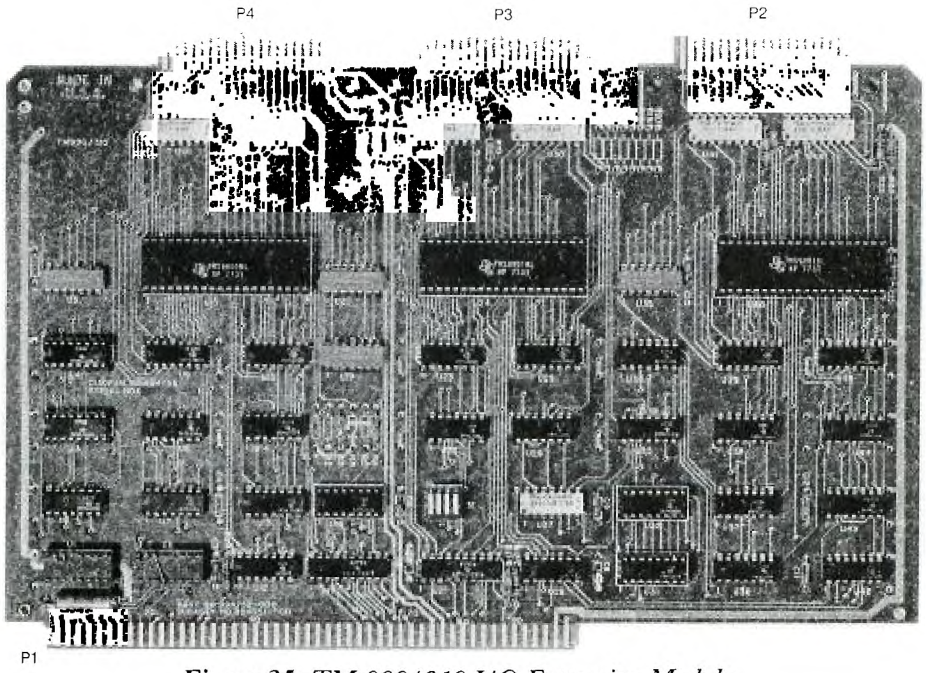


Figure 35. *TM 990/310 I/O Expansion Module*

9◂

Table 6. P1 Connections

P1 PIN	SIGNAL	P1 PIN	SIGNAL	P1 PIN	SIGNAL
33	D0	71	A14	12	$\overline{\text{INT13}}$
34	D1	72	A15	11	$\overline{\text{INT14}}$
35	D2	22	$\overline{\phi 1}$	14	$\overline{\text{INT15}}$
36	D3	24	$\overline{\phi 3}$	28	EXTCLK
37	D4	92	$\overline{\text{HOLD}}$	3	+5V
38	D5	86	HOLDA	4	+5V
39	D6	82	DBIN	97	+5V
40	D7	26	$\overline{\text{CLK}}$	98	+5V
41	D8	80	$\overline{\text{MEMEN}}$	75	+12V
42	D9	84	$\overline{\text{MEMCYC}}$	76	+12V
43	D10	78	$\overline{\text{WE}}$	73	−12V
44	D11	90	READY	74	−12V
45	D12	87	$\overline{\text{CRUCLK}}$	1	GND
46	D13	30	CRUOUT	2	GND
47	D14	29	CRUIN	21	GND
48	D15	19	IAQ	23	GND
57	A0	94	$\overline{\text{PRES}}$	25	GND
58	A1	88	$\overline{\text{IORST}}$	27	GND
59	A2	16	$\overline{\text{INT1}}$	31	GND
60	A3	13	$\overline{\text{INT2}}$	77	GND
61	A4	15	$\overline{\text{INT3}}$	79	GND
62	A5	18	$\overline{\text{INT4}}$	81	GND
63	A6	17	$\overline{\text{INT5}}$	83	GNG
64	A7	20	$\overline{\text{INT6}}$	85	GND
65	A8	6	$\overline{\text{INT7}}$	89	GND
66	A9	5	$\overline{\text{INT8}}$	91	GND
67	A10	8	$\overline{\text{INT9}}$	99	GND
68	A11	7	$\overline{\text{INT10}}$	100	GND
69	A12	10	$\overline{\text{INT11}}$	93	$\overline{\text{RESTART}}$
70	A13	9	$\overline{\text{INT12}}$		

9

Using the TM990/310 Board

The TMS 9901s on the TM990/310 board are accessed in the same manner as the TMS9901 on the TM990/100M board except the CRU base addresses differ. These hardware base addresses are user selectable by a DIP switch that is on the TM990/310 board. The position of the switch and the corresponding addresses are given in *Figure 36.* The first column of addresses are the actual CRU hardware addresses and the second column is the software address that is to be loaded into workspace register 12 to access the appropriate TMS 9901. The addresses shown correspond to the first TMS 9901 on the TM990/310 board and the positions on the DIP switch. The addresses to be loaded into workspace register 12 for the second TMS9901 are obtained by adding 80_{16} to the addresses of the first TMS 9901. The addresses for the third TMS 9901 are obtained by adding 80_{16} to the addresses of the second TMS 9901 (or 100_{16} to the addresses of the first TMS 9901). For example, if S1 was set to binary 4, workspace register 12 would be loaded with: 0800_{16} to access the first TMS 9901, 0880_{16} to access the second TMS 9901, or 0900_{16} to access the third TMS 9901. The first TMS 9901 corresponds to the P2 pins, the second to the P3 pins, and the third to the P4 pins.

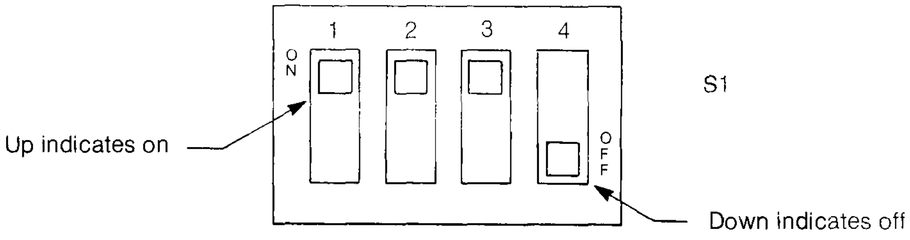
Switch all S1 positions on so the hardware base address 0100 is used for the /310 to correspond to the example in *Figure 11.* The third 9901 will be used so the software base address to be loaded in the program will be 0300_{16} and the I/O software base address will be 0320_{16}. The connection to the 5MT I/O modules will be thru P4 on the TM990/310 as shown in *Figures 27* and *35.* This connection should be made at this time.

Changing the Program

To change the program, the software address at the labels BASE 1 and BASE 2 needs to be changed. In the assembled program, these are at FC02 and FC04. The TIBUG monitor mode is obtained. A memory inspect/change (M) command to address FC02 will allow a change of the contents at that address to 0300_{16}. A space obtains address FC04 and its contents can be changed to 0320_{16}. However, when this change is made, the 9901 in the TM990/100M module no longer is enabled to receive the keyboard interrupt from the 9902 and, thus, the mode operation cannot be interrupted. Additional program changes must be made at $FC44_{16}$, $FCA2_{16}$, and $FCD6_{16}$ to continue to enable the 9901 INT4 in the module.

More sophisticated program changes could be made but one pattern that can be used for such changes is as follows:

```
1. (CR)                        ; CARRIAGE RETURN TO MONITOR
2. R                           ; OBTAIN WORKSPACE POINTER
3. (SP)                        ; OBTAIN PROGRAM COUNTER
4. 09E6 (CR)                   ; SET PC FOR LBLA
5. E                           ; EXECUTE LBLA
6. /FC44 (SP) (CR)             ; GO TO FC44
7. LI R12,>0100  (SP) (CR)     ; LOAD SOFTWARE BASE ADDRESS FOR
                                 9901 ON MODULE
```



S1 Switch Settings 1	2	3	4	Binary Equal	TM990/310 Module CRU Base Address (Hex)	Register 12 Contents (Hex)
ON	ON	ON	ON	0	0100	0200
ON	ON	ON	OFF	1	01C0	0380
ON	ON	OFF	ON	2	0280	0500
ON	ON	OFF	OFF	3	0340	0680
ON	OFF	ON	ON	4	0400	0800
ON	OFF	ON	OFF	5	04C0	0980
ON	OFF	OFF	ON	6	0580	0B00
ON	OFF	OFF	OFF	7	0640	0C80
OFF	ON	ON	ON	8	0700	0E00
OFF	ON	ON	OFF	9	07C0	0F80
OFF	ON	OFF	ON	A	0880	1100
OFF	ON	OFF	OFF	B	0940	1280
OFF	OFF	ON	ON	C	0A00	1400
OFF	OFF	ON	OFF	D	0AC0	1580
OFF	OFF	OFF	ON	E	NOT USED	NOT USED
OFF	OFF	OFF	OFF	F	NOT USED	NOT USED

Figure 36. Programming Base Address of TM 990/310 Module

On the terminal the routine looks like this:

```
Q
?R
W=FF20
P=FC1A 09E6
?E
FE00    /FC44
FC44    020C    LI  R12,>100
FC46    0100
FC48
```

The same program change must be made at FCA2 and FCD6. When these are made, return to TIBUG by pressing the ESC key. The memory location just changed can be checked with the M command and the memory location.

To run the program, press the R key (it gives the WP) then (SP) to get the PC. Change the PC to $FC00_{16}$ and execute the program by pressing (CR) and the E key.

Incidentally, after these program changes, the only thing that needs to be done to change the 5MT I/O to the microcomputer module connector P4 is to change the original software base addresses at FC02 = >0100 and FC04 = >0120. No other changes need be made.

FUTURE EXTENSIONS

Now that the system is available there are endless variations that can be accomplished. Here are some that come to mind immediately:

1. Change the time interval on Mode 2 by:
   a. Changing the value in R6
   b. Changing the value loaded into the clock register
2. Add more modules to the 5MT43 and program a different input-output relationship.
3. Reprogram so that the program itself shifts the 5MT I/O to the /310 module if a /310 is present. Otherwise, the interface would remain on P4 of the microcomputer module.
4. Expand to more modules thru the TM990/310 modules.
5. Investigate how interrupts come through the TM990/310 module to the processor. There are some special linkages that must be connected on the /310 module to choose the interrupts that will come through the /310 to the processor.

CONCLUSION

It has been quite an experience starting at the first encounter and proceeding to the point where a microcomputer system is up and running and capable of being programmed to sense and control real-world industrial level energy. Components are available to easily apply the systems to many varieties of problem solutions.

Continue the learning process by finding real things to do with the system. Build on it to use it to its full capability and then add to it or replace it with a larger system to expand the applications. And remember, all the software that has been learned will be applicable to the new system applications, to different 9900 family members, and to new family members to be added in the future. Common compatible software is a real advantage. It's built into the 9900 family, so build on it. Good Luck.

A Low Cost Data Terminal

ABSTRACT

The architecture of the TMS 9940 Microcomputer is briefly reviewed. The microcomputer portion of a data terminal which currently employs the TMS 8080A Microprocessor is described. An equivalent design, which significantly reduces the chip count by using the TMS 9940 Microcomputer, is discussed in detail. Software comparisons between the two systems are made. A cost analysis of the two designs is discussed.

INTRODUCTION

As the complexity of LSI (large scale integration) electronics continues to increase, the system designer gains more and more freedom in designing low cost systems. One example of this capability is the Texas Instruments (TI) Model 745 Electronic Data Terminal, first introduced by TI in 1975. The Model 745 is a self-contained compact, telecommunications terminal which uses the thermal printing technique to achieve silent operation. The Model 745 features a 58 key, TTY33-compatible modular keyboard with integral numeric keypad, carrier detect indicator, two-key rollover, and key debounce circuitry. The Model 745 is capable of operating in full or half duplex modes at 10 or 30 characters per second, using a character set and code compatible with the American Standard Code for Information Interchange (ASCII).

The particular design of the Model 745 Data Terminal was made possible by the use of a microcomputer system as its controller. The Model 745 incorporates a TMS 8080A Microprocessor as the CPU of the Microcomputer. The purpose of this paper is to show how the Model 745 Terminal could be simplified even further by utilizing the newest addition to the 990/9900 Computer family: the TMS 9940 Microcomputer.

MICROCOMPUTER ARCHITECTURES

TMS 8080A MICROPROCESSOR

The TMS 8080A is an eight-bit general purpose Microprocessor *(Figure 1)*. The TMS 8080A chip contains seven registers and has a 78-instruction repertoire. The chip requires three power supplies (+12, ±5Vdc) and accepts a two-phase high-level clock input. The TMS 8080A features 64K byte addressing of off-chip memory, and is packaged in a 40-pin package.

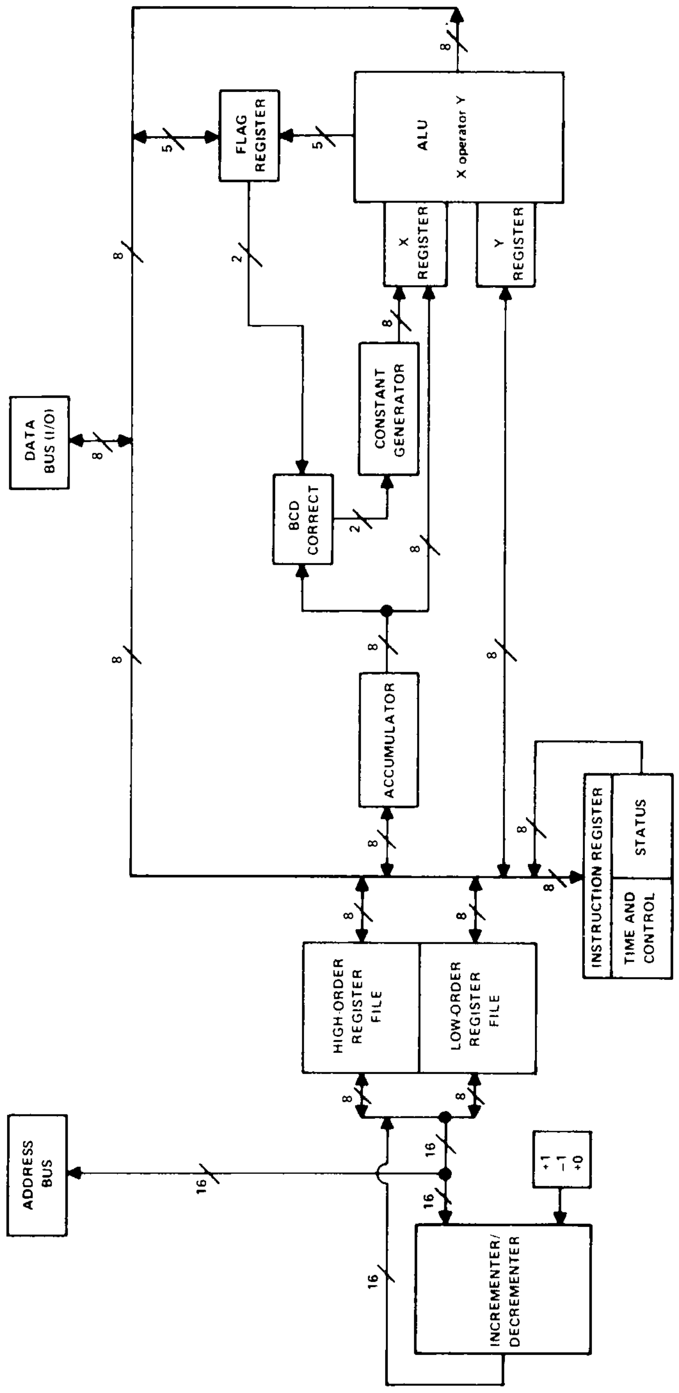


Figure 1. TMS 8080A Functional Block Diagram

TMS 9940 Microcomputer

The TMS 9940 is a 16-bit general purpose, single-chip microcomputer *(Figure 2)*. The TMS 9940 contains 2K bytes of ROM (or EPROM) and 128 bytes of RAM, along with a programmable timer/event counter. The 9940 is software-compatible with the 990/9900 family of microprocessors/minicomputers, and executes 68 instructions. The TMS 9940 requires a single 5-volt power supply and incorporates an (external) crystal-controlled oscillator on the chip. The circuit has 32 bits of general purpose I/O (expandable to 256 bits), and is housed in a 40-pin package.

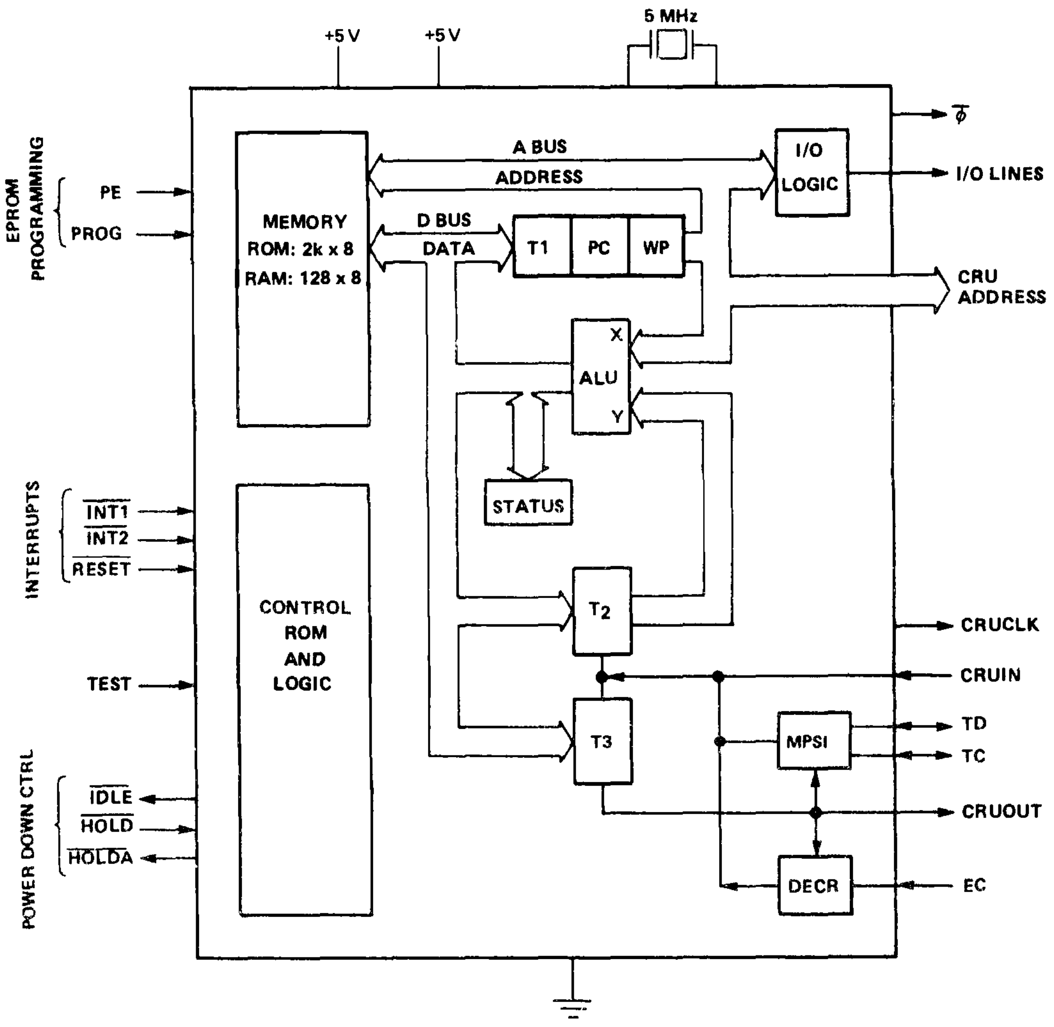


Figure 2. TMS 9940 Functional Block Diagram

HARDWARE DESIGN

A functional block diagram of the Model 745 Data Terminal is shown in *Figure 3.* The control electronics monitor all terminal inputs and generate all necessary timing and control signals to effect data transfers, cause printhead and paper motion, and create printable characters through the thermal printhead. Each block of the diagram is discussed separately below.

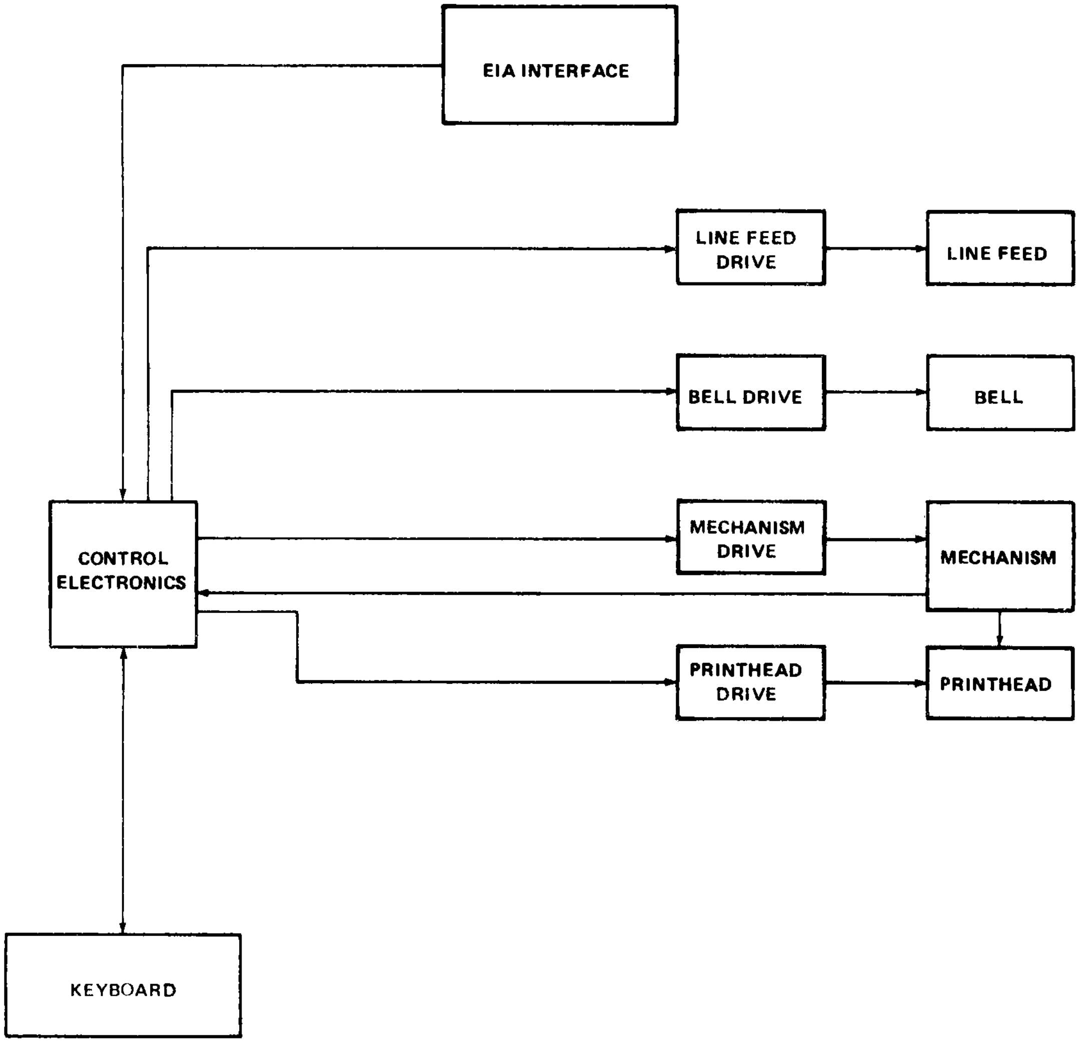


Figure 3. Model 745 Data Terminal Functional Block Diagram

Keyboard

The Model 745 keyboard is a TTY33-compatible, alphanumeric keyboard with an integral numeric keypad. The keyboard is equipped with 54 single-action keys, four alternate action switches, and an indicator lamp which signals that the data carrier signal is being received by the terminal. The control electronics must generate control signals to scan the keyboard and debounce key switch depressions. When a key depression is detected during a scan, the character is encoded and the appropriate action is taken by the terminal. Each scan is total so as to detect possible multiple key depressions. When simultaneous depressions are detected during a scan, neither key is acted upon. This scanning/debounce technique effects a two-key rollover with lockout.

Printhead

The printhead consists of a five by seven dot matrix of 35 heating elements *(Figure 4)* mounted on a monolithic chip. The chip is mounted on a heatsink, and is connected to the printhead drive electronics through a flexible ribbon cable. Upon receipt of a character from the keyboard or the communications line, the control electronics must generate the appropriate control signals to form the selected character utilizing the five by seven dot matrix format. The PRINT signal is switched on; then the matrix data is transferred to the printhead one column at a time. Each of the 35 heating elements on the printhead contains an SCR which controls the heating current. When both X and Y inputs are positive to a given element, the SCR energizes and reamins on (approximately 10 msec) until PRINT is switched off.

The X and Y address drivers are implemented on two SN98614 linear integrated circuits, each of which consists of six driver circuits. Each driver circuit has a low power TTL-AND input stage and a totem-pole, power transistor output stage. The drivers are enabled by the signal LDPRHD.

Printhead Lift

The printhead is lifted to relieve pressure upon the paper during line feed and carriage return operations. The control electronics must generate a signal (LFTHD) to control the solenoid which lifts the printhead.

Mechanism

Horizontal movement of the printhead is controlled by a three-phase 15-degree stepping motor. An optical sensor is mounted on the motor shaft to provide feedback for the control of stepping motion during printing and slew motion during carriage return. The print/step cycle operates synchronously up to 35 characters per second. The control electronics must output five signals to control the motor. The STEP and FAST signals are used to control the current in the motor windings; and PHA, PHB, and PHC are drive signals for the three motor phases. The mechanism drive electronics converts these TTL logic level signals into the closed loop controller dc current required by the motor.

▶9

The optical sensor provides data on motor position so that the control electronics "know" when to apply braking to change phases, or to make other decisions concerning motion of the printhead carriage. The sensor consists of a 24-position slotted,wheel which interrupts a light path between an IR emitting diode and a photosensitive transistor. The sensor issues pulses to the control electronics as the slots interrupt the light path.

Bell

A buzzer (a piezoelectric disc) produces an audible signal at a nominal frequency of 3.2 kHz. Upon receipt of the BEL character from the keyboard or communications line, the control electronics must generate a timed signal (250 ± 25 msec) to produce the sound.

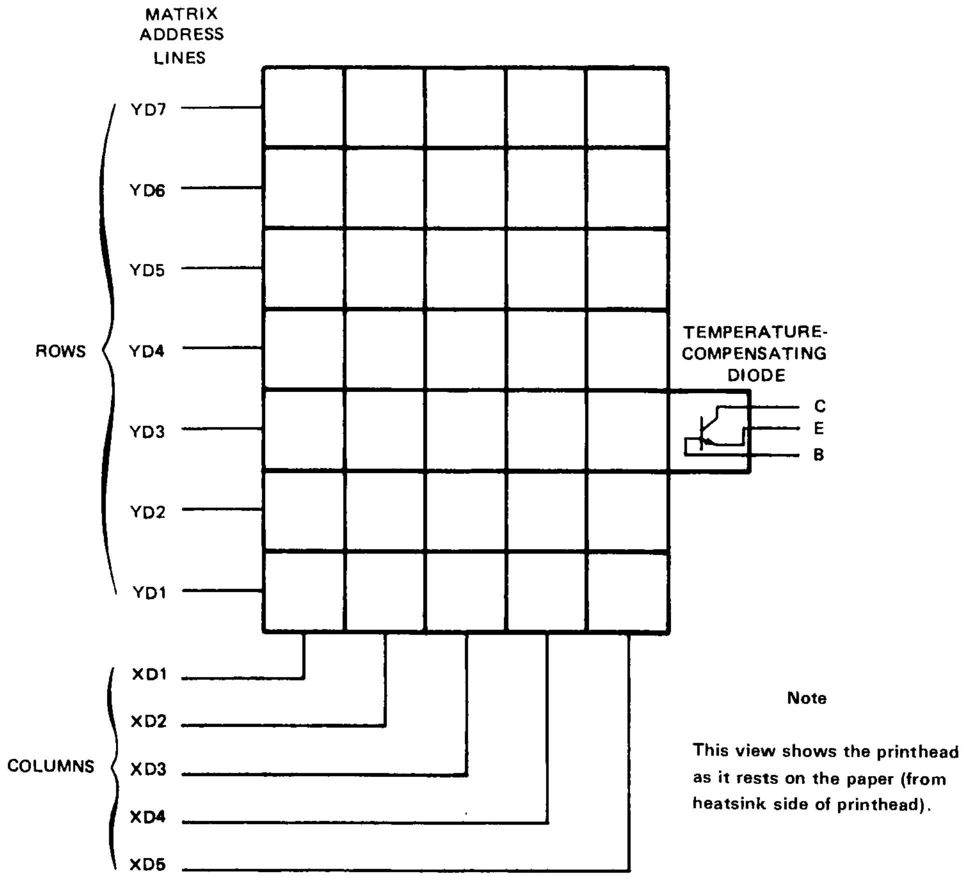

MATRIX ADDRESS LINES
ROWS
YD7
YD6
YD5
YD4
YD3
YD2
YD1
COLUMNS
XD1
XD2
XD3
XD4
XD5
TEMPERATURE-COMPENSATING DIODE
C
E
B
Note
This view shows the printhead as it rests on the paper (from heatsink side of printhead).

Figure 4. Printhead Matrix Address Lines

Line Feed

Vertical movement of the paper is controlled by the line feed solenoid which is mechanically coupled to a rachet mechanism. To advance the paper one line, the control electronics must lift the printhead and output a timed signal (15 msec) followed by an off period of 16.8 msec to the line feed solenoid.

EIA Interface

The control electronics must transmit and receive asynchronous serial data in accord with *ANSI Standard for Character Structure and Parity Sense,* X3.16-1966 and *ANSI Standard for Bit Sequence,* X3.15-1967. The TTL-level signals RCVD and XD are converted to standard EIA RS-232-C levels in the EIA interface.

Control Electronics

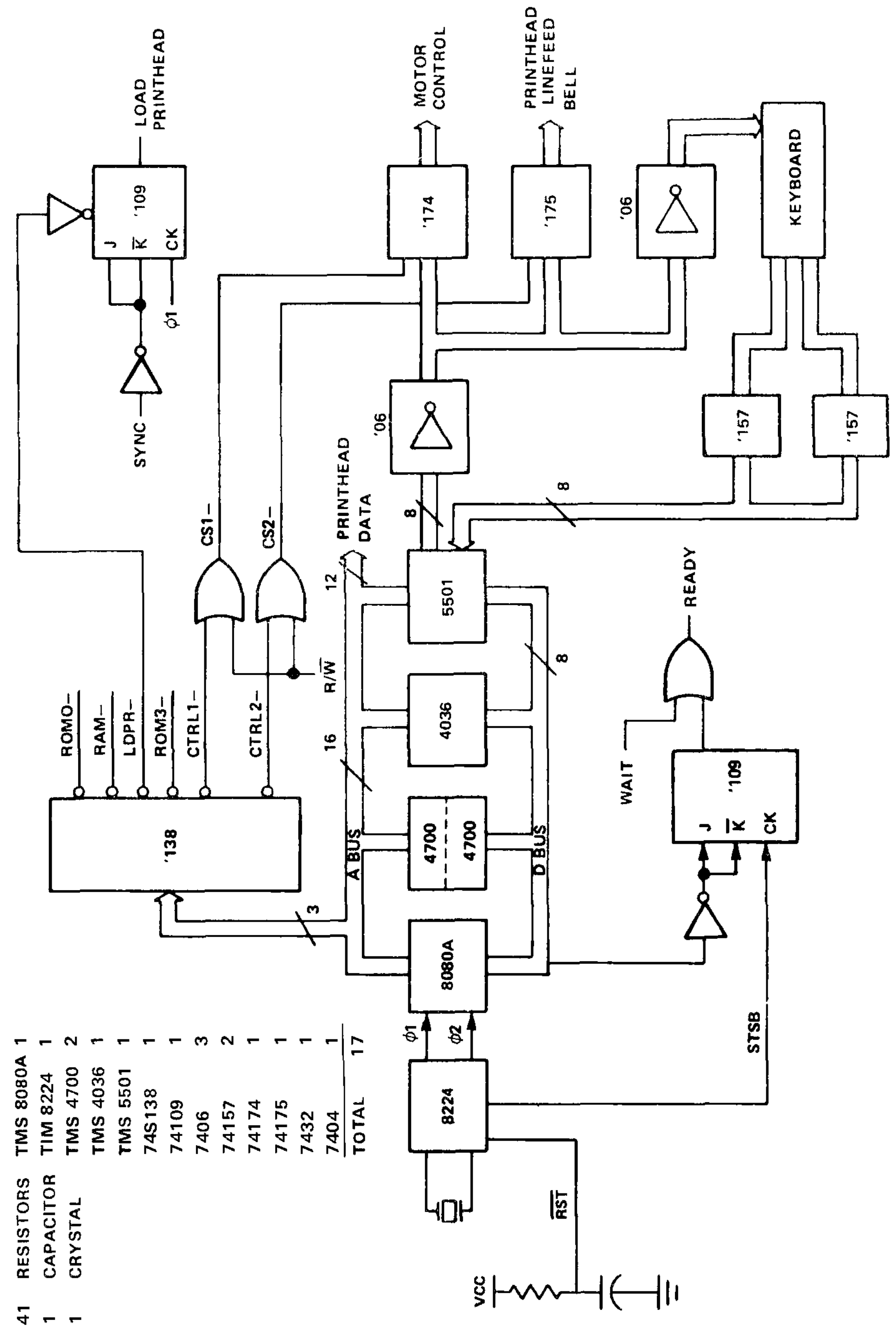
The control electronics function is performed by an interrupt driven, stored program microcomputer. As aforementioned the system requirements for the microcomputer I/O consist of:

Keyboard:	Matrix scan lines
Printhead:	Print data (12),LDPRHD,PRINT,LFTHD
Mechanism:	Step,FAST,PHA,PHB,PHC,SENSOR
Bell:	BELL
Linefeed:	LNFD
EIA Interface:	RCVD,XD

The microcomputer must generate these signals in the specified times and sequences to control the system.

TMS 8080A Microcomputer System

A schematic of the microcomputer design using the TMS 8080A Microprocessor is shown in *Figure 5.* The complete design requires 17 integrated circuits, 41 resistors, one crystal, and one capacitor. The memory consists of 2K bytes of ROM (two TMS 4700's) and 64 bytes of RAM (one TMS 4036). The TMS 5501 is an 8080A peripheral I/O controller which contains a universal asynchronous receiver/transmitter, programmable timers, interrupt prioritization and control, an eight-bit input port, and an eight-bit output port. The eight-bit output port is expanded by using TTL components (7406, 74174, 74175) to provide the necessary number of direct outputs for the keyboard and latched outputs for the static outputs. The input port is expanded using 2-to-1 multiplexers (74157) to permit elimination of diodes from the keyboard matrix. Data is sent to the printhead over 12 bits of the address bus by loading the data into the HL registers, and then executing a dummy MOVM instruction while the 74109 JK flip-flop outputs the LDPRHD strobe signal. The 74S138, 3-to-8 decoder generates the required chip selects for the various components. The SENSOR input feeds into the TMS 5501 interrupt logic to interface to the TMS 8080A.



41	RESISTORS
1	CAPACITOR
1	CRYSTAL

CHIP COUNT

TMS 8080A	1
TIM 8224	1
TMS 4700	2
TMS 4036	1
TMS 5501	1
74S138	1
74109	1
7406	3
74157	2
74174	1
74175	1
7432	1
7404	1
TOTAL	17

Figure 5. TMS 8080A Microcomputer System

TMS 9940 Microcomputer System

A schematic of the microcomputer design using the TMS 9940 Microcomputer is shown in *Figure 6.* The complete design requires two integrated circuits, 18 resistors, one crystal, one capacitor and 16 diodes. The internal memory of the TMS 9940 provides 2K bytes of ROM and 128 bytes of RAM. The TMS 9902 Asynchronous Communications Controller is a TMS 9900-family peripheral which contains a universal asynchronous receiver/transmitter and a programmable timer. The 32 I/O lines provided by the TMS 9940 interface to all the I/O functions with 10 lines software-multiplexed between the keyboard scan, TMS 9902 control, and printhead data. When P14 through P20 are in the input mode, the keyboard is scanned by sequentially raising P1 through P10 *high* (with the others being held *low*) while switching P14 through P20 to the output mode and outputting *high* signals, isolates P1 through P10 so that they can be used for other purposes. The LDPRHD signal is divided into two signals (LDPRHD1 and LDPRHD2) to obtain an output current sink needed for the SN98614's. The two interrupt inputs are used by the SENSOR input (highest priority) and the $\overline{\text{INT}}$ output from the TMS 9902.

FIRMWARE DESIGN

A block diagram of the Model 745 firmware, *Figure 7,* shows that the system firmware can be divided into three major sections: (1) keyboard scanning and encoding, (2) printhead control, and (3) internal data control. The keyboard and printhead routines represent the major portion of the system: the data control routine is used to direct character processing between the keyboard, the printhead, and the EIA interface.

Keyboard Routine

The keyboard is viewed by the control electronics as a matrix of key switches, with all keyboard scanning, debouncing, and encoding done by the microcomputer. The keyboard is scanned once each 4.3 msec. When a key depression is detected, the character is encoded by the addition of a constant number to the row/column number of the key to provide the ASCII code, and the appropriate action is taken by the terminal. (*Note:* In the numeric mode a look-up table is used to provide the ASCII code).

After a depression is detected, 12 msec are allowed for all contact-make bounce to settle out and then scanning resumes at 4.3-msec intervals. No other key depressions are processed by the terminal until the first depression is released. When this occurs, 12 msec are allowed for contact-break bounce, then the keyboard scan again resumes at 4.3-msec intervals. Each scan is a complete scan so that multiple key depressions may be detected. When simultaneous depressions are detected, neither key is acted upon, thus effecting a two-key-rollover-with-lockout operation.

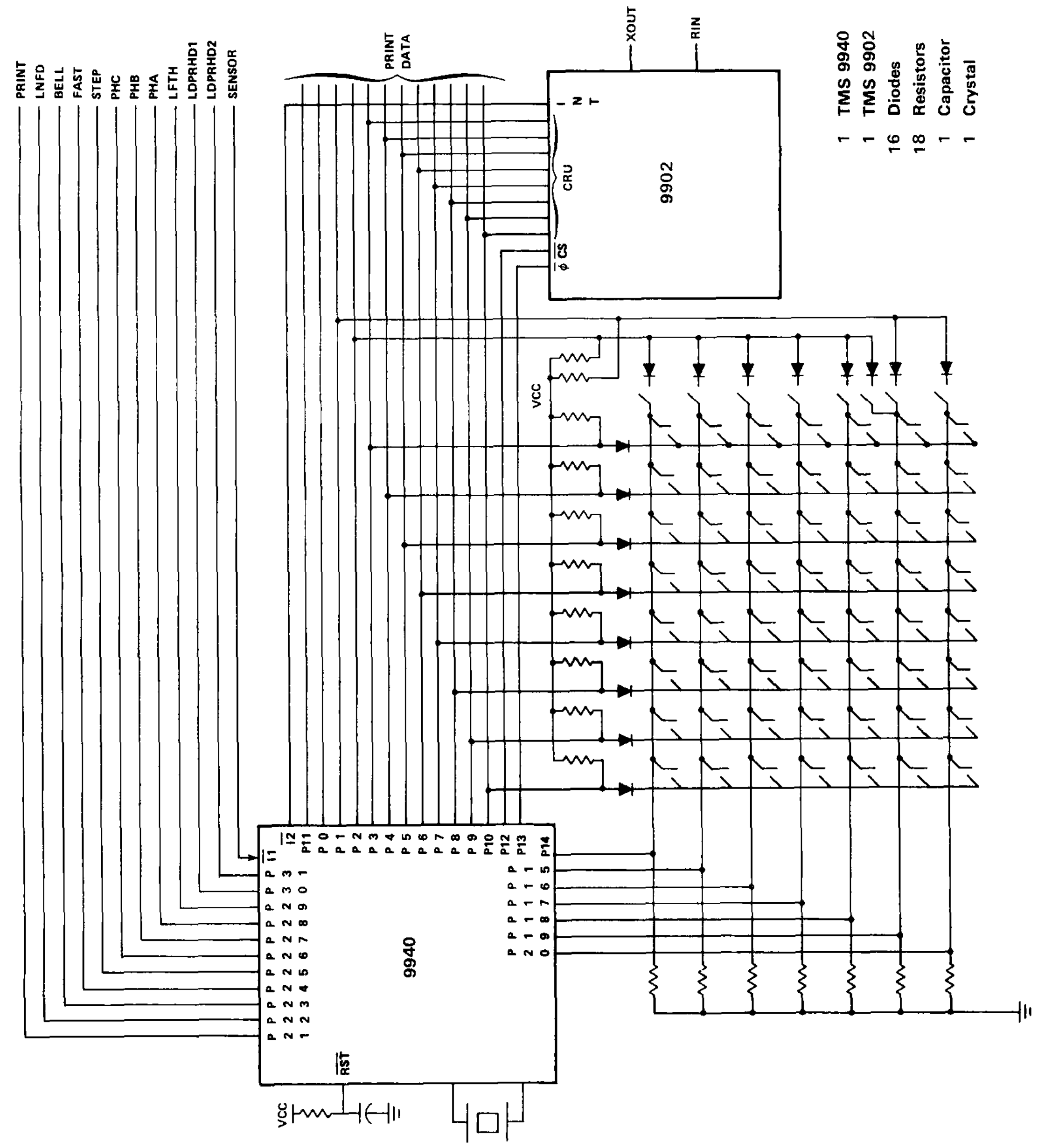


Figure 6. TMS 9940 Microcomputer System

Printhead Control

The microcomputer positions the printhead horizontally by timing different levels of current through the phase windings of the stepping motor. The print/step cycle operates asynchronously up to 35 CPS, with the cycle time divided into three basic segments: settle (11.3 msec), print (10 msec), and step (7.2 msec). Slew time for a full 80 columns is a maximum of 195 msec with backspace operations performed in one character-time. An automatic carriage return/line feed is executed upon receipt of the 81st character in a line. Upon applying power the printhead is backspaced to the left margin.

Fault detection methods are used by the microcomputer to prevent damage during power cycling conditions, obstruction of printhead motion, or loss of optical sensor signal. During the print segment, the microcomputer energizes the printhead voltage (PRINT), indexes into the dot matrix table (part of the 2K of ROM) by the ASCII character value, chooses the appropriate dot pattern, and loads the printhead one column at a time. The printhead is loaded during the first 200 μsec of PRINT; the PRINT signal remains on for 10 msec to allow the thermal sensitive paper to convert.

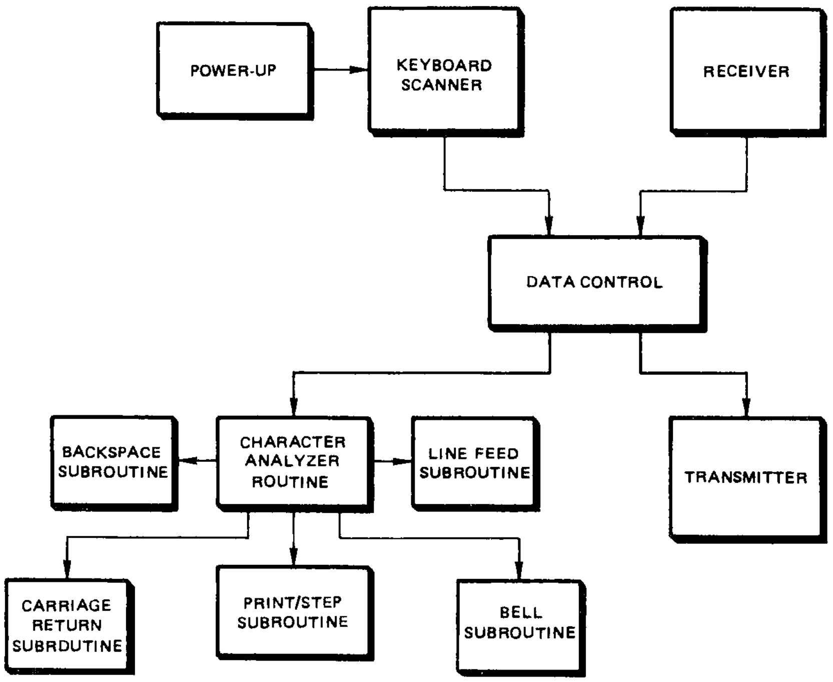


Figure 7. Model 745 Firmware Structure

The step segment steps the printhead one column by using two timers and the sensor. One timer is used to control pulse widths for the FAST and STEP pulses. These pulses control the amount of current in both the leading and lagging winding of the stepper motor, thus controlling the torque generated by the motor. The sensor signals the beginning of braking. The second timer is used to time the total step and is divided into two segments: The first verifies that the sensor occurred, and the second segment defines the end of the step. The use of the second timer makes the step time independent of when the sensor interrupt occurs so that the microcomputer can compensate for varying friction loads on the printhead.

The carriage return operation will slew the head to column one under control of the microcomputer using two timers and the sensor input. The step current remains on during the entire carriage return to develop high torques in the motor. One timer is used to control the fast pulse, thus controlling the current in the lagging phase of the stepper motor. The second timer is used as a reference to which to compare the sensor information, and this comparison results in the microcomputer accelerating or decelerating the motor to maintain control of printhead speed.

Firmware Implementation

Table 1 lists the number of instructions and memory bytes required to implement the system firmware for both the TMS 8080A and the TMS 9940. The three major sections [(1) keyboard routine, (2) printhead control, and (3) data control] are listed separately, along with the dot pattern table for the five by seven printhead matrix. The number of memory bytes required for each system is 2048 (the number available) and the number of instructions required is 867 for the TMS 8080A and 584 for the TMS 9940.

Table 1. *System Firmware Implementation*

Routine	TMS 8080A Microprocessor		TMS 9940 Microcomputer	
	Number of Instructions	Bytes	Number of Instructions	Bytes
Keyboard	260	486	178	472
Printhead	411	855·	291	884
Control	196	367	115	352
Dot Pattern	–	340	–	340
TOTAL	867	2048	584	2048

COST ANALYSIS

Table 2 illustrates the component cost for the two microcomputer systems, assuming a production level of 10,000 units. The component cost of the TMS 8080A System is $48.81, and the cost of the TMS 9940 System is $22.78. In addition, other cost reductions will be realized from savings in incoming test (17 IC's versus two IC's), PC board area (approximately 45 square inches versus 6 square inches), and associated assembly labor and overhead. In total a significant overall cost savings will be realized in the recurring cost of the end product.

Table 2. Component Cost Analysis

TMS 8080A System	$48.81
TMS 9940 System	$22.78

TMS 9900 Floppy Disk Controller

TABLE OF CONTENTS

▶9

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I

INTRODUCTION

This application report describes a TMS 9900 microprocessor system which controls a floppy disk drive and interfaces to an RS-232C type terminal. In addition to providing useful information for the design of a similar system, this application report also shows many of the design considerations for any TMS 9900 microprocessor system design.

The floppy disk is rapidly becoming the most widely accepted bulk storage medium for microprocessor systems. Using standard encoding techniques, a single floppy disk will contain in excess of 400K bytes of unformatted data. Access time to a random record of data is vastly superior to serial media such as cassettes and cartridges, and the medium is both non-volatile and removable.

The use of a microprocessor in the floppy-disk controller or "formatter" is desirable for a number of reasons. The number and cost of components is reduced: this design contains 24 integrated circuits, while random-logic designs typically contain more than 100. The commands from the user interface (in this case, the terminal) to the controller may be more sophisticated, relying on the microprocessor to intrepret the commands. The microprocessor also enables the controller to perform diagnostic functions, both on the controller itself and on its associated drives, not available with a random-logic system.

The Texas Instruments TMS 9900 microprocessor is particularly well-suited to this application. The TMS 9900 is a 16-bit microprocessor capable of performing operations on single bits, bytes, and words. The CRU provides an economical port for bit-oriented input/output, while the parallel memory bus is available for high-speed data. The speed of operation of the TMS 9900 minimizes additional hardware requirements. The powerful memory-to-memory instruction set and large number of available registers simplify software, both in terms of number of assembly language statements and total program memory requirements.

SECTION II

SYSTEM DESCRIPTION

Figure 1 illustrates the relationship of the system elements. Commands are entered by the user at the terminal. These commands are serially transmitted to the controller. The controller interprets the commands and performs the operations specified, such as stepping the read/write head of the drive to a particular track, and reading or writing selected data.

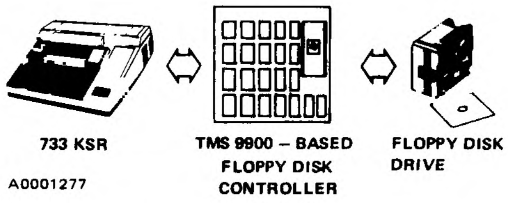


Figure 1. TMS 9900 Floppy Disk Controller System

2.1 DATA TERMINAL

The terminal used in this design is the Texas Instruments 733 KSR Silent Electronic Data Terminal (see Figure 2). Slight modifications to the software will allow the use of virtually any RS-232 terminal.

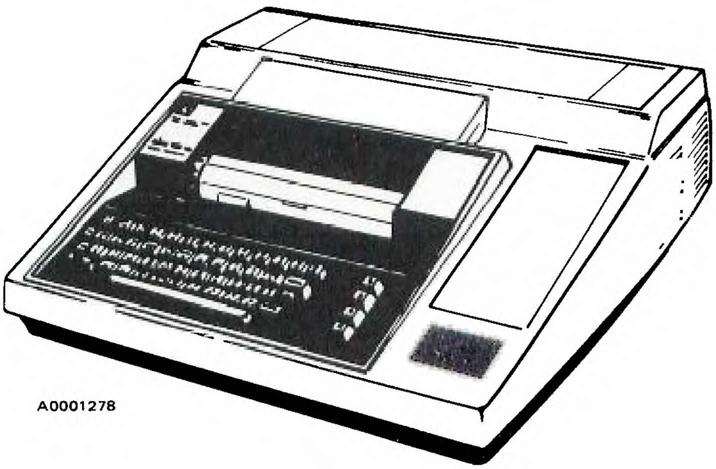


Figure 2. TI 733 KSR Terminal

The 733 KSR consists of a keyboard, printer, and a serial-communication line to the controller. The keyboard enables the operator to enter control commands and data for storage on floppy disc. The printer is used for echoing operator entries, data printout, and reporting of operational errors. The serial interface is full duplex, allowing data transmission both to and from the data terminal simultaneously.

Characters entered on the keyboard are transmitted to the controller in 7-bit ASCII code using asynchronous format, and characters to be printed are sent from the controller to the terminal in the same way. Transmission speed is 300 baud. The format for data transmission is shown in Figure 3.

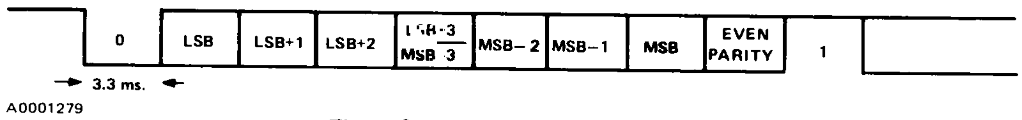


Figure 3. Data Transmission Format

The line idle condition is represented by a logic one. When a character is to be transmitted, the ASCII character is preceded by a zero bit, followed by the 7-bit ASCII code, even parity bit, and the logic-one stop bit. Any amount of idle time may separate consecutive characters by maintaining the logic-one level. Reading data is accomplished by continuously monitoring the line for the one-to-zero transition at the beginning of the start bit. After delaying one-half bit time (1.67 ms) the line is again sampled to ensure that the start bit is valid. If so, the line is sampled each bit time (3.33 ms) until all of the bits of the character have been sampled. The initial one-half bit delay causes subsequent samples to be taken at the theoretical center of each bit, thus providing a margin for distortion due to time base differences between the transmitter and receiver.

The control signals for the terminal are shown in Figure 4.

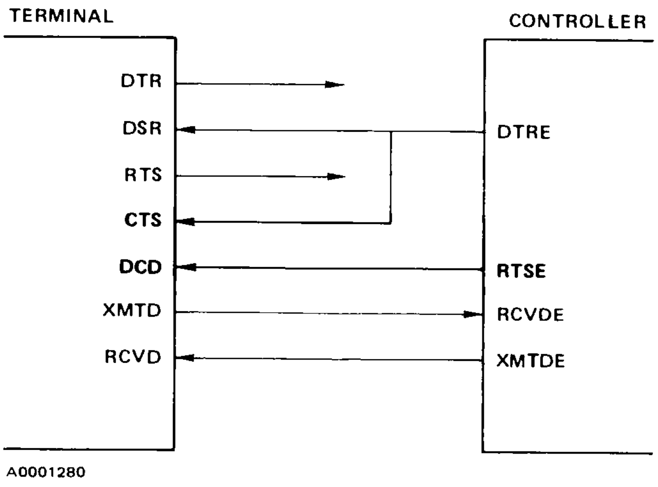


Figure 4. Terminal Interface

9◀

Detailed description of the signals is provided in *Electronics Industriès Association Standard RS-232C.* The signals used in this design are briefly described below.

DTRE – Data Terminal Ready is always on when power is applied to the controller, enabling operation of the serial interface by the terminal.

RTSE – Request to Send is on when a character is transmitted from the controller to the terminal.

XMTDE – Transmitted Data from the controller to the terminal.

RCVDE – Received Data from the terminal to the controller.

Signal levels conform to EIA Standard RS-232C, as shown in Table 1.

Table 1. RS-232C Signal Levels

Voltage Level	Data (XMTDE,RCVDE)	Control (DTRE,RTSE)
−25 to −3 VDC	1	OFF
+3 to +25 VDC	0	ON

The other important parameter for interfacing to the terminal is the amount of time required for a carriage return by the printer, which is 200 ms maximum for the 733 KSR.

2.2 FLOPPY-DISK DRIVE

The floppy-disk drive (Figure 5) is the electromechanical unit in which the recording medium, the floppy disk is inserted. The drive contains the electronics which control the rotation of the floppy disk, the reading and writing of data, and the positioning of the read/write head to select a particular track on the diskette.

2.2.1 Floppy Disk

The floppy disk, or diskette, is the recording medium (see Figure 6). It is enclosed in a plastic protective envelope which keeps foreign particles away from the recording surface. The inner material of the envelope is specially treated to minimize friction and static electricity discharge. The read/write head opening enables the head to come in contact with the recording surface. The index-access hole enables detection of the index hole.

When the index hole in the diskette becomes aligned with the index-access hole, an optical sensor generates the index pulse, providing a reference point for the beginning of each track. There are 77 concentric tracks for recording data. A particular track is accessed by moving the read/write head radially until the desired track is located.

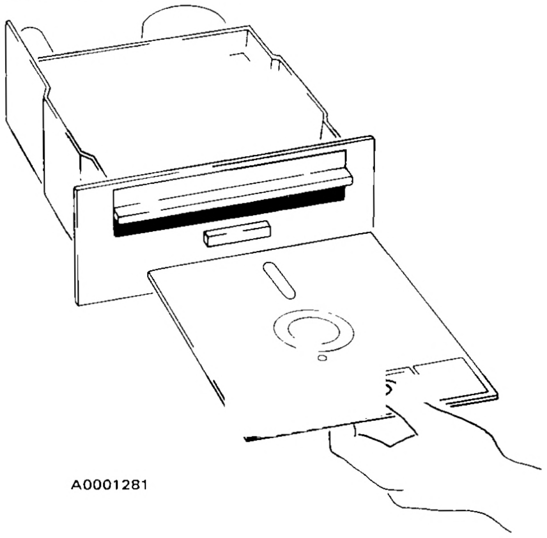


Figure 5. Floppy Disk Drive

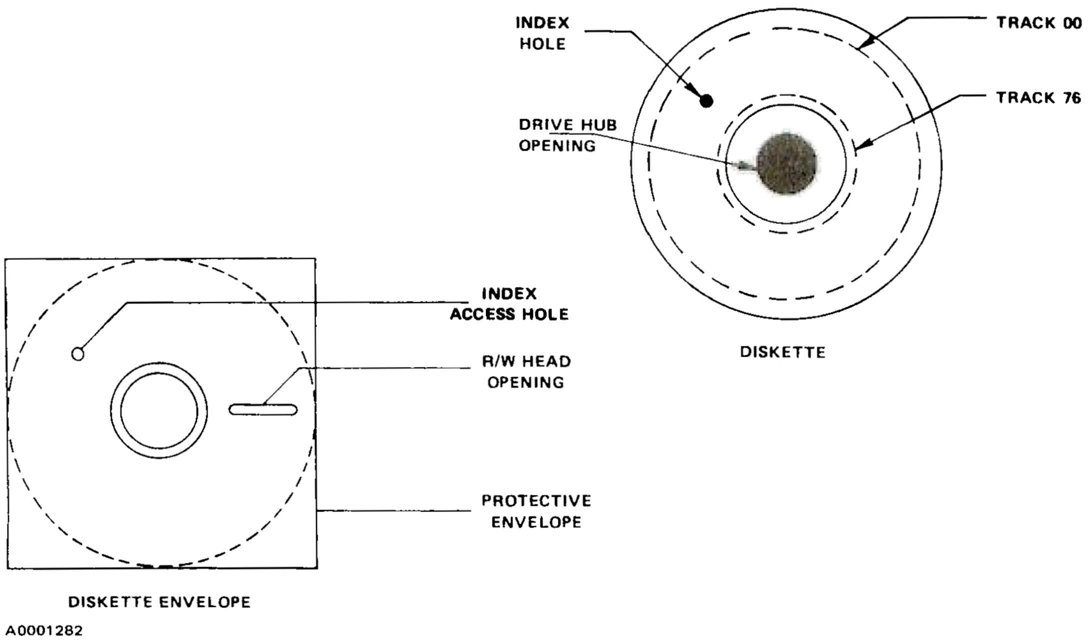


Figure 6. Diskette Envelope and Diskette

2.2.2 Physical Data Structure

The 77 tracks on a diskette are numbered from 00 (outermost) to 76 (innermost). Each track is subdivided into 26 sectors, or records, numbered sequentially from 1 to 26. Each sector consists of two fields: the ID field, which contains sector identification (track and sector number) and the data field, which contains 128 bytes of data.

2.2.3 Encoding Technique

The encoding technique used for representation of data on the diskette is a form of frequency modulation (FM), as shown in Figure 7. Each bit period is 4 microseconds long, resulting in a data-transfer rate of 250K bits per second. A pulse occurs at the beginning of each normal bit period. This pulse is called the clock pulse. If the data bit is a one, a pulse will occur also in the middle of the bit period, 2 μs after the clock bit. If the data bit is a zero, no pulse occurs in the middle of the bit period.

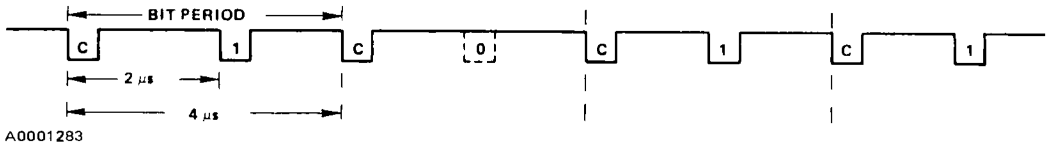


Figure 7. FM Data Pattern 1011

Selected clock bits are deleted in special characters called marks. The absence of the clock bits results in unique sequences, used for synchronization at the beginning of fields.

2.2.4 Track Format

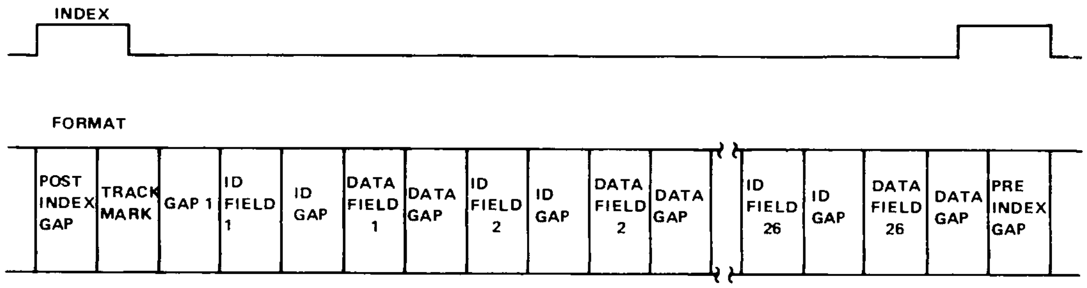
Each track is formatted to provide 26 "soft" sectors. The term soft sectoring means that the beginning of each sector is encoded on the medium through a unique bit sequence. Each of the sectors is separated by a gap of dummy data. Each of the two fields (ID and data) in each sector are also separated by a gap. The first byte of each field is a mark in which the clock pattern for the byte is $C7_{16}$ rather than FF_{16}. The organization of data and clock bits on each track is shown in Figure 8.

2.2.5 Cyclic Redundancy Check Character

The last two bytes at the end of each ID and data field comprise the 16-bit cyclic redundancy check character (CRC). The CRC is generated by performing modulo-2 division on the data portion of the entire field (including the mark) by the polynomial $X^{16} + X^{12} + X^5 + 1$. Before generation of the CRC begins, the initial value is $FFFF_{16}$.

▶9

The analogous hardware operation is illustrated in Figure 9. All flip-flops are initially set to one. Each data bit in the field, beginning with the MSB of the mark byte, is shifted into the logic at DATAIN. The previous



POST INDEX GAP – 46 BYTES, DATA = 00, CLOCK = FF_{16}

TRACK MARK – 1 BYTE, DATA = FC_{16}, CLOCK = $D7_{16}$

GAP 1 – 32 BYTES, DATA = 00, CLOCK = FF_{16}

ID FIELD – 7 BYTES:

	1	2	3	4	5	6	7
CLOCK	$C7_{16}$	FF_{16}	FF_{16}	FF_{16}	FF_{16}	FF_{16}	FF_{16}
DATA	FE_{16}	TRACK NUMBER	00	SECTOR NUMBER	00	CRC1	CRC2

ID GAP – 17 BYTES, DATA = 00, CLOCK = FF_{16}

DATA FIELD – 131 BYTES:

	1	2-129	130	131
CLOCK	$C7_{16}$	FF_{16}	FF_{16}	FF_{16}
DATA	FB_{16} OR $F8_{16}$	DATA	CRC1	CRC2

DATA GAP – 33 BYTES, DATA = 00, CLOCK = FF_{16}

PRE INDEX GAP – 241 BYTES, DATA = 00, CLOCK = FF_{16}

A0001284

Figure 8. Track Recording Format

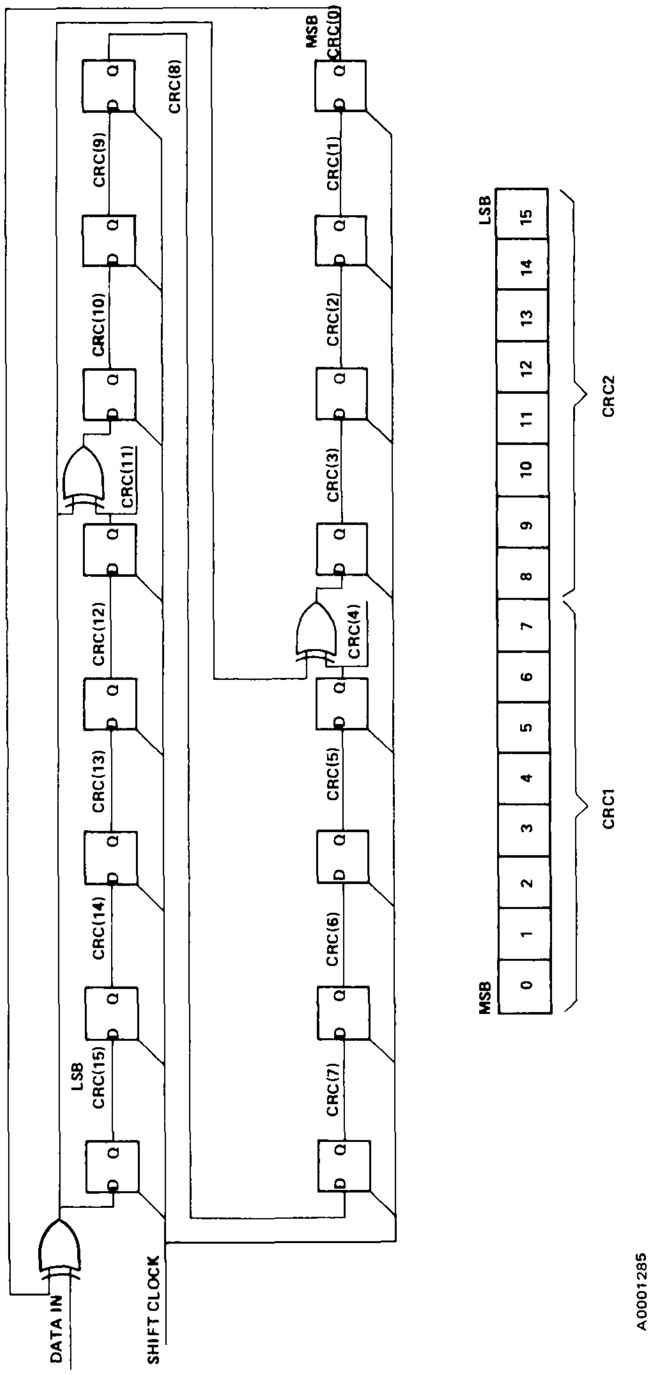


Figure 9. Hardware CRC Generation

▶9

MSB is exclusive ORed with the new input bit to generate a feedback term. This feedback term is stored in the LSB of the register, and is also exclusive ORed with other terms of the CRC. After all data bits of the field have been shifted in, the value in the register is the CRC. The most-significant byte is CRC1 and the least-significant byte is CRC2.

When reading the field, the identical operation is performed, presetting all flip-flops and shifting in all data bits. When reading, it is convenient to also shift in the CRC, causing the resultant value in the register to finally become all zeroes.

In this design, the CRC is calculated by software; however, the algorithm is identical.

2.2.6 Reading Data

The procedure for reading diskette data is as follows:

1. Search the serial-bit string for the ID mark (clock = $C7_{16}$, data = FE_{16}).
2. Read the next four bytes to determine if the desired sector has been located. If not, return to 1.
3. Read the CRC for the ID field and compare it to the expected value. If incorrect, report error and/or return to 1.
4. Search the serial-bit string for either the data mark (clock = $C7_{16}$, data = FB_{16}) or the deleted-data mark (clock = $C7_{16}$, data = $F8_{16}$).
5. Read the next 128 bytes and save.
6. Read the CRC for the data field and compare it to the expected value. If incorrect, report error and/or return to 1.

Normally, if the process is not completed before two index pulses are detected, indicating a complete diskette revolution, the try has failed. Either a retry will be performed, or an error is reported.

2.2.7 Writing Data

When writing data, the sector is located as in steps 1 through 3 above. Then, the ID gap, the data field complete with CRC, and a pad byte (data = 0, clock = FF_{16}) are written.

2.2.8 Track Formatting

The formatting process consists of writing all of the gaps, track mark, ID fields, and data fields, putting dummy data into the data bytes of the data field. After a track is formatted, only the ID gap, data field, and the first byte of the data gap are altered when updating sectors. The number of bytes in the pre-index gap will possibly vary slightly, due to variations in the speed of revolution of the diskette.

2.2.9 Floppy-Disk Timing

Several important timing parameters pertain to the operation of the disk drive:

Bit transfer rate	250,000 bits/second
Track-to-track stepping time	10 milliseconds
Settling time (before read/write)	10 milliseconds
Rotational speed	360 RPM ±2%
Head load time (before read/write)	35 milliseconds

Thus, data is transferred at a rate of 250K bits/second, or 31.25K bytes/second ±2%. Stepping the head each track position requires 10 ms. An additional 10 ms delay must be observed after the final step before reliable data may be written or read. A delay of 35 ms must occur after the head is loaded ($\overline{\text{RDY}}$ = 0) before reliable data may be written or read.

SECTION III

HARDWARE DESCRIPTION

A complete logic diagram of the system is contained in the center of this report. The operation of each section is described separately.

3.1 CLOCK GENERATION AND RESET

The TIM 9904 is used to generate the 4-phase MOS clocks for the TMS 9900 (see Figure 10). Ten ohm resistors are connected in series to the clock lines for damping. The TIM 9904 should always be located physically close to the TMS 9900 to minimize the length of the conductor run for the MOS clocks. The $\overline{\phi 3}$ TTL-level output is used in the synchronous disk read/write control logic.

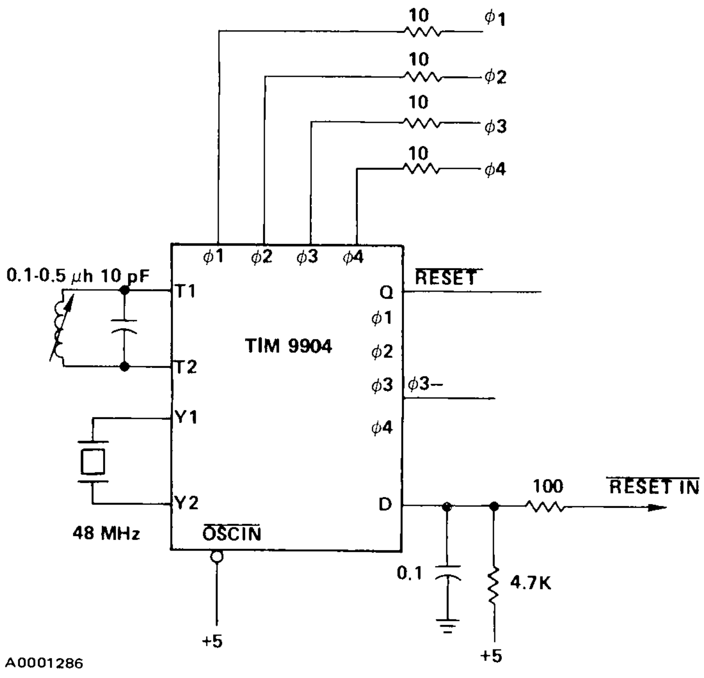


Figure 10. Clock Generation and Reset

9◀

A 48 MHz, third overtone crystal causes the clock frequency to be 3 MHz. The inductor of the LC tank circuit need not be variable; however, in wire-wrap prototypes the capacitance due to interconnect is difficult to predict. The $\overline{\text{OSCIN}}$ input is held high to disable the external clock input.

The RC input to the Schmitt-D input provides power-on detection. The $\overline{\text{RESETIN}}$ input is connected to an external pushbutton. The 100 ohm series resistor reduces contact arcing, thereby extending switch life.

3.2 CPU

The TMS 9900 requires a minimum of external logic. Note that both the data and address buses are connected directly to the memory and disk read/write control logic without buffering as shown in Figure 11. This is due to the ability of the TMS 9900 outputs to sink up to 3.2 mA with 200 pF capacitive load.

The READY input is used to synchronize data transfers to and from the disk read/write control logic, eliminating the need for buffer registers. The $\overline{\text{HOLD}}$, $\overline{\text{LOAD}}$, and interrupt functions are not used in this design and are tied to their inactive (high) level.

3.3 MEMORY CONTROL

Memory control logic, shown in Figure 12, consists of a simple decode of the high-order address lines, enabled by $\overline{\text{MEMEN}}$. Memory enabling signals are generated for EPROM (ROMSEL–), RAM (RAMSEL–), and the disk interface (DISKSEL–). Table 2 shows the memory address assignments.

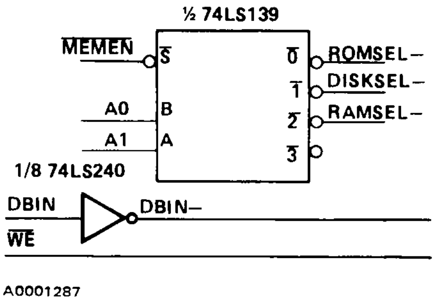


Figure 12. Memory Control

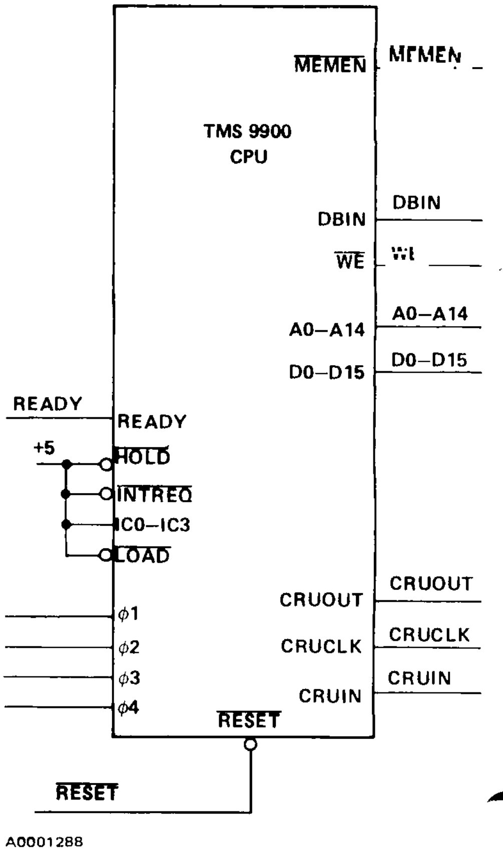


Figure 11. TMS 9900 CPU

Table 2. Memory Address Assignments

Signal	A0	A1	Address Space	Function	Actually Used
ROMSEL–	0	0	000-3FFF	EPROM	000-07FF
DISKSEL–	0	1	4000-7FFF	Disk	7F8E-7FFE
RAMSEL–	1	0	8000-BFFF	RAM	8000-81FF
	1	1	C000-FFFF	Not Used	

Each of the enabling signals will be active when a memory cycle is being performed ($\overline{\text{MEMEN}} = 0$) accessing its address space.

3.4 DISK READ/WRITE SELECT

The DISKSEL signal is further decoded to generate separate select lines for disk read (DISKRD–) and disk write (DISKWT–) operations.

$$\text{DISKRD–} = \overline{(\text{DISKSEL})\,(\text{DBIN})\,(\text{A14–})}\text{, and}$$
$$\text{DISKWT–} = \overline{(\text{DISKSEL})\,(\text{DBIN–})\,(\text{A14})}.$$

Disk read and write operations are specified by different addresses, and are selected only when the DBIN signal is at the proper level for the direction of transfer (see Figure 13). This is required because of the sequence of machine cycles performed by the TMS 9900 when performing a memory-write operation. In the MOV instruction, the CPU first fetches the contents of the memory location to be altered, then replaces this value with the source operand. In this design, the disk read and write operations are controlled by the READY line to synchronize data transfers. If read and write signals were not generated separately, there would be ambiguity with respect to the type of operation desired.

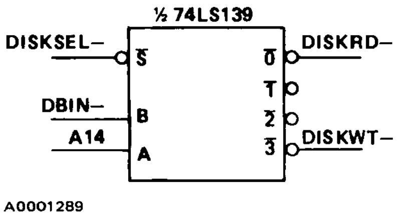


Figure 13. Disk Read/Write Select

This applies to all memory-mapped interfaces in TMS 9900 systems, i.e., the MOV instruction will cause a read operation to precede the write operation to the specified destination address.

3.5 STORAGE MEMORY

Storage memory, shown in Figure 14, is used for implementing workspace registers, maintenance of software pointers and counters, and buffering of a full sector of data.

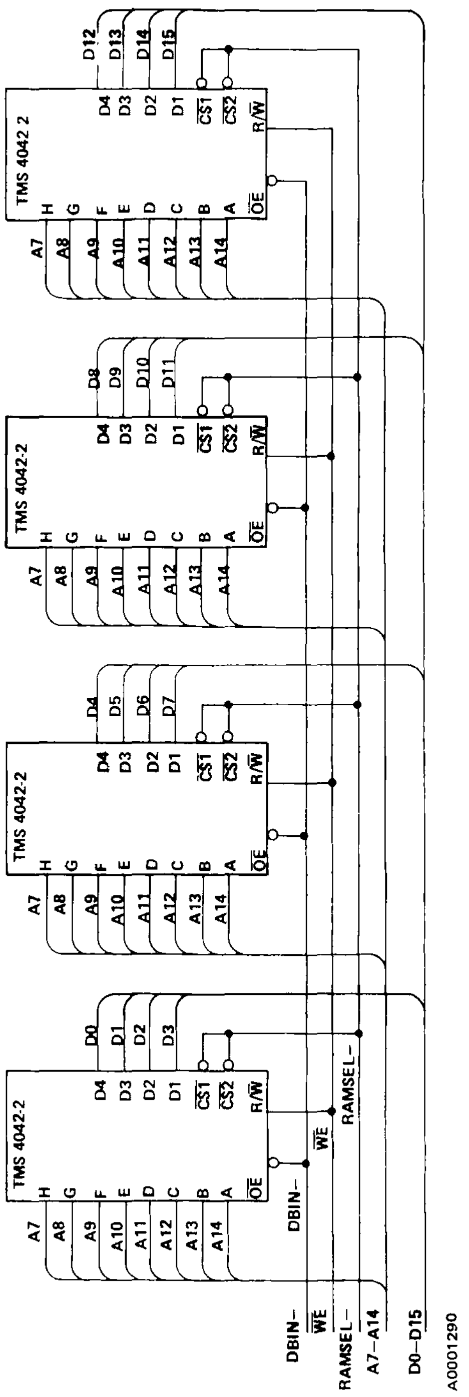


Figure 14. Storage Memory

This design utilizes four TMS 4042-2 RAMs, resulting in a 256-word array of RAM for temporary storage. This 256-word array may be addressed at locations 8000-BFFF, causing each memory location to be multiply defined (e.g., memory address 8000 selects the same word as memory address 8200). For simplificity, RAM will be referred to only as locations 8000-81FF.

Access times for the TMS 4042-2 are sufficiently fast to allow the TMS 9900 to access RAM without any wait states, thus READY will always be true when RAM is addressed. The output enable ($\overline{\text{OE}}$) inputs require that the DBIN output from the TMS 9900 be inverted to gate RAM onto the data bus. The $\overline{\text{WE}}$ output from the TMS 9900 is directly compatible with the R/$\overline{\text{W}}$ input. Data and address lines are connected directly to the CPU.

3.6 PROGRAM MEMORY

Program memory (Figure 15) is used for storage of the machine code program to be executed by the TMS 9900. Also, constants, the RESET vector and XOP vectors are contained in this space.

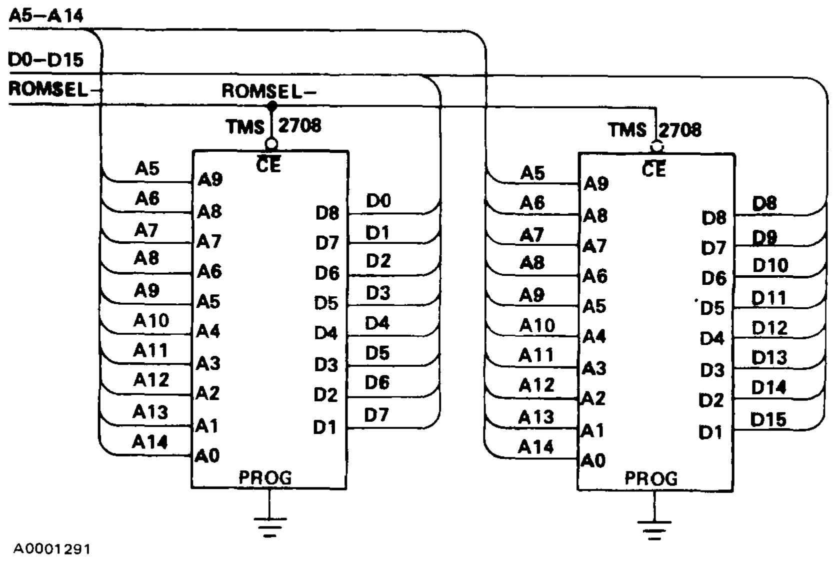


Figure 15. Program Memory

Two TMS 2708 erasable programmable read-only memories (EPROMs) comprise the program memory for this design, resulting in 1024 words of EPROM. EPROM is addressed at memory locations 0000-3FFF. Since these addresses are multiply defined, EPROM will be described only as memory addresses 0000-07FF. Access times for the TMS 2708 are such that no wait states are required.

3.7 CONTROL I/O

All of the control and status signals which require individual testing, setting, or resetting are implemented on the CRU, the bit addressable I/O port for the TMS 9900.

The benefits of using the CRU for these functions is twofold. First, eight bits of input and eight bits of output can be implemented with two 16-pin devices, which are substantially smaller and lower in cost than if these functions were implemented on the parallel-data bus. The second benefit is increased software efficiency. Control and status testing operations can be performed with single one-word instructions, rather than the ORing, ANDing, and maintenance of software images necessary when performing single-bit I/O on the memory bus.

Eight bits of output are implemented with the TIM 9906 8-bit addressable latch. The CRUCLK line must be inverted for input to the TIM 9906. The eight input bits are implemented using the TIM 9905 8-to-1 multiplexer. Individual I/O bits are selected using the three least-significant address lines, A12–A14. The control I/O is illustrated in Figure 16.

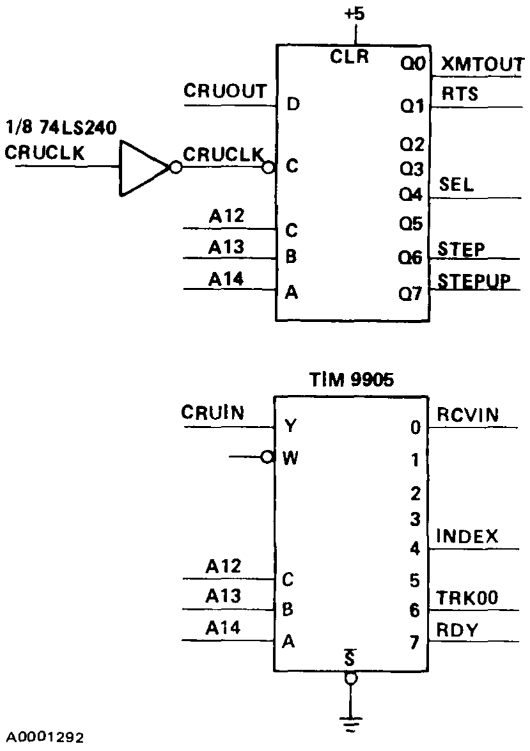


Figure 16. Control I/O

3.8 FLOPPY-DISK-DRIVE INTERFACE

All outputs to the drive are 7406 open-collector, high-voltage and current drivers. Pullups for the output signals are provided in the drive electronics. All inputs are terminated by 150 ohm pullup resistors to +5 volts, and are buffered and inverted. All input and output signals are active low.

$\overline{\text{SEL}}$ – Active when a stepping operation or a data transfer is being performed.

$\overline{\text{RDY}}$ – Active when the disk is ready to perform a stepping or transfer operation (i.e., $\overline{\text{SEL}} = 0$, diskette is in place, door is closed, power is furnished to the drive).

$\overline{\text{STEP}}$ – A minimum 10 μs pulse causes the read/write head to move one track position in the direction selected by $\overline{\text{STEPUP}}$.

$\overline{\text{STEPUP}}$ – When $\overline{\text{STEPUP}}$ = 0, the read/write head moves in one track position. When $\overline{\text{STEPUP}}$ = 1, the head will move out (toward track 00).

$\overline{\text{TRK00}}$ – Active when the read/write head is located on the outermost track (track 00).

$\overline{\text{INDEX}}$ – As the diskette rotates in the drive, the index pulse occurs once per revolution, providing a reference point for the beginning of each track.

$\overline{\text{WRITE ENABLE}}$ – This signal must be active a minimum of 4 μs before a write operation begins, and must be maintained active during the entire write operation.

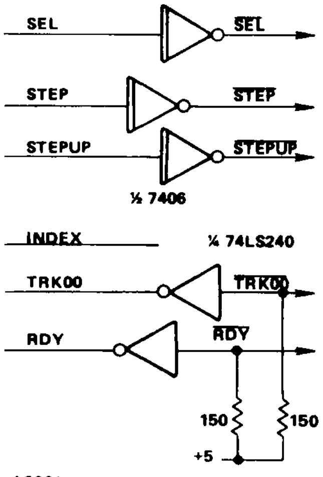
$\overline{\text{WRITE DATA}}$ – This signal contains a series of pulses representing the data to be written to the disk in the FM format previously described.

$\overline{\text{READ DATA}}$ – This signal contains a series of pulses representing the data to be read from the disk in the FM format previously described.

Figure 17 illustrates the floppy-disk-drive interface.

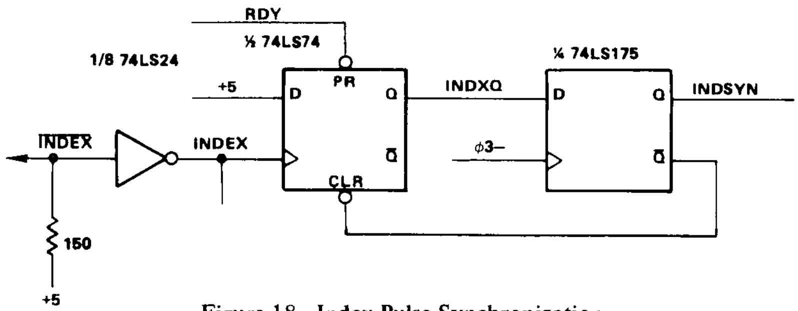
3.9 INDEX PULSE SYNCHRONIZATION

Since the index pulse is a term in some of the expressions that are sampled by the CPU, it must be synchornous to the CPU. The circuit shown in Figure 18 generates a signal one φ3 clock cycle long at the beginning of each index pulse from the drive. RDY will be inactive when the drive is turned off or the door is open, thus connection of RDY to the preset input of the flip-flop shown causes INDSYN to be active as long as RDY = 0 (see Figure 19). Forcing INDSYN to be one when RDY = 0 prevents the CPU from remaining in a wait state when the drive is disabled during data transfer.



A0001293

Figure 17. Floppy-Disk Drive Interface



A0001294

Figure 18. Index-Pulse Synchronization

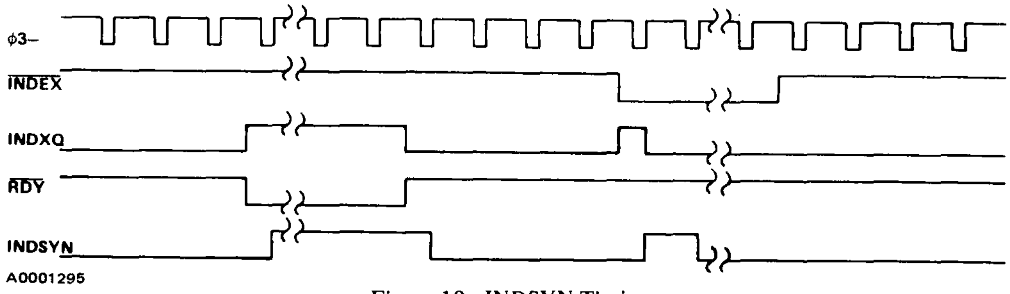


Figure 19. INDSYN Timing

3.10 READ PULSE SYNCHRONIZATION

The read-pulse synchronization logic, Figure 20, generates an active signal, BITIN, one clock cycle long each time a read pulse is detected during read operations. During write operations BITIN is maintained at a logic-one level.

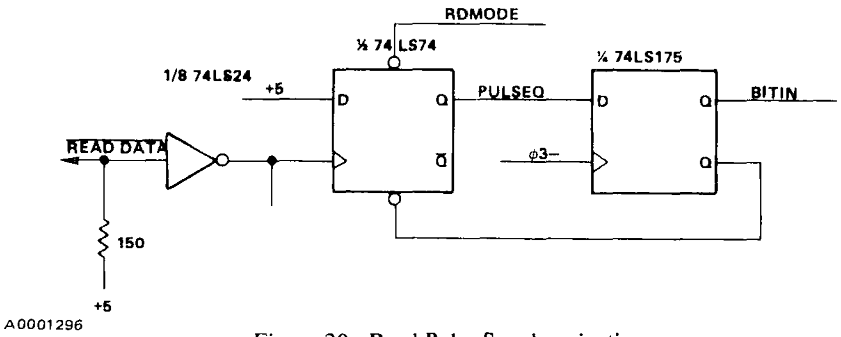


Figure 20. Read-Pulse Synchronization

3.11 BIT DETECTOR

The bit detector, Figure 21, consists of a 74LS163 counter and random logic contained in PROM. During write operations, the counter is used to time the 2 μs spacing between clock bits and data bits. During read operations the bit detector is used to determine the time interval between successive read pulses. The key signal generated by the bit detector is BITTIME, which is active for one clock cycle every 2 μs during disk writing, and which is active each time a one or zero bit is detected during read operations.

3.12 BIT COUNTER

The bit counter, Figure 22, is a 74LS163 used to count the number of bits currently read or written during disk-data transfers. Each time a clock or data bit is detected or written (BITTIME = 1) the bit counter is

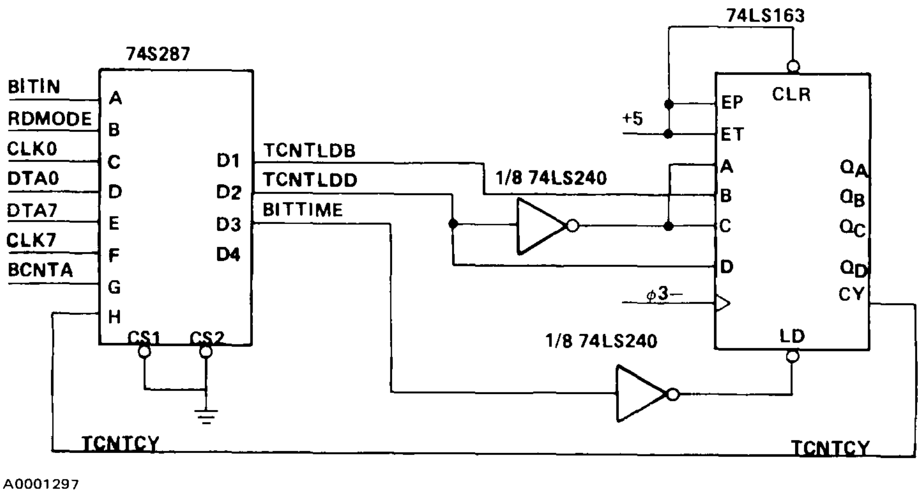


Figure 21. Bit Detector

incremented. The two key outputs are BCNTA and BCNT = 15. BCNTA is the least-significant bit of the counter and is used to alternately select clock (BCNTA = 0) and data (BCNTA = 1) bits as the counter increments. BCNTA = 15 is active when a complete byte has been read or written. This signal establishes byte boundaries for the data and is used to synchronize the parallel data from the CPU to the serial-bit string and from the disk.

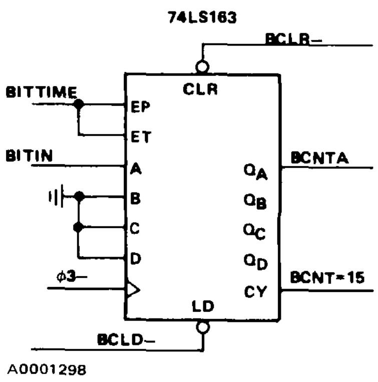


Figure 22. Bit Counter

3.13 WRITE CONTROL AND DATA

Writing to the diskette is controlled by $\overline{\text{WRITE ENABLE}}$, which is the inverted and buffered WTMODE signal. WTMODE is active when a write operation has been initiated by the CPU. The $\overline{\text{WRITE DATA}}$ signal is a series of negative pulses representing FM data to be recorded on the diskette. Figure 23 illustrates write control and data.

3.14 DATA SHIFT REGISTER

The data shift register, see Figure 24, is used for accumulation of data bits during read operations and storage of data bits to be shifted out during write operations. Data is transferred to and from the CPU via the eight most-significant data lines (D0–D7). The data shift register is device type 74LS299.

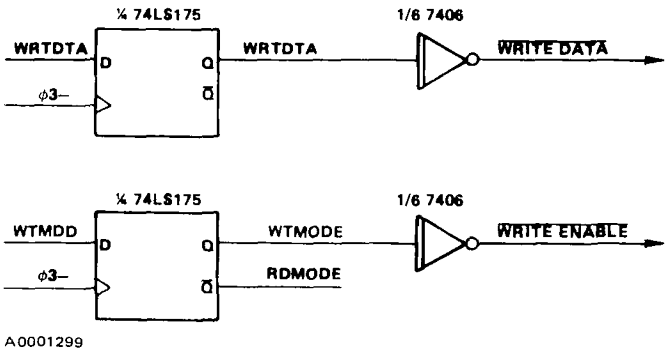


Figure 23. Write Control and Data

3.15 CLOCK SHIFT REGISTER

The clock shift register, Figure 25, is used for accumulation of clock bits during read operations and storage of clock bits to be shifted out during write operations. The clock shift register is device type 74198, which has separate parallel inputs and outputs. Three address lines, A9–A11, are connected to the parallel inputs. As data is loaded into the data shift register during write operations, these three address lines select the clock pattern for that byte (i.e., C7 for ID and data marks, D7 for track mark, FF for normal data). The parallel outputs (CLK0-CLK7) are used to detect mark clock patterns during read operations.

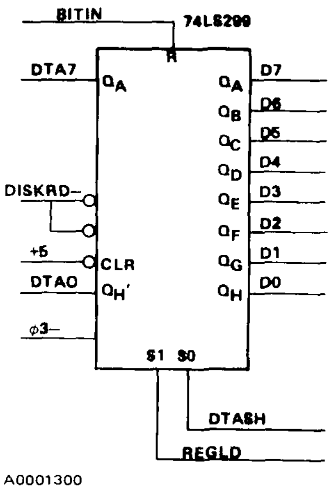


Figure 24. Data Shift Register

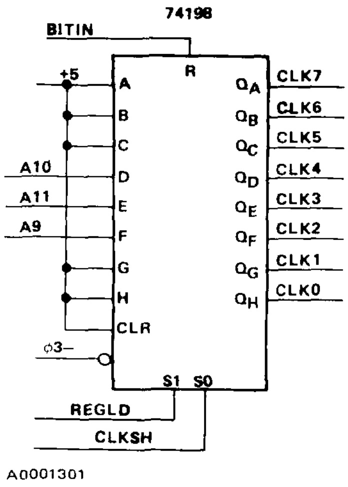


Figure 25. Clock Shift Register

SECTION IV

DISKETTE DATA TRANSFER

The previous section described the various functional blocks in the TMS 9900 floppy-disk controller. However, detailed information was not provided with respect to the logical relationships and timing of the control signal in the read/write control logic.

Most of the read/write control logic varies in function depending on the direction of transfer. This section will describe the operation of the logic separately for read and write operations. After both operations have been completely described, the combined operation will be explained.

4.1 DISK-WRITE OPERATIONS

Disk writing is initiated by executing an instruction which writes data to the data shift register (i.e., when DISKWT– = 0). When this transfer occurs, READY is held low until a byte boundary occurs (BCNT = 15), then READY becomes active, permitting completion of the write cycle. In this way, the data transfers are synchronized to the serial bit string.

To complete the transfer, READY must be active to the CPU, and the CLKSH, DTASH, and REGLD signals to the clock and data shift registers must be active to permit loading. READY = CLKSH = DTASH = REGLD = (DISKWT) (A13) (BCNT = 15) + . . .

The preceding equation indicates that the disk write must be performed with A13 = 1 for data transfer on byte boundaries. When formatting a track, the write operation must be synchronized with the index pulse, and the bit counter must be cleared regardless of its current state. When this type of write operation is to be performed, A13 must be 0.

READY = CLKSH = DTASH = REGLD = (DISKWT) (A13) (BCNT = 15) + (DISKWT) (A13–) (INDSYN) + . . .

BCLR– = $\overline{\text{(DISKWT) (A13–) (INDSYN) + . . .}}$

As the data byte is loaded into the data shift register, address lines A9, A10, and A11 select the clock pattern to be loaded into the clock shift register (see Table 3).

Table 3. Write Clock Patterns

A9	A10	A11	Clock Pattern
0	0	0	C7 (ID and Data Mark)
0	0	1	D7 (Track Mark)
1	1	1	FF (Normal Data)

When the transfer is complete to the clock and data shift registers, the write mode (WTMODE) flip flop is set, causing $\overline{\text{WRITE ENABLE}}$ to become active. If another byte is not written at the next byte boundary, WTMODE is reset, causing the control logic to revert to the read mode (RDMODE = 1). Also, control reverts to read mode and the bit counter is cleared when the index pulse occurs and when no write operation synchronized to the index pulse is being performed. This is useful when formatting a track, since $\overline{\text{WRITE ENABLE}}$ will automatically be turned off when the second index pulse occurs. If an index pulse occurs during a write operation with A13 = 1, the CPU proceeds, but no data transfer takes place.

WTMDD = (WTMODE) (BCNT = 15–) (INDSYN–) + (DISKWT) (A13) (BCNT = 15) + (DISKWT) (A13–) INDSYN)

BCLR– = $\overline{\text{INDSYN} + \ldots}$

READY = (DISKWT) [(A13) (BCNT = 15) + INDSYN)] + . . .

While WTMODE = 1, write data is generated by alternately shifting out bits from the clock and data shift register every two microseconds. Shifting of the clock shift register occurs when CLKSH = 1, and shifting of the data shift register when DTASH = 1. The shift is enabled by BITTIME, which is active for one clock cycle every 2 μs by loading the counter with 10_{10} each time TCNTCY = 1.

BITTIME = (WTMODE) (TCNTCY) + . . .

TCNTLDD = TCNTLDB = WTMODE + . . .

CLKSH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA–) (BITTIME) + . . .

DTASH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA) (BITTIME) + . . .

WRTDTAD = (WTMODE) (BITTIME) [(CLK0) (BCNTA–) + (DTA0) (BCNTA)]

On even bit counts (BCNTA = 0) clock bits are shifted, and on odd bits (BCNTA = 1) data bits are shifted, producing the desired interleaving of clock and data bits. (See Figure 26.)

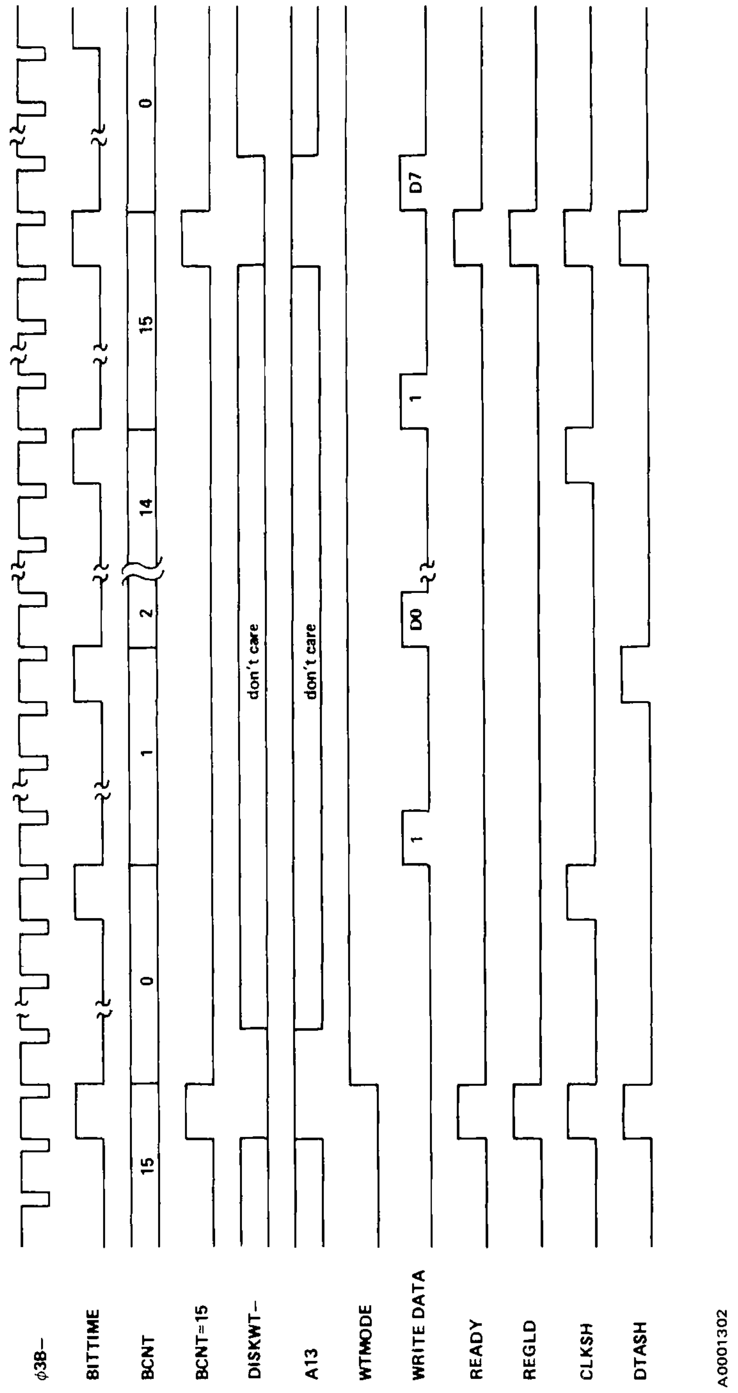


Figure 26. Write Timing

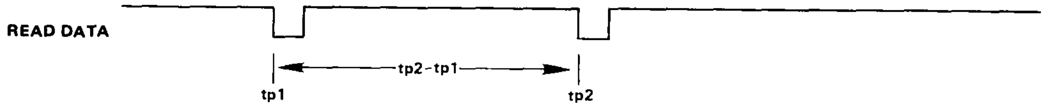
4.2 DISK READ OPERATIONS

Any time disk write operations are not being performed, the read/write control logic defaults to the read mode (RDMODE = 1). The following functions are performed to enable the CPU to read diskette data:

1. Conversion of FM to digital data;
2. Separation of clock and data bits;
3. Byte synchronization of the bit string;
4. Assembly of the seria data into bytes to be ready by CPU.

4.2.1 Clock and Data Bit Detection

Clock and data bits read from the disk are represented as a series of pulses. Each logic one clock or data bit is simply a pulse. Logic zero data and clock bits are indicated by the absence of a pulse between two pulses separated by a full data period (4 μs). Under ideal circumstances, detection of zero bits could be achieved by simply measuring the time between pulses. If $t_{P2}-t_{P1}$ = 2 μs, no zero bit is present; and if $t_{P2}-t_{P1}$ = 4 μs, a zero bit occurs between the two pulses.



Three phenomena make zero-bit detection more complex:

1. Variations in rotational speed of the disk;
2. Uncertainty of measured delays when using synchronous counters;
3. Apparent positional distortion or "bit-shifting" resulting from the tendency of pulses to move away from adjacent pulses.

Disk speed variations are typically specified at ±2% by diskette drive manufacturers. Figure 27 illustrates the bit shifting phenomenon:

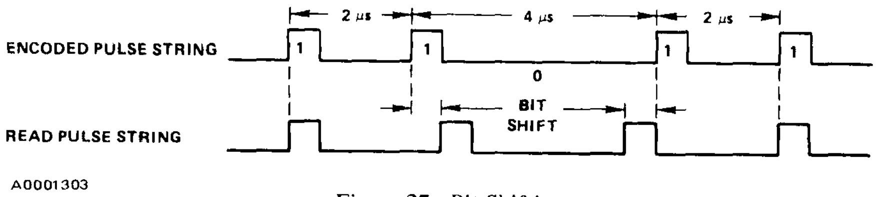


Figure 27. Bit Shifting

9

Pulses in the string have a tendency to move away from each other, and the closer together the pulses, the stronger the tendency to separate. A zero bit causes contiguous pulses to move toward each other, reducing pulse separation and complicating zero detection.

The bit detector is used to generate the synchronous signal BITTIME, which is active when a one or zero bit has been detected.

BITTIME = (RDMODE) (BITIN) + . . .

Detection of zero bits is accomplished by measuring the time between successive pulses. When TCNTCY = 1 and BITIN = 0, a zero bit is detected.

BITTIME = (RDMODE) (BITIN + TCNTCY) + . . .

Data and clock bits could be detected by measuring the time between read pulses, and if this time is greater than 3 μs, a zero bit is present; otherwise, no zero bit is present. Since the read pulse is asynchronous to the system, the time between pulses can only be measured to an accuracy of 333 ns (±1 clock cycle). For example, if the counter in Figure 28 is loaded with seven, no zero will be detected if the time between pulses ($t_{P2} - t_{P1}$) is less than 3.0 μs, and a zero will always be detected if $t_{P2} - t_{P1} > 3.333$ μs. If 3.0 μs $< t_{P2} - t_{P1} <$ 3.333 μs, an ambiguity occurs in that a zero may or may not be detected. Similarly, if the counter is loaded with eight rather than seven, no zero bit will be detected if $t_{P2} - t_{P1} < 2.667$ μs, a zero bit will be detected if $t_{P2} - t_{P1} > 3.0$ μs, and the result is indeterminate if 2.667 μs $< t_{P2} - t_{P1} <$ 3.0 μs. Most floppy-disk drive manufacturers specify that the maximum shift for any bit is 500 ns. Thus, two consecutive 1 bits may be separated by nearly 3.0 μs, and two 1 bits separated by a zero bit may shift toward each other to result in a minimum separation of nearly 3.0 μs. The combined distortion of consecutive 1 bits never fully reaches 1 μs, but the 667 ns margin provided by loading the counter with either seven or eight does not provide for reliable, accurate reading of data. (See Figure 28.)

As stated previously, adjacent 1 bits affect the direction of distortion of a particular 1 bit, with the closest pulses having the greatest effect. Empirical observation indicates that only the two bit positions on either side of a pulse have significant effect on a pulse, as shown in Table 4.

Table 4. Bit Shift Direction

Bit n-2	Bit n-1	Bit n	Direction of Distortion For Bit n	Bit n+1	Bit n+2
0	1	1	→	0	1
0	1	1	–	1	0
0	1	1	←	1	1
1	0	1	–	0	1
1	0	1	←	1	0
1	0	1	←	1	1
1	1	1	→	0	1
1	1	1	→	1	0
1	1	1	–	1	1

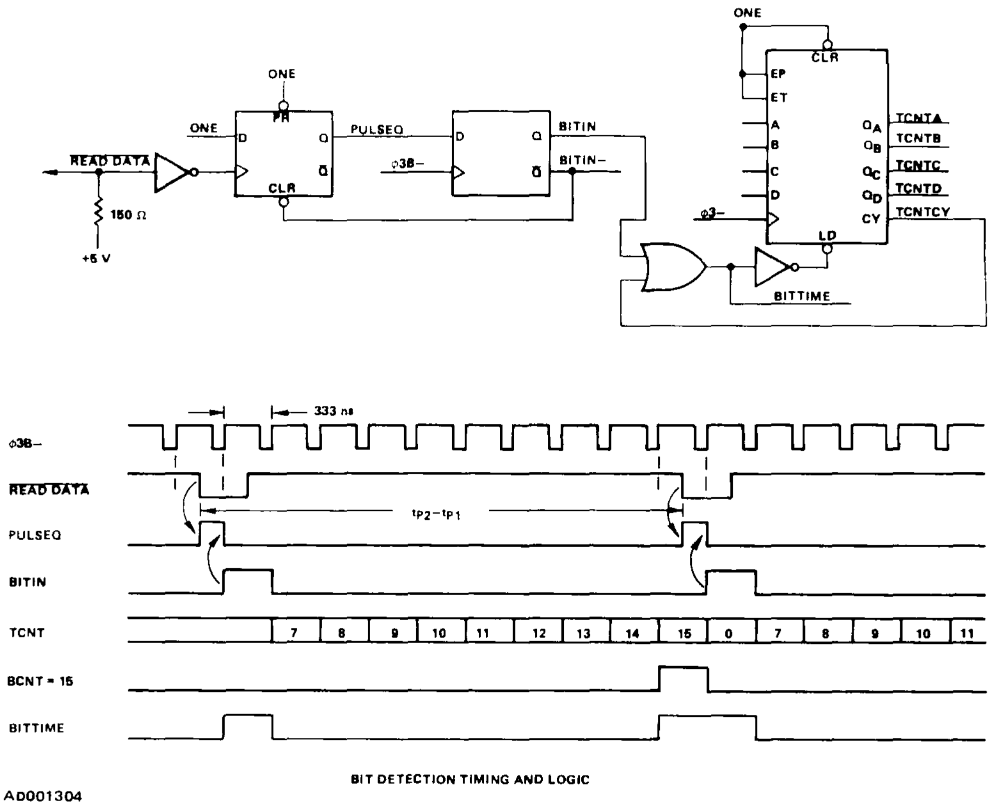


Figure 28. Bit Detection Timing and Logic

The most difficult detection problem is that of differentiating between two contiguous 1 bits which are shifted away from each other (worst case 11) and two 1 bits separated by a zero bit where the 1 bits move toward each other (worst case 101). The worst case 11 occurs in the patterns

			←	→			
Pattern A	0	1	1	1	1	0	, and
Pattern B	1	0	1	1	0	1	.
			→	←			

The worst case 101 occurs in the patterns

			→		←		
Pattern C	0	1	1	0	1	1	, and
Pattern D	1	1	1	0	1	1	.
			→		←		

The timing logic is such that the period of uncertainty does not lie in the area where a severely distorted pulse will occur; that is, when the worst case 11 can occur, and $t_{P2} - t_{P1} < 3.0\,\mu s$, the logic always indicates that no zero was detected; when the worst case 101 can occur and $t_{P2} - t_{P1} > 3.0\ \mu s$, a zero is always detected. To accomplish this, the value loaded into the counter is shown in Table 5.

Table 5. Worst Case Pattern Load Values

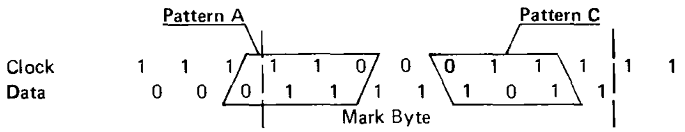
Pattern	Bit n−2	Bit n−1	Bit n	Bit n+1	Bit n+2	Bit n+3	Load Value
A	0	1	1	1	1	0	7
B	1	0	1	1	0	1	7
C	0	1	1	0	1	1	8
D	1	1	1	0	1	1	8

When bit n is detected, the counter is loaded with the value shown, dependent upon the data pattern.

Accommodation of patterns B and D are simple, since bits following that being sampled don't matter. Patterns A and C present the problem that, as the serial pulses are being read, the logic does not know what bits n+1, n+2, and n+3 are going to be.

Further analysis of the data format reveals that patterns A and C occur only when an ID or data mark are being read, see Table 6.

Table 6. Data Mark



Pattern A can only occur at the beginning of an ID, data, or deleted data mark, and pattern C can only occur in a data mark. With pattern A, the first 0 is a data bit, and with pattern C, the first 0 is a clock bit. BCNTA selects whether the current 1 bit is to be shifted into the clock or data shift register. The previous two bits are CLK7 and DTA7, the LSB's of the clock and data shift registers, and the order of these bits is determined by BCNTA. Using this information, the values loaded into the counter are as shown in Table 7.

TCNTLDD = (RDMODE)](CLK7) (DTA7) + (BCNTA–) (DTA7)] + . . .

TCNTLDB = (RDMODE) [(DTA7–) + (BCNTA) (CLK7–)] + . . .

The bit detector will thus adjust its count interval to accommodate the worst-case distortion which can occur for the anticipated data pattern.

Table 7. Bit Detector Counter Load Values

BCNTA	CLK7	DTA7	Load Value
0	0	0	Illegal
0	0	1	8
0	1	1	8
0	1	0	7
1	1	0	7
1	1	1	8
1	0	1	7
1	0	0	Illegal

4.2.2 Clock/Data Separation

Each time BITTIME is active, a new clock or data bit is shifted in. The value of the clock or data bit is BITIN. Since clock and data bits are interleaved, the value of BITIN will be alternately shifted into the clock or data shift register each time BITTIME is active. This is accomplished by incrementing the bit counter each time BITTIME is active, causing BCNTA to toggle. The equations for shifting the clock and data shift registers are:

CLKSH = (BITTIME) (BCNTA–) (RDMODE)+ . . .

DTASH = (BITTIME) (BCNTA) (RDMODE) + . . .

When four consecutive zeroes are detected in the clock shift register, the order in which bits go to the clock and data shift registers is reversed, since four consecutive zero clock bits never occur in the recording format used. This is accomplished by the control signal:

BCLD– = $\overline{\text{(CLK4–) (CLK5–) (CLK6–) (CLK7–)}}$.

When this signals becomes active, the bit counter is cleared to zero, and remains cleared until the next 1 bit is detected. This 1 bit is directed to the clock shift register, causing BCLD– to become inactive and normal operation is resumed. Synchronization is thus assured at the beginning of each ID and data field because each field is preceded by several bytes with all zero data bits and all one clock bits.

The timing for clock/data separation is shown in Figure 29.

4.2.3 Byte Synchronization

9

Initial byte synchronization is achieved when reading an ID or data field by detecting the unique clock pattern of $C7_{16}$ which occurs only in ID and data marks. The mark detect signal is expressed by the equation:

MRKDT = (CLK0) (CLK1) (CLK2–) (CLK3–) (CLK4–) (CLK5) (CLK6) (CLK7)

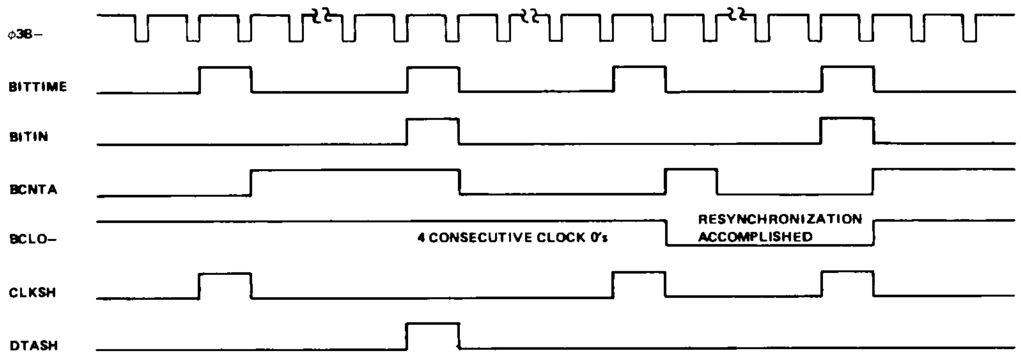


Figure 29. Clock/Data Separation Timing

After the mark is detected, one additional BITTIME must occur, allowing the data bit to be shifted into the data shift register.

4.2.4 Reading Disk Data

Two types of disk reads may be performed. When reading an ID or data field, the first byte read is always the ID or data mark. This is accomplished by performing a disk read with A13 = 0. The READY input signal will not become active until MRKDT = 1 and BITTIME = 1. After the mark is read, byte synchronization is established and subsequent disk reads are performed with A13 = 1. In this case, READY becomes true at each byte boundary when BCNT = 15.

READY = (DSKRD) [(BCNTA) (MRKDT) (BITTIME) (A13–) + (BCNT = 15) (A13) + INDSYN] + . . .

The addresses for the two types of disk reads are $7FF8_{16}$ for reading marks, and $7FFC_{16}$ for reading normal data. The INDSYN term of the above equation causes the read operation to be completed any time the index pulse is detected or when the disk becomes not ready. (See Figure 30.)

4.3 READ/WRITE LOGIC COMBINATION

This subsection summarizes the equations for the control lines resulting from the combination of the read and write control functions.

BCLD–
BCLD– = $\overline{\text{(CLK4–) (CLK5–) (CLK6–) (CLK7–)}}$

BCLR–
BCLR– = $\overline{\text{(RDMODE) (MRKDT) (BCNTA) (BITTIME) + (INDSYN)}}$

9

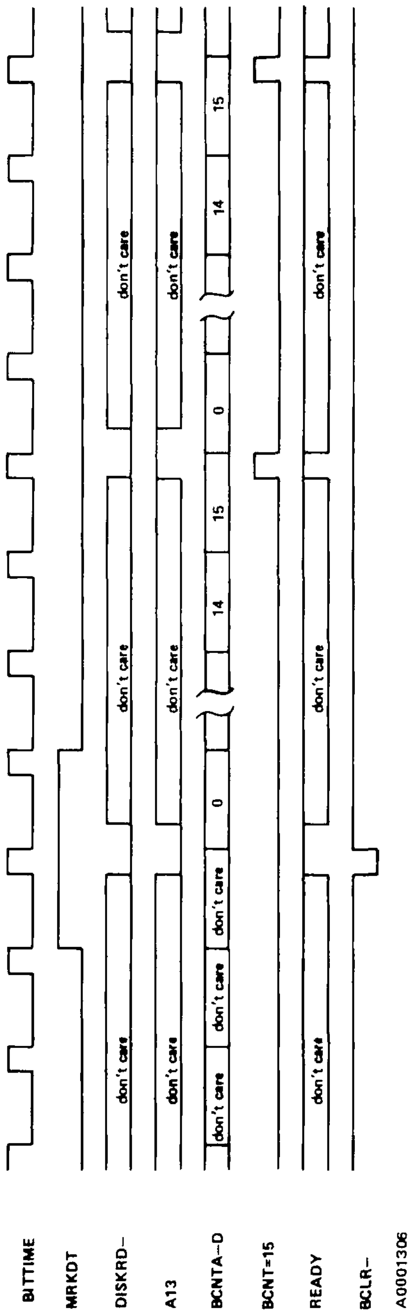
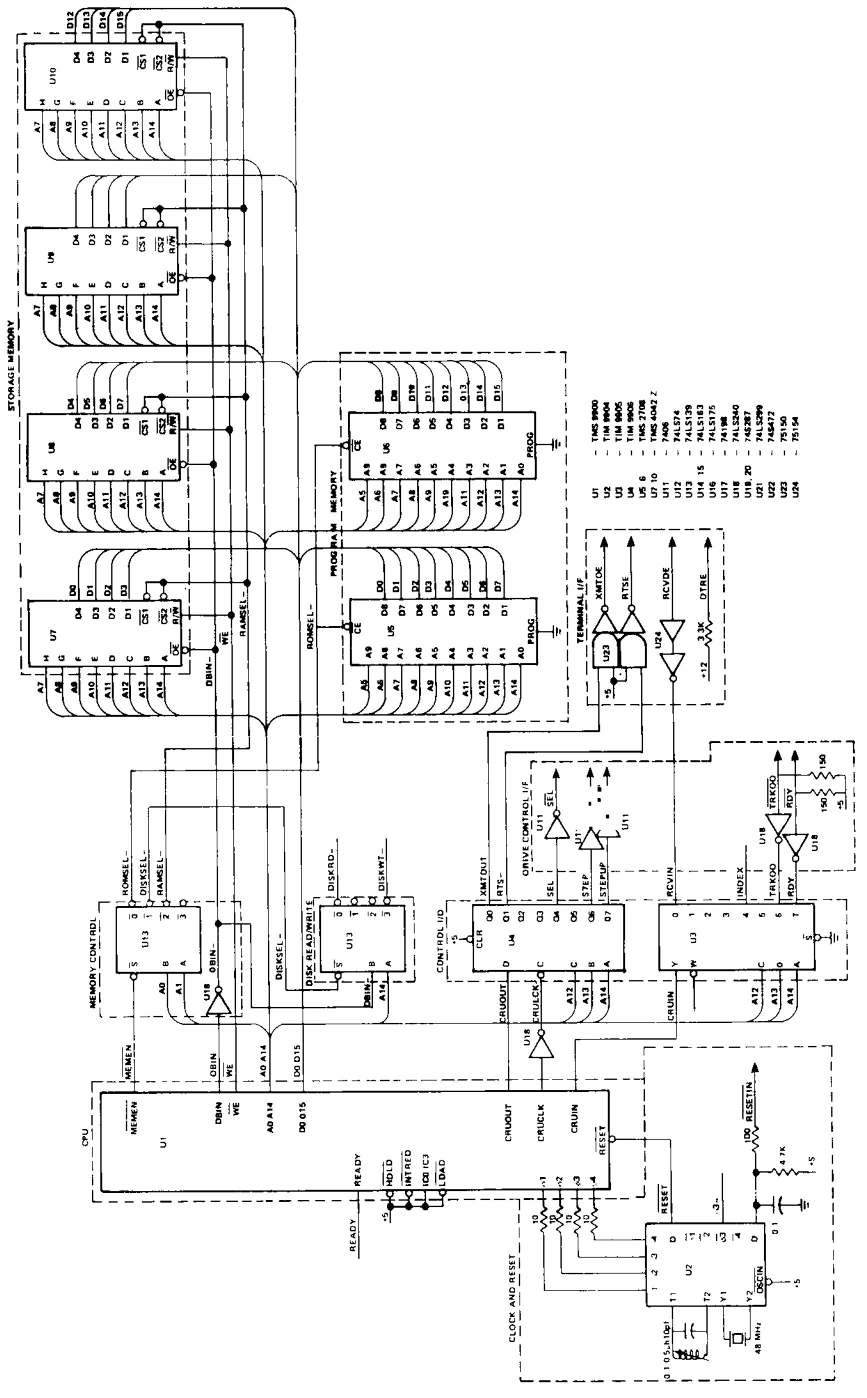
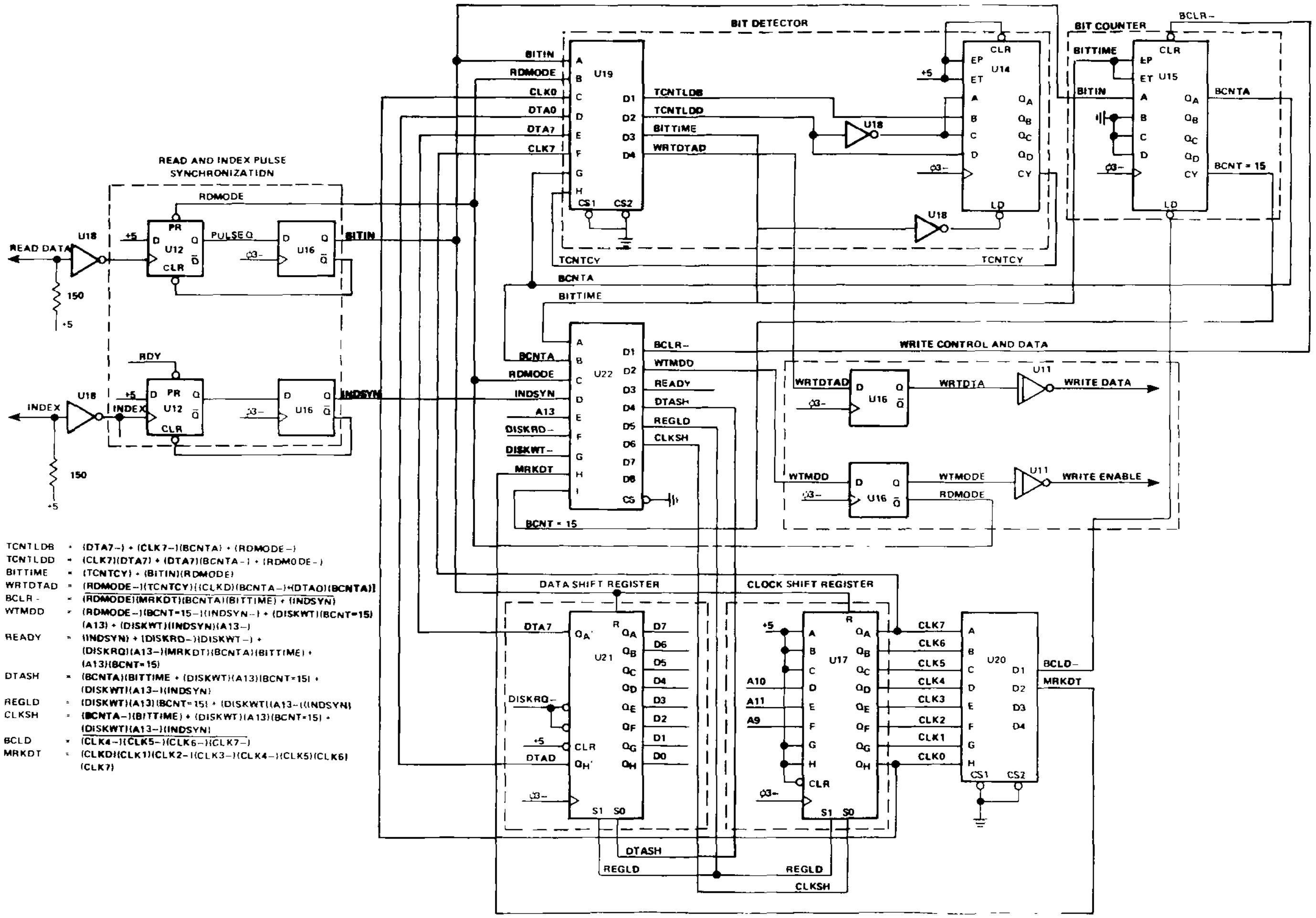


Figure 30. Disk Read Timing



Logic Diagram, TMS 9900 Floppy Disk Controller
(Sheet 1 of 2)



TCNTLDB	=	(DTA7–) + (CLK7–)(BCNTA) + (RDMODE–)
TCNTLDD	=	(CLK7)(DTA7) + (DTA7)(BCNTA–) + (RDMODE–)
BITTIME	=	(TCNTCY) + (BITIN)(RDMODE)
WRTDTAD	=	(RDMODE–)(TCNTCY)[(CLKD)(BCNTA–)+(DTA0)(BCNTA)]
BCLR–	=	$\overline{\text{(RDMODE)(MRKDT)(BCNTA)(BITTIME)}}$ + (INDSYN)
WTMDD	=	(RDMODE–)(BCNT=15–)(INDSYN–) + (DISKWT)(BCNT=15)(A13) + (DISKWT)(INDSYN)(A13–)
READY	=	(INDSYN) + (DISKRD–)(DISKWT–) + (DISKRQ)(A13–)(MRKDT)(BCNTA)(BITTIME) + (A13)(BCNT=15)
DTASH	=	(BCNTA)(BITTIME + (DISKWT)(A13)(BCNT=15) + (DISKWT)(A13–)(INDSYN)
REGLD	=	(DISKWT)(A13)(BCNT=15) + (DISKWT)(A13–)(INDSYN)
CLKSH	=	(BCNTA–)(BITTIME) + (DISKWT)(A13)(BCNT=15) + (DISKWT)(A13–)(INDSYN)
BCLD	=	$\overline{\text{(CLK4–)(CLK5–)(CLK6–)(CLK7–)}}$
MRKDT	=	(CLKD)(CLK1)(CLK2–)(CLK3–)(CLK4–)(CLK5)(CLK6)(CLK7)

A0001307

Logic Diagram, TMS 9900 Floppy Disk Controller
(Sheet 2 of 2)

BITTIME
BITTIME = (WTMODE) (TCNTCY) + (RDMODE) [(BITIN) + (TCNTCY)]
= (TCNTCY) + (RDMODE) (BITIN)

CLKSH
CLKSH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA–) (BITTIME) + (RDMODE) (BCNTA–) (BITTIME)
= (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (BCNTA–) (BITTIME)

DTASH
DTASH = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (WTMODE) (BCNTA) (BITTIME) + (RDMODE) (BCNTA) (BITTIME)
= (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)] + (BCNTA) (BITTIME)

MRKDT
MRKDT = (CLK0) (CLK1) (CLK2–) (CLK3–) (CLK4–) (CLK5) (CLK6) (CLK7)

READY
READY = (DISKWT) [(A13) (BCNT = 15) + (INDSYN)] + (DISKWT–) (DISKRD–) + (DISKRD) [(A13) (BCNT = 15) + (INDSYN) + (A13–) (MRKDT) (BCNTA) (BITTIME)]
= (DISKWT–) (DISKRD–) + (A13) (BCNT = 15) + (INDSYN) + (DISKRD) (A13–) (MRKDT) (BCNTA) (BITTIME)

REGLD
REGLD = (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)]

TCNTLDB
TCNTLDB = (WTMODE) + (RDMODE) [(DTA7–) + (BCNTA) (CLK7–)]
= (WTMODE) + (DTA7–) + (BCNTA) (CLK7–)

TCNTLDD
TCNTLDD = (WTMODE) + (RDMODE) [(CLK7) (DTA7) + (BCNTA–) (DTA7)]
= (WTMODE) + (CLK7) (DTA7) + (BCNTA–) (DTA7)

WRTDTAD
WRTDTAD = (WTMODE) (BITTIME) [(CLK0) (BCNTA–) +(DTA0) (BCNTA)]
= (WTMODE) (TCNTCY) [(CLK0) (BCNTA–) + (DTA0) (BCNTA)]

WTMDD
WTMDD = (WTMODE) (BCNT = 15–) (INDSYN–) + (DISKWT) [(A13) (BCNT = 15) + (A13–) (INDSYN)]

9

SECTION V

SOFTWARE

The software design of a microprocessor system is as important as its hardware design. In this system, several functions which are normally performed by hardware are instead done in software in order to reduce device count. Examples of hardware/software tradeoffs include timing, transmit/receive, and CRC calculation.

5.1 SOFTWARE INTERFACE SUMMARY

The memory map in Figure 31 shows the memory address assignments for program memory, storage memory and the floppy-disk interface.

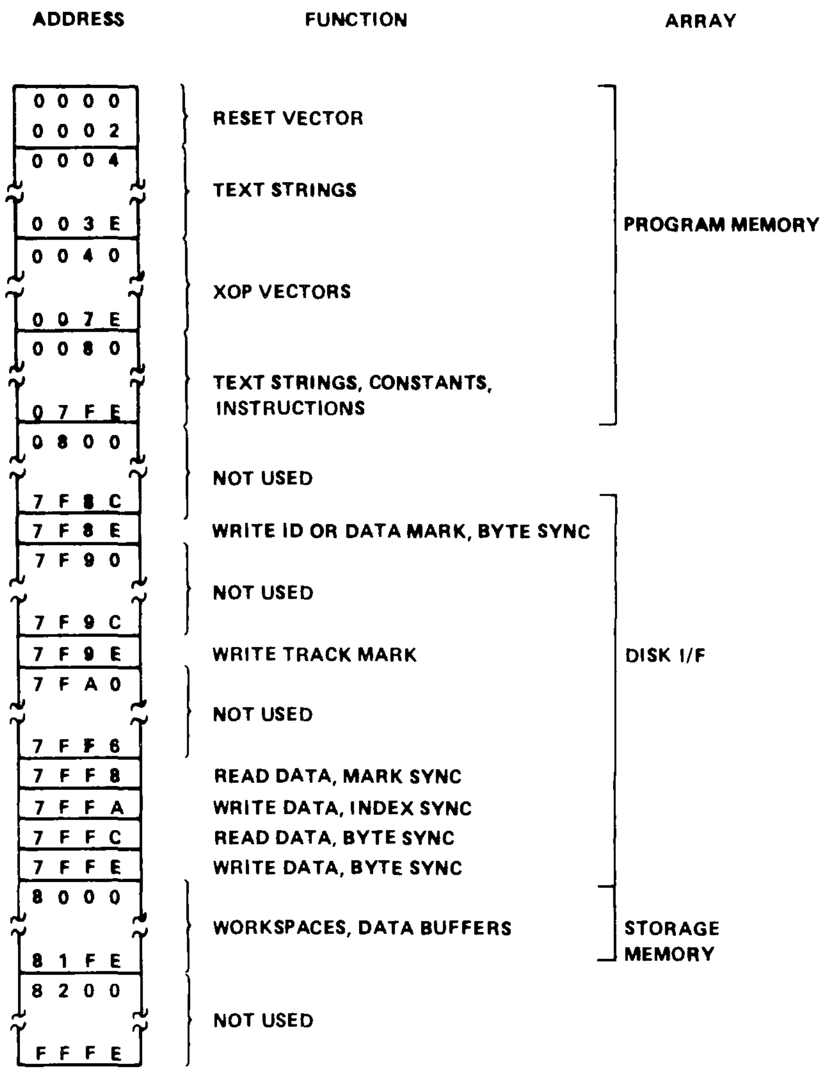
The CRU bit address assignments are summarized in Table 8 below.

Table 8. CRU Address Assignments

Bit Address	Output	Input
0	XMTOUT	RCVIN
1	RTS–	
2		
3		
4	SEL	INDEX
5		
6	STEP	TRK00
7	STEPUP	RDY

5.2 CONTROL SOFTWARE

Rather than providing individual examples of each individual control and data transfer function, all of the functions are combined to demonstrate complete system operation. The control software is modular, and the various subroutines may easily be adapted to different configurations of a TMS 9900 floppy-disk controller.



A0001309

Figure 31. Memory Address Assignments

5.2.1 Floppy-Disk Control Program

This program contains the complete software for interfacing the TMS 9900 floppy-disk controller to both the RS-232 terminal and the floppy-disk drive.

5.2.2 Operator Commands

The commands listed in Table 9 are available to the terminal operator. These commands enable the user to write and read data to and from the diskette, format tracks, display and enter data from memory, and execute from a selected address. The user is able to load and execute diagnostics in addition to performing normal data transfer operations. When errors are encountered, error information is reported at the terminal.

Table 9. Operator Commands

?WA	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?WH	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?WD	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?RA	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?RH	TRACK = ct st,	SECTOR = cs ss,	NUMBER = sn
?FM	TRACK = ct st	END TRACK = st et	
?MD	sadd eadd		
?ME	sadd		
?MX	sadd		

Underscored characters are entered by the user. All others are supplied by the controller. The lower case fields are hexadecimal values. If the users enters a blank into these fields, the default value is used by the controller. Entry of any non-printable character (e.g., Carriage Return, ESCape) during command entry causes the command to be aborted. Entry of a non-hexadecimal value in hexadecimal fields causes the command to be aborted.

Table 10 lists the command entry parameters and Table 11 gives a summary of the commands.

Table 10. Command Entry Parameters

Parameter	Definition	Default Value	Range
ct	Current track number	–	$00 \leqslant ct \leqslant 4C\ (76_{10})$
st	Starting track number	ct	$00 \leqslant st \leqslant 4C$
cs	Current sector number	–	$01 \leqslant cs \leqslant 1A\ (26_{10})$
ss	Starting sector number	cs	$01 \leqslant ss \leqslant 1A$
sn	Number of sectors	01	$01 \leqslant sn \leqslant FF(255_{10})$
et	Ending track number	st	$st \leqslant et \leqslant 4C$
sadd	Starting address	8000	$0 \leqslant sadd \leqslant FFFF$
eadd	Ending address	sadd	$0 \leqslant eadd \leqslant FFFF$

Table 11. Command Summary

Command	Description
WA	Write ASCII. The ASCII character strings entered by the user are written sequentially onto the diskette. Each sector may be terminated, filling remaining bytes with 00, by entry of any non-printable character (ASCII code < 20_{16}) other than ESCape. Entry of ESCape aborts the command.
WH	Write Hexadecimal. Hexadecimal bytes entered by the user are written sequentially onto the diskette. Sector termination and abort are performed in the same way as for the WA command.
WD	Write Deleted Data. Same as WH command, except the Deleted Data Mark (Clock = $C7_{16}$ Data = $F8_{16}$) rather than the Data Mark (Clock = $C7_{16}$, Data = FB_{16}) is written at the beginning of the Data Field.
RA	Read ASCII. The specified sectors are read and printed out as ASCII character strings. Each sector is printed beginning at a new line, and printing continues until the end of the sector, or until a non-printable ASCII character is encountered. When more than 80 characters are printed, the controller prints the eighty-first character in the first position of the next line. The command may be aborted at the end of any sector by depressing the BREAK key before the last character of the sector is printed. If a Deleted Data field is encountered, it is reported, and normal operation continues.
RH	Read Hexadecimal. The specified sectors are read and printed out as hexadecimal bytes, 16 bytes per line. The command may be aborted by depressing the BREAK key before the last character of any line is printed. If a Deleted Data field is encountered, it is reported and normal operation continues.
FM	Format Track. The specified tracks are completely rewritten with gaps, Track Marks, ID fields, and Data fields. All zero data is written into the 128 bytes of the data field.
MD	Memory Display. The contents of the specified memory addresses are printed out in hexadecimal byte format. The address of the first word of each line is printed, followed by 16 bytes. The command may be aborted by depressing the BREAK key before the last character of any line is printed.
ME	Memory Enter. Beginning with the selected location, the memory address and contents are printed. If it is to be modified, the user enters a hexadecimal byte value which will be stored at that address. If the value is not to be changed, the user enters a blank character (SPACE bar). The address is then incremented and the process is repeated until a non-hex character is entered, terminating the command.
MX	The CPU begins execution at the selected memory location.

Figure 32 shows the control software for the system described in this application report.

SECTION VI

SUMMARY

This application report has provided a thorough discussion of the TMS 9900 floppy-disk controller hardware and software system design. The economy of the CRU and the high throughput capability of the memory bus result in an economical, powerful system. The memory-to-memory architecture of the TMS 9900, along with its powerful instruction set and addressing capability, make the TMS 9900 ideally suited for applications where large amounts of data manipulation are necessary. Also, software development time is optimized by the minimization of lines of code resulting from the memory-to-memory instructions and large number of working registers.

It is likely that the designer using this application report will have requirements that are not addressed in this design. Variations in the sector length are accommodated with slight software modification. Higher density recording formats such as MFM and M^2FM require changes in the bit detector and data-separation logic. Higher throughput can be achieved by using an LSI terminal interface such as the TMS 9902 asynchronous communication controller and hardware CRC generation. Controlling multiple disks requires only the addition of drive select control lines. In short, variations on this design are easily implemented through slight hardware and software modifications.

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0001

                    IDT   'FDCTRL'
              ******************************************************************
              *
              *          FLOPPY DISK CONTROL PROGRAM
              *
              *          DECEMBER 21, 1976
              *
              *      THIS PROGRAM CONTAINS THE CONTROL SOFTWARE FOR THE
              *      SYSTEM DESCRIBED IN THE "TMS 9900 FLOPPY DISK
              *      CONTROL SYSTEM" APPLICATION REPORT.  THE PROGRAM
              *      ALLOWS THE USER TO READ, WRITE, AND FORMAT DATA ON
              *      FLOPPY DISK.  ADDITIONALLY, THE USER MAY ENTER,
              *      DISPLAY, AND INITIATE EXECUTION FROM
              *      ANY LOCATION IN MEMORY.  IT IS ASSUMED THAT THE
              *      CONSOLE USED FOR COMMAND ENTRY AND DATA DISPLAY IS
              *      A 300 BAUD, RS-232C TYPE TERMINAL.  THE COMMANDS
              *      USED IN INTERFACING THE TERMINAL OPERATOR
              *      INTERFACE TO THE CONTROLLER ARE FULLY
              *      DESCRIBED IN SECTION 5.3 OF THE "TMS 9900
              *      FLOPPY DISK CONTROL SYSTEM" APPLICATION
              *      REPORT.
              *
              ******************************************************************
              *
              *      DISK TRANSFER ADDRESSES
              *
        7F8E  MRKWT  EQU  >7F8E                DATA MARK WRITE
        7F9E  TKMWT  EQU  >7F9E                TRACK MARK WRITE
        7FFA  INDSWT EQU  >7FFA                INDEX SYNC WRITE
        7FFE  DTAWT  EQU  >7FFE                BYTE SYNC WRITE
        7FF8  MRKRD  EQU  >7FF8                MARK SYNC READ
        7FFC  DTARD  EQU  >7FFC                BYTE SYNC READ
              ******************************************************************
              *
              *      RAM EQUATES
              *
        80C0  SECBUF EQU  >80C0                CRC BUFFER FOR FORMATTING
        80F7  IDFLD  EQU  >80F7                ID FIELD IMAGE
        80F8  TKNUM  EQU  >80F8                TRACK NUMBER
        80FA  SECNUM EQU  >80FA                SECTOR NUMBER
        80FC  IDCRC  EQU  >80FC                CRC FOR ID FIELD
        80FF  DTAFLD EQU  >80FF                DATA FIELD IMAGE
        8100  DTABUF EQU  >8100                128 BYTE DATA BUFFER
        8180  DTACRC EQU  >8180                CRC FOR DATA FIELD
        8170  FDWP5  EQU  >8170                WORKSPACE 5
        8190  FDWP4  EQU  >8190                WORKSPACE 4
        81A0  FDWP3  EQU  >81A0                WORKSPACE 3
        81C0  FDWP2  EQU  >81C0                WORKSPACE 2
        81E0  FDWP1  EQU  >81E0                WORKSPACE 1
```

Figure 32. Floppy Disk Control Program (Sheet 1 of 28)

```
FLOPPY DISK CONTROL PROGRAM                              PAGE 0002

                 ************************************************************
                 *
                 *        CRU EQUATES
                 *
          0000   RIN      EQU   0                 RECEIVE IN
          0004   INDEX    EQU   4                 INDEX PULSE
          0006   TRK00    EQU   6                 TRACK 00 INDICATOR
          0007   RDY      EQU   7                 DRIVE READY
          0000   XOUT     EQU   0                 TRANSMIT OUT
          0001   RTS      EQU   1                 REQUEST TO SEND
          0004   SEL      EQU   4                 DRIVE SELECT
          0006   STEP     EQU   6                 HEAD STEP CONTROL
          0007   STEPUP   EQU   7                 STEP DIRECTION CONTROL
                 ************************************************************
                 *
                 *        XOP  EQUATES
                 *
                          DXOP  ERRT,1            ERROR REPORT
                          DXOP  IDRD,2            READ ID FIELD
                          DXOP  TKST,3            SET TRACK
                          DXOP  SINC,4            INCREMENT SECTOR
                          DXOP  DSON,5            SELECT DRIVE ON
                          DXOP  AXMT,6            ASCII DATA TRANSMIT
                          DXOP  CRCI,7            ID FIELD CRC CALCULATION
                          DXOP  CRCD,8            DATA FIELD CRC CALCULATION
                          DXOP  TINC,9            INCREMENT TRACK
                          DXOP  HRC2,10           RECEIVE HEX BYTE
                          DXOP  HXM2,11           TRANSMIT HEX BYTE
                          DXOP  NLIN,12           NEW LINE
                          DXOP  RECV,13           RECEIVE CHARACTER
                          DXOP  XMIT,14           TRANSMIT CHARACTER
                          DXOP  DLAY,15           SOFTWARE TIME DELAY
                 ************************************************************
                 *
                 *        TIME CONSTANTS
                 *
          00FA   HBDLY    EQU   250               HALF BIT (1.667 MS.)
          01F4   FBDLY    EQU   500               FULL BIT (3.333 MS.)
          03E8   B2DLY    EQU   1000              2 BITS (6.667 MS.)
          7530   B30DLY   EQU   30000             CARRIAGE RETURN
                 *                                (200 MS.)
          05DC   HSDLY    EQU   1500              HEAD STEP (10 MS.)
          1482   HDLDLY   EQU   5250              HEAD LOAD (35 MS.)
                 ************************************************************
                 *
                 *        POWER ON RESET VECTOR
                 *
    0000  81E0   RSETVC   DATA  FDWP1,START
    0002  ----
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 2 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                 PAGE 0003

                   ***************************************************************
                   *
                   *       ERROR MESSAGES
                   *
     0004     49   NIDMSG TEXT 'ID NOT FOUND'
     0010     00          BYTE 0
     0011     44   NDMMSG TEXT 'DATA MARK NOT FOUND'
     0024     00          BYTE 0
     0025     44   NRDYMS TEXT 'DRIVE NOT READY'
     0034     00          BYTE 0
     0035     43   CRCMSG TEXT 'CRCC ERROR'
     003F     00          BYTE 0
                   ***************************************************************
                   *
                   *       XOP VECTORS
                   *
     0040   FFFF          DATA -1,-1
     0042   FFFF
     0044   81E0          DATA FDWP1,ERRTPC
     0046   ----
     0048   81C0          DATA FDWP2,IDRDPC
     004A   ----
     004C   8180          DATA FDWP4,TKSTPC
     004E   ----
     0050   81C0          DATA FDWP2,SINCPC
     0052   ----
     0054   8180          DATA FDWP4,DSONPC
     0056   ----
     0058   81C0          DATA FDWP2,AXMTPC
     005A   ----
     005C   81A0          DATA FDWP3,CRCIPC
     005E   ----
     0060   81A0          DATA FDWP3,CRCOPC
     0062   ----
     0064   81A0          DATA FDWP3,TINCPC
     0066   ----
     0068   81A0          DATA FDWP3,HPC2PC
     006A   ----
     006C   81A0          DATA FDWP3,HXM2PC
     006E   ----
     0070   81A0          DATA FDWP3,NLINPC
     0072   ----
     0074   8180          DATA FDWP4,RECVPC
     0076   ----
     0078   8190          DATA FDWP4,XMITPC
     007A   ----
     007C   8170          DATA FDWP5,DLAYPC
     007E   ----
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 3 of 28)

```
FLOPPY DISK CONTROL PROGRAM                          PAGE 0004

                    ************************************************************
                    *
                    *       ASCII VALUES
                    *
     0080      41   ASCIIA TEXT 'A'
     0081      46   ASCIIF TEXT 'F'
     0082      4D   ASCIIM TEXT 'M'
     0083      1B   ESC    BYTE >1B
     0084      20   BLANK  BYTE >20
     0085      3F   QUEST  BYTE >3F
     0086      07   BELL   BYTE >07
     0087      08   BACKSP BYTE >08
     0088      0D   CARRET BYTE >0D
     0089      0A   LINEFD BYTE >0A
                    ************************************************************
                    *
                    *       ADDITIONAL TEXT MESSAGES
                    *
     008A      20   TKMSG  TEXT '    TRACK = '
     0093      00          BYTE 0
     0094      20   ENDMSG TEXT '    END'
     0099      00          BYTE 0
     009A      2C   SCTMSG TEXT ', SECTOR = '
     00A4      00          BYTE 0
     00A5      2C   NUMMSG TEXT ', NUMBER = '
     00AF      00          BYTE 0
     00B0      20   ADDMSG TEXT '   ADDRESS = '
     00BB      00          BYTE 0
     00BC      44   DLDMSG TEXT 'DELETED DATA FIELD'
     00CE      00          BYTE 0
                    ************************************************************
                    *
                    *       DISK MARK CONSTANTS
                    *
     00CF      F8   DLDMRK BYTE >F8
     00D0      FE   IDMRK  BYTE >FE
     00D1      FB   DTMRK  BYTE >FB
     00D2      FC   TKMRK  BYTE >FC
                           EVEN
                    ************************************************************
                    *
                    *       SUBROUTINE:  DLAY
                    *
                    *       CALLING SEQUENCE:  DLAY @COUNT
                    *
                    *       A SOFTWARE LOOP WILL BE EXECUTED THE NUMBER
                    *       OF TIMES SPECIFIED BY THE CALLING PROGRAM.
                    *       EACH ITERATION OF THE LOOP RESULTS IN A
                    *       DELAY OF 6.67 MICROSECONDS.
                    *
     00D4    060B   DLAYRC DEC  R11                 DECREMENT COUNT
     007E**00D4'
     00D6    16FE          JNE  DLAYRC              LOOP IF NOT 0
     00D8    0380          RTWP                     RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 4 of 28)

```
FLOPPY DISK CONTROL PROGRAM                              PAGE 0005

                   **************************************************************
                   *
                   *      SUBROUTINE:  RECV
                   *
                   *      CALLING SEQUENCE:  RECV @LOCATN
                   *
                   *      AN ASCII CHARACTER WITH CORRECT FORMATTING
                   *      IS RECEIVED AND THEN RETRANSMITTED
                   *      AT 300 BAUD.  THE RECEIVED CHARACTER IS STORED
                   *      AT THE SPECIFIED LOCATION.
                   *
      00DA  04CC   RECVRC CLR  R12                  SET CRU BASE
      0076**00DA'
      00DC  1F00   RCV    TB   RIN                  TEST RECEIVE INPUT
      00DE  13FE          JEQ  RCV                  LOOP UNTIL RIN = 0
      00E0  2FE0          DLAY @HBDLY               DELAY HALF BIT TIME
      00E2  00FA
      00E4  1F00          TB   RIN                  TEST RECEIVE INPUT
      00E6  13FA          JEQ  RCV                  IF RIN = 0, VALID START BIT
      00E8  020A          LI   R10, 3F              INITIALIZE ACCUMULATOR
      00EA  003F
      00EC  2FE0   RCVLP  DLAY @FBDLY               DELAY FULL BIT TIME
      00EE  01F4
      00F0  1F00          TB   RIN                  TEST RECEIVE INPUT
      00F2  16--          JNE  RCVOFF               SET MSB OF ACCUMULATOR
      00F4  026A          ORI  R10,>8000            IF RIN = 1
      00F6  8000
      00F8  091A   RCVOFF SRL  R10,1                SHIFT ACCUMULATOR
      00F2**1602
      00FA  18F8          JOC  RCVLP                IF CARRY, RECEIVE NEXT BIT
      00FC  2FE0          DLAY @B2DLY               DELAY 2 BIT TIMES
      00FE  03E8
      0100  1F00          TB   RIN                  TEST RECEIVE INPUT
      0102  16EC          JNE  RCV                  IF RIN = 0, FRAMING ERROR
      0104  D6CA          MOVB R10,*R11             MOVE RECEIVED CHARACTER
                   *                                TO SPECIFIED LOCATION AND
                   *                                RETRANSMIT.
```

Figure 32. Floppy Disk Control Program (Sheet 5 of 28)

```
FLOPPY DISK CONTROL PROGRAM                             PAGE 0006

                    ************************************************************
                    *
                    *       SUBROUTINE:  XMIT
                    *
                    *       CALLING SEQUENCE:  XMIT @LOCATN
                    *
                    *       AN ASCII CHARACTER WITH CORRECT FORMATTING
                    *       AND EVEN PARITY IS TRANSMITTED AT 300 BAUD.
                    *       THE LOCATION OF THE CHARACTER TO BE TRANS-
                    *       MITTED IS SPECIFIED AS THE CALLING PARAMETER.
                    *
0106   04CC  XMITPC CLR  R12                  INITIALIZE CRU BASE
007A**0106
0108   020A         LI   R10,3                INITIALIZE ACCUMULATOR
010A   0003
010C   0209         LI   R9,>8000             INITIALIZE PARITY MASK
010E   8000
0110   D29B         MOVB *R11,R10             FETCH CHARACTER
0112   1C--         JOP  PARADJ               IF ODD PARITY INVERT MSB
0114   04C9         CLR  R9                   ELSE, CLEAR PARITY MASK
0116   2A89  PARADJ XOR  R9,R10               XOR PARITY MASK
0112**1C01
                    *                         WITH CHARACTER
0118   1E01         SBZ  RTS                  TURN ON RTS
011A   0B8A         SRC  R10,8                ROTATE CHARACTER
011C   18--  XMTLP1 JOC  XOUTON               TEST TRANSMIT BIT
011E   1E00         SBZ  XOUT                 IF 0, RESET XOUT
0120   10--         JMP  XFBDLY               AND SKIP
0122   1D00  XOUTON SBO  XOUT                 ELSE, SET XOUT
011C**1802
0124   2FE0  XFBDLY DLAY @FBDLY               DELAY FULL BIT TIME
0126   01F4
0120**1001
0128   091A         SRL  R10,1                SHIFT ACCUMULATOR 1 BIT
012A   16F8         JNE  XMTLP1               IF NOT ZERO, TRANSMIT NEXT BIT
012C   1D01         SBO  RTS                  TURN OFF RTS
012E   0380         RTWP                      RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 6 of 28)

```
FLOPPY DISK CONTROL PROGRAM                         PAGE 0007

                  ****************************************************************
                  *
                  *       SUBROUTINE:   DSON
                  *
                  *       CALLING SEQUENCE:   DSON 0
                  *
                  *       THE FLOPPY DISK DRIVE IS SELECTED AND
                  *       THE SELECT DELAY PERIOD IS EXECUTED.  IF THE
                  *       DEVICE IS NOT READY, AN ERROR MESSAGE IS
                  *       PRINTED AND THE OPERATION IS ABORTED.
                  *       OTHERWISE, CONTROL RETURNS TO THE CALLING
                  *       PROGRAM.
                  *
 0130   04CC      DSONPC CLR  R12                 INITIALIZE CRU BASE
 0056**0130'
 0132   1D04             SBO  SEL                 SELECT DRIVE
 0134   2FE0             DLAY @HDLDLY             DELAY FOR HEAD LOAD
 0136   1482
 0138   1F07             TB   RDY                 TEST DRIVE STATUS
 013A   13--             JEQ  DSONRT              IF READY, NORMAL RETURN
 013C   2C60             ERRT @NRDYMS             ELSE, ABORT AND PRINT
 013E   0025'
                  *                               ERROR MESSAGE.
 0140   0380      DSONRT RTWP                     NORMAL RETURN
 013A**1302
                  ****************************************************************
                  *
                  *       SUBROUTINE:   HRC2
                  *
                  *       CALLING SEQUENCE:   HRC2 @LOCATN
                  *
                  *       A BLANK IS TRANSMITTED AND 2 CHARACTERS
                  *       ARE RECEIVED.  IF EITHER CHARACTER IS A
                  *       BLANK, NO OPERATION IS PERFORMED AND THE
                  *       NORMAL RETURN IS EXECUTED.  IF TWO HEXADEC-
                  *       IMAL VALUES ARE ENTERED, THE HEXADECIMAL
                  *       BYTE IS STORED AT THE LOCATION SPECIFIED
                  *       AS THE CALLING PARAMETER.  IF EITHER CHARACTER
                  *       IS AN ESCAPE, CONTROL IS RETURNED TO THE MAIN
                  *       PROGRAM AT THE POINT WHERE OPERATOR COMMANDS
                  *       ARE REQUESTED.  IF ANY OTHER CHARACTER IS
                  *       RECEIVED, NO OPERATION IS PERFORMED AND THE
                  *       RETURN PC VALUE WILL BE THE CONTENTS OF REG-
                  *       ISTER 10 OF THE CALLING PROGRAM.
                  *
 0142   81E0      RTRNVC DATA FDWP1,TOP           RETURN VECTOR
 0144   ----
 0146   2FA0      HRC2PC XMIT @BLANK              TRANSMIT BLANK
 0148   0094
 006A**0146
 014A   04CA             CLR  R10                 CLEAR HEX ACCUMULATOR
 014C   0708             SETO R8                  INITIALIZE CHARACTER COUNTER
 014E   2F49      HRC2LP RECV R9                  FETCH CHARACTER
 0150   9809             CB   R9,@ESC             COMPARE TO ESCAPE
```

9◀

Figure 32. Floppy Disk Control Program (Sheet 7 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                  PAGE 0008

0152  0083'
0154  16--              JNE   NOTEIC           IF NOT,CONTINUE
0156  0420              BLWP  @RTRNVC          ELSE, ABORT COMMAND
0158  0142
015A  9809      NOTEIC  CB    R9,@BLANK        COMPARE TO BLANK
015C  0084
0154**1602
015E  13--              JEQ   HRC2RT           IF = BLANK, RETURN
                *                              ELSE, CONVERT TO HEXADECIMAL
0160  0229              AI    R9,->3000        SUBTRACT ASCII BIAS
0162  D000
0164  11--              JLT   HRC2AB           IF LESS THAN >30, ABORT
0166  0289              CI    R9,>A00          TEST FOR NUMERIC
0168  0A00
016A  11--              JLT   NOHAJ            IF NUMERIC, SKIP
016C  0229              AI    R9,->700         ELSE, SUBTRACT ALPHA BIAS
016E  F900
0170  0289              CI    R9,>A00          IF LESS THAN >41, ABORT
0172  0A00
0174  11--              JLT   HRC2AB
0176  0289              CI    R9,>FFF          COMPARE TO ASCII F
0178  0FFF
017A  15--              JGT   HRC2AB           IF GREATER THAN, ABORT
017C  F289      NOHAJ   SOCB  R9,R10           STORE HEX VALUE IN
016A**1108
                *                              ACCUMULATOR
017E  0588              INC   R8               INCREMENT CHARACTER COUNT
0180  16--              JNE   HRC2ND           IF NOT 0, SKIP
0182  0A4A              SLA   R10,4            SHIFT HEX ACCUMULATOR
0184  10E4              JMP   HRC2LP           FETCH SECOND CHARACTER
0186  D6CA      HRC2ND  MOVB  R10,*R11         STORE HEX VALUE
0180**1602
                *                              AT SPECIFIED LOCATION
0188  0380      HRC2RT  RTWP                   RETURN
015E**1314
018A  C3AD      HRC2AB  MOV   @20(R13),R14     MODIFY RETURN PC
018C  0014
0164**1112
0174**110A
017A**1507
018E  10FC              JMP   HRC2RT           RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 8 of 28)

```
FLOPPY DISK CONTROL PROGRAM                         PAGE 0009

                  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                  ◆
                  ◆      SUBROUTINE:   HXM2
                  ◆
                  ◆      CALLING SEQUENCE:  HXM2 @LOCATN
                  ◆
                  ◆      THE HEXADECIMAL EQUIVALENT OF THE VALUE CONTAINED
                  ◆      IN THE LOCATION SPECIFIED BY THE PARAMETER IS
                  ◆      TRANSMITTED, PRECEDED BY A BLANK
                  ◆
0190  2FA0  HXM2PC XMIT @BLANK             TRANSMIT BLANK
0192  0084'
006E◆◆0190
0194  D29B         MOVB ◆R11,R10           FETCH BYTE
0196  06A0         BL   @HEXXMT            TRANSMIT FIRST CHARACTER
0198  ----
019A  0A4A         SLA  R10,4              SHIFT BYTE
019C  06A0         BL   @HEXXMT            TRANSMIT SECOND CHARACTER
019E  ----
01A0  0380         RTWP                    RETURN
01A2  C24A  HEXXMT MOV  R10,R9             MOVE CHARACTER
0198◆◆01A2
019E◆◆01A2'
01A4  0949         SRL  R9,4               SHIFT RIGHT 4 BITS
01A6  0289         CI   R9,>A00            TEST FOR NUMERIC
01A8  0A00
01AA  11--         JLT  NHADJ              IF SO, SKIP
01AC  0229         AI   R9,>700            ELSE,ADD ALPHA BIAS
01AE  0700
01B0  0229  NHADJ  AI   R9,>3000           ADD ASCII BIAS
01B2  3000
01AA◆◆1102
01B4  2F89         XMIT R9                 TRANSMIT HEX ASCII CHARACTER
01B6  045B         RT                      RETURN
                  ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                  ◆
                  ◆      SUBROUTINE:   NLIN
                  ◆
                  ◆      CALLING SEQUENCE:  NLIN 0
                  ◆
                  ◆      THE PRINTER IS ADVANCED TO THE BEGINNING
                  ◆      OF THE NEXT LINE
                  ◆
01B8  2FA0  NLINPC XMIT @CRRET             CARRIAGE RETURN
01BA  0088'
0072◆◆01B8
01BC  2FE0         DLAY @CRDDLY            CARRIAGE RETURN DELAY
01BE  7530
01C0  2FA0         XMIT @LINEFD            LINE FEED
01C2  0089'
01C4  0380         RTWP                    RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 9 of 28)

9◀

```
FLOPPY DISK CONTROL PROGRAM                                PAGE 0010

                   ************************************************************
                   *
                   *       SUBROUTINE:  IDRD
                   *
                   *       CALLING SEQUENCE:  IDRD 0
                   *
                   *       EACH ID FIELD OF THE CURRENT DISKETTE TRACK
                   *       IS READ UNTIL THE ID FIELD WITH THE CORRECT
                   *       TRACK, SECTOR, AND CRC IS FOUND, AT WHICH
                   *       TIME THE ROUTINE IS EXITED.  IF THE CORRECT
                   *       FIELD IS NOT FOUND WITHIN A COMPLETE DISK
                   *       REVOLUTION (IE BEFORE 2 INDEX PULSES ARE
                   *       DETECTED), THE OPERATION IS ABORTED AND AN
                   *       ERROR MESSAGE IS REPORTED.
                   *
 01C6  2DC0   IDRDRC CRCI 0                    UPDATE ID FIELD IMAGE CRC
 004A**01C6'
 01C8  2D40          DSON 0                    TURN ON DRIVE
 01CA  0209          LI    R9,2                INITIALIZE INDEX PULSE COUNT
 01CC  0002
 01CE  020A   IDMRD  LI R10,IDFLD              SET POINTER TO ID FIELD
 01D0  90F7
 01D2  9EA0          CB    @MRKRD,*R10+        COMPARE DISK BYTE TO
 01D4  7FF8
                   *                           MARK CHARACTER
 01D6  13--          JEQ   MRKFND              IF MARK, CONTINUE
 01D8  1F04          TB    INDEX               ELSE, TEST FOR INDEX SIGNAL
 01DA  16F9          JNE   IDMRD               IF NO INDEX, REREAD DISK
 01DC  0609          DEC   R9                  IF INDEX,DECREMENT INDEX COUNT
 01DE  16F7          JNE   IDMRD               IF NOT 0, REREAD DISK
 01E0  2C60          ERRT  @NIDMSG             ELSE, REPORT ID READ ERROR
 01E2  0004'
 01E4  0209   MRKFND LI    R9,6                LOAD BYTE COUNT
 01E6  0006
 01D6**1306
 01E8  9B3A   IDRDLP CB    *R10+,@DTARD        COMPARE DISK DATA
 01EA  7FFC
                   *                           TO ID FIELD IMAGE
 01EC  16F0          JNE   IDMRD               IF NOT EQUAL, START OVER
 01EE  0609          DEC R9                    DECREMENT BYTE COUNT
 01F0  16FB          JNE IDRDLP                IF NOT 0, READ NEXT BYTE
 01F2  0380          RTWP                      ELSE, ID FOUND, RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 10 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                   PAGE 0011

                  ****************************************************************
                  *
                  *       SUBROUTINE:  ERRT
                  *
                  *       CALLING SEQUENCE:  ERRT @MESSAGE
                  *
                  *       THE MESSAGE WHOSE ADDRESS IS CONTAINED IN R11
                  *       WHEN THE ROUTINE IS ENTERED IS PRINTED,
                  *       FOLLOWED BY THE CURRENT TRACK AND SECTOR
                  *       NUMBER.  THE DRIVE IS TURNED OFF AND CONTROL
                  *       IS RETURNED TO THE COMMAND ENTRY PROGRAM.
                  *
      01F4  2F00  ERRTRC NLIN 0                      NEW LINE
      0046**01F4
      01F6  2D9B         AXMT *R11                   PRINT SELECTED MESSAGE
      01F8  2DA0         AXMT @TKMSG                 PRINT TRACK MESSAGE
      01FA  008A'
      01FC  2EE0         HXM2 @TKNUM                 PRINT TRACK NUMBER
      01FE  80F8
      0200  2DA0         AXMT @SCTMSG                PRINT SECTOR MESSAGE
      0202  009A'
      0204  2EE0         HXM2 @SECNUM                PRINT SECTOR NUMBER
      0206  80FA
      0208  1E04         SBZ  SEL                    TURN OFF DISK DRIVE
      020A  0420         BLWP @RTRNVC                RETURN TO COMMAND
      020C  0142'
                  *                                  ENTRY PROGRAM
```

9◄

Figure 32. Floppy Disk Control Program (Sheet 11 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                        PAGE 0012

                    ****************************************************************
                    *
                    *       SUBROUTINE:   AXMT
                    *
                    *       CALLING SEQUENCE:   AXMT @MESSAGE
                    *
                    *       THE ASCII CHARACTER STRING, THE
                    *       BEGINNING ADDRESS OF WHICH IS CONTAINED
                    *       IN R11, IS TRANSMITTED.  THE END OF THE
                    *       STRING IS INDICATED BY A NON-PRINTABLE
                    *       CHARACTER (IE LESS THAN HEX 20)
                    *
020E  020A  AXMTPC LI    R10,80               LOAD MAX CHARACTERS
0210  0050
005A**020E
                    *                         PER LINE
0212  D27B  AXMTLP MOVB  *R11+,R9             FETCH CHARACTER
0214  9809         CB    R9,@BLANK            PRINTABLE CHARACTER?
0216  0084'
0218  11--         JLT   AXMTRT               IF NOT, RETURN
021A  2F89         XMIT  R9                   ELSE, PRINT CHARACTER
021C  060A         DEC   R10                  DECREMENT MAX CHAR COUNT
021E  16F9         JNE   AXMTLP               IF NOT 0, FETCH NEXT CHAR
0220  981B         CB    *R11,@BLANK          ELSE, IS NEXT CHAR
0222  0084'
                    *                         PRINTABLE?
0224  11--         JLT   AXMTRT               IF NOT, RETURN
0226  2F00         NLIN  0                    NEW LINE
0228  10F4         JMP   AXMTLP               PRINT REST OF STRING
022A  0380  AXMTRT RTWP                       STRING PRINTED, RETURN
0218**1108
0224**1102
```

Figure 32. Floppy Disk Control Program (Sheet 12 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                PAGE 0013

                  ************************************************************
                  *
                  *      SUBROUTINE:  CRCI
                  *
                  *      CALLING SEQUENCE:  CRCI 0
                  *
                  *      THE CRC IS CALCULATED FOR THE ID FIELD IMAGE
                  *      CONTAINED IN MEMORY AND STORED IN THE LAST 2
                  *      BYTES OF THE FIELD.
                  *
     022C  020A   CRCIPC LI    R10,IDFLD         SET UP ID FIELD POINTER
     022E  80F7
     005E**022C
     0230  0209          LI    R9,5              SET UP ID FIELD COUNT
     0232  0005
     0234  06A0          BL    @CRCALC           CALCULATE CRC
     0236  ----
     0238  0380          RTWP                    RETURN
                  ************************************************************
                  *
                  *      SUBROUTINE:  CRCD
                  *
                  *      CALLING SEQUENCE:  CRCD 0
                  *
                  *      THE CRC IS CALCULATED FOR THE DATA FIELD IMAGE
                  *      CONTAINED IN MEMORY AND STORED IN THE LAST 2
                  *      BYTES OF THE FIELD.
                  *
     023A  020A   CRCDPC LI    R10,DTAFLD        SET UP DATA FIELD POINTER
     023C  80FF
     0062**023A
     023E  0209          LI    R9,129            SET UP DATA FIELD COUNT
     0240  0081
     0242  06A0          BL    @CRCALC           CALCULATE CRC
     0244  ----
     0246  0380          RTWP                    RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 13 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                     PAGE 0014

                     ***********************************************************
                     *
                     *      SUBROUTINE:  CRCALC
                     *
                     *      CALLING SEQUENCE:  LI    R10,FLDADD
                     *                         LI    R9,FLDCNT
                     *                         BL    @CRCALC
                     *
                     *      THE CYCLIC REDUNDANCY CHECK CHARACTER (CRC) FOR
                     *      THE FIELD ADDRESSED BY R10 IS CALCULATED
                     *      AND STORED IN THE LAST 2 BYTES OF THE
                     *      FIELD.  THE LENGTH OF THE FIELD (EXCLUDING CRC)
                     *      IS SPECIFIED BY R9.  THE CRC POLYNOMIAL IS
                     *      X**16+X**12+X**5+1.  BEFORE CRC CALCULATION
                     *      BEGINS, THE PARTIAL CRC IS PRESET TO ALL ONES.
                     *      R7, R8, R9, AND R10 ARE DESTROYED.
                     *
     0248   0708    CRCALC SETO R8                      PRESET PARTIAL CRC
     0236**0248'
     0244**0248
     024A   04C7    CRCLP  CLR  R7                      CLEAR SCRATCH REGISTER
     024C   D1FA           MOVB *R10+,R7                FETCH NEXT BYTE
     024E   2A07           XOR  R7,R8                   XOR NEW BYTE WITH CRC
     0250   C1C8           MOV  R8,R7                   MOVE TO SCRATCH REG
     0252   0947           SRL  R7,4                    SHIFT SCRATCH RIGHT 4
     0254   29C8           XOR  R8,R7                   XOR CRC WITH SCRATCH
     0256   0247           ANDI R7, FF00                MASK OFF LOWER BYTE
     0258   FF00
     025A   0947           SRL  R7,4                    SHIFT SCRATCH RIGHT 4
     025C   2A07           XOR  R7,R8                   XOR SCRATCH WITH CRC
     025E   0B77           SRC  R7,7                    ROTATE SCRATCH RIGHT 7
     0260   2A07           XOR  R7,R8                   XOR SCRATCH WITH CRC
     0262   06C8           SWPB R8                      REVERSE BYTES IN CRC
     0264   0609           DEC  R9                      DECREMENT BYTE COUNT
     0266   16F1           JNE  CRCLP                   IF NOT 0, FETCH NEXT BYTE
     0268   DE88           MOVB R8,*R10+                ELSE, TRANSFER
     026A   06C8           SWPB R8                      CRC TO THE END
     026C   D688           MOVB R8,*R10                 OF THE FIELD
     026E   045B           RT                           RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 14 of 28)

```
FLOPPY DISK CONTROL PROGRAM                         PAGE 0015

                      ************************************************************
                      *
                      *      SUBROUTINE:  SINC
                      *
                      *      CALLING SEQUENCE:  SINC 0
                      *
                      *      THE SECTOR NUMBER IS INCREMENTED BY 1.
                      *      IF THE NEW VALUE IS GREATER THAN 26,
                      *      THE SECTOR NUMBER IS SET TO 1 AND
                      *      THE TRACK NUMBER IS INCREMENTED,
                      *      AND THE HEAD IS STEPPED TO THE NEXT TRACK.
                      *
      0270   D2A0     SINCPC MOVB @SECNUM,R10       FETCH SECTOR NUMBER
      0272   80FA
      0052**0270'
      0274   022A            AI   R10,>100          ADD 1 TO SECTOR NUMBER
      0276   0100
      0278   028A            CI   R10,27*>100       COMPARE TO 27
      027A   1B00
      027C   14--            JHE  SECNXT            IF HIGH OR EQUAL,
                      *                             INCREMENT TRACK
      027E   D80A     SECXIT MOVB R10,@SECNUM       RESTORE SECTOR NUMBER
      0280   80FA
      0282   0380            RTWP                   RETURN
      0284   2E40     SECNXT TINC 0                 INCREMENT TRACK NUMBER
      027C**1403
      0286   020A            LI   R10,>100          LOAD NEW SECTOR NUMBER
      0288   0100
      028A   10F9            JMP  SECXIT            STORE SECTOR NUMBER
                      ************************************************************
                      *
                      *      SUBROUTINE:  TKST
                      *
                      *      CALLING SEQUENCE:  TKST @TRACK
                      *
                      *      THE READ/WRITE HEAD OF THE DISK DRIVE IS
                      *      STEPPED TO THE TRACK NUMBER SPECIFIED BY THE
                      *      LEFT BYTE OF R11, UNLESS THE DISK IS NOT
                      *      READY, IN WHICH CASE THE OPERATION IS ABORTED.
                      *      IF THE SPECIFIED TRACK IS OUT OF RANGE (IE
                      *      GREATER THAN 76, THE HEAD IS STEPPED TO TRACK
                      *      0.  THE NEW TRACK NUMBER REPLACES THE OLD
                      *      TRACK NUMBER IN MEMORY.  IF THE NEW TRACK
                      *      NUMBER IS TO BE 0, THE TRACK IS STEPPED
                      *      UNTIL THE TRK00 STATUS SIGNAL IS DETECTED.
                      *      IF THE OLD TRACK NUMBER WAS 0, THE HEAD IS
                      *      STEPPED TO TRACK 0 BEFORE THE NEW STEPPING
                      *      OPERATION BEGINS.
                      *
      028C   04CC     TKSTPC CLR  R12               INITIALIZE CRU BASE
      0046**028C'
      028E   1D04            SBO  SEL               SELECT DRIVE
      0290   2FE0            DLAY @HDLDLY           HEAD LOAD DELAY
      0292   1482
```

Figure 32. Floppy Disk Control Program (Sheet 15 of 28)

```
FLOPPY DISK CONTROL PROGRAM                          PAGE 0016

 0294  1F07          TB    RDY          CHECK DRIVE STATUS
 0296  13--          JEQ   TKCNTU       IF READY, CONTINUE
 0298  2C60          ERPT  @NRDYMS      ELSE, REPORT ERROR
 029A  0025'
 029C  C24B   TKCNTU MOV   R11,R9       SAVE NEW TRACK NUMBER
 0296◆◆1302
 029E  0989          SRL   R9,8         TO RIGHT BYTE OF R9
 02A0  13--          JEQ   TKT00        IF 0, CLEAR TRACK
 02A2  0289          CI    R9,76        NEW TRACK NUMBER IN RANGE?
 02A4  004C
 02A6  12--          JLE   TKNZRO       IF SO, SKIP
 02A8  04C9          CLR   R9           ELSE, CLEAR NEW TRACK NUMBER
 02AA  06A0   TKT00  BL    @TKCLR       STEP TO TRACK 00
 02AC  ----
 02A0◆◆1304
 02AE  10--          JMP   TKSTRT       RETURN
 02B0  D2A0   TKNZRO MOVB  @TKNUM,R10   FETCH OLD TRACK NUMBER
 02B2  80F8
 02A6◆◆1204
 02B4  098A          SRL   R10,8        MOVE TO RIGHT BYTE
 02B6  16--          JNE   TKNZR1       IF NOT 00, CONTINUE
 02B8  06A0          BL    @TKCLR       ELSE, STEP TO TRACK 00
 02BA  ----
 02BC  8289   TKNZR1 C     R9,R10       COMPARE NEW TRACK
 02B6◆◆1602
              ◆                         TO OLD TRACK NUMBER
 02BE  11--          JLT   STPOUT       IF LESS THAN, STEP OUT 1 TRACK
 02C0  13--          JEQ   TKSTRT       IF EQUAL, RETURN
 02C2  1D07          SBO   STEPUP       ELSE, STEP IN 1 TRACK
 02C4  058A          INC   R10          INCREMENT OLD TRACK
 02C6  10--          JMP   TKGO         STEP HEAD
 02C8  1E07   STPOUT SBZ   STEPUP       SELECT STEP OUT
 02BE◆◆1104
 02CA  060A          DEC   R10          DECREMENT OLD TRACK
 02CC  06A0   TKGO   BL    @TKSTEP      STEP HEAD
 02CE  ----
 02C6◆◆1002
 02D0  10F5          JMP   TKNZR1       REPEAT FOR NEXT STEP
 02D2  06C9   TKSTRT SWPB  R9           MOVE NEW TRACK NUMBER
 02AE◆◆1011
 02C0◆◆1308
              ◆                         TO LEFT BYTE
 02D4  D809          MOVB  R9,@TKNUM    UPDATE TRACK NUMBER
 02D6  80F8
 02D8  0380          RTWP               RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 16 of 28)

```
FLOPPY DISK CONTROL PROGRAM                          PAGE 0017

                   ***************************************************************
                   *
                   *      SUBROUTINE:  TKCLR
                   *
                   *      CALLING SEQUENCE:   BL    @TKCLR
                   *
                   *      THE READ/WRITE HEAD IS STEPPED OUT UNTIL
                   *      THE TRK00 STATUS SIGNAL BECOMES ACTIVE.
                   *      THE CONTENTS OF R8 AND R11 ARE DESTROYED.
                   *
 02DA   C20B       TKCLR   MOV   R11,R8               SAVE RETURN LINKAGE
 02AC**02DA'
 02BA**02DA'
 02DC   1F07       TKCLP   TB    RDY                  TEST DRIVE STATUS
 02DE   16--               JNE TKCABT                 IF NOT READY, ABORT
 02E0   1F06               TB    TRK00                TEST TRACK 00 STATUS SIGNAL
 02E2   16--               JNE   TKICNT               IF NOT ACTIVE,CONTINUE
 02E4   0458               B     *R8                  ELSE, RETURN
 02E6   1E07       TKICNT SBZ   STEPUP                SET TO STEP OUT
 02E2**1601
 02E8   06A0               BL    @TKSTEP              STEP HEAD
 02EA   ----
 02EC   10F7               JMP   TKCLP                CONTINUE LOOP
 02EE   04C8       TKCABT CLR   R8                    SET TRACK
 02DE**1607
 02F0   D808               MOVB  R8,@TKNUM            NUMBER TO 00
 02F2   80F8
 02F4   2C60               ERPT  @NRDYMS              REPORT ERROR AND ABORT
 02F6   0025'
                   ***************************************************************
                   *
                   *      SUBROUTINE:  TKSTEP
                   *
                   *      CALLING SEQUENCE:   BL    @TKSTEP
                   *
                   *      THE STEP PULSE IS GENERATED FOR 11.3
                   *      MICROSECONDS AND THE HEAD STEP DELAY
                   *      IS OBSERVED.
                   *
 02F8   1D06       TKSTEP SBO   STEP                  SET STEP SIGNAL
 02CE**02F8'
 02EA**02F8
 02FA   1000               NOP                        DUMMY DELAY
 02FC   1E06               SBZ   STEP                 RESET STEP SIGNAL
 02FE   2FE0               DLAY  @HSDLY               DELAY FOR HEAD STEP
 0300   05DC
 0302   045B               RT                         RETURN
```

9◄

Figure 32. Floppy Disk Control Program (Sheet 17 of 28)

```
FLOPPY DISK CONTROL PROGRAM                            PAGE 0018

                 ************************************************************
                 *
                 *      SUBROUTINE:  TINC
                 *
                 *      CALLING SEQUENCE:  TINC 0
                 *
                 *      THE HEAD IS MOVED TO THE NEXT CONSECUTIVE
                 *      POSITION.  IF ON THE INNERMOST TRACK (76),
                 *      THE HEAD IS MOVED TO TRACK 00.
                 *
0304  D2E0       TINCPC MOVB @TKNUM,R11          FETCH TRACK NUMBER
0306  80F8
0066**0304'
0308  022B              AI   R11,>100            ADD 1 TO TRACK NUMBER
030A  0100
030C  2CDB              TKST *R11                MOVE HEAD
030E  0380              RTWP                     RETURN
                 ************************************************************
                 *
                 *      COMMAND CHARACTER LIST
                 *
0310    57       CMDLST TEXT 'WAWHWDRARHFMMDMEMX'
                 *
                 *      COMMAND ENTRY POINT TABLE
                 *
0322  ----       CMDENT DATA WTASCI,WRTHEX,WRTDEL,RDASCI,RDHEX
0324  ----
0326  ----
0328  ----
032A  ----
032C  ----              DATA FORMAT,DUMP,ENTER,EXECUT
032E  ----
0330  ----
0332  ----
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 18 of 28)

```
FLOPPY DISK CONTROL PROGRAM                          PAGE 0019

                  ************************************************************
                  *
                  *      POWER-ON RESET ENTRY POINT
                  *
0334   04CC   START   CLR   R12               INITIALIZE CRU BASE
0002**0334'
0336   020B           LI    R11,>300          LOAD CRU INITIALIZATION VALUE
0338   0300
033A   320B           LDCR  R11,8             AND OUTPUT TO CRU
033C   020B           LI    R11,IDFLD         SET ID FIELD IMAGE POINTER
033E   80F7
0340   020A           LI    R10,>100          SET INITIAL SECTOR VALUE
0342   0100
0344   DEE0           MOVB  @IDMRK,*R11+      ID MARK DATA PATTERN TO
0346   00D0'
                  *                           FIRST BYTE OF ID FIELD IMAGE
0348   DECC           MOVB  R12,*R11+         0 TO SECOND BYTE
                  *                           (TRACK NUMBER)
034A   DECC           MOVB  R12,*R11+         0 TO THIRD BYTE
034C   DECA           MOVB  R10,*R11+         01 TO FOURTH BYTE
                  *                           (SECTOR NUMBER)
034E   D6CC           MOVB  R12,*R11          0 TO FIFTH BYTE
0350   2CDC           TEST  *R12              SET READ/WRITE HEAD TO TRACK 0
0352   1E04           SBZ   SEL               TURN OFF DRIVE
                  ************************************************************
                  *
                  *      OPERATOR COMMAND REQUEST ENTRY POINT
                  *
0354   2F00   TOP     NLIN  0                 NEW LINE
0144**0354'
0356   2FA0           XMIT  @QUEST            PRINT PROMPTING MESSAGE
0358   0085'
035A   2FA0           XMIT  @BELL             (QUESTION MARK, BELL)
035C   0086'
035E   2F4A           RECV  R10               READ FIRST CHARACTER
                  *                           OF COMMAND
0360   06CA           SWPB  R10               SAVE IN RIGHT BYTE
0362   2F4A           RECV  R10               READ SECOND CHARACTER
                  *                           OF COMMAND
0364   06CA           SWPB  R10               REVERSE CHARACTERS IN R10
0366   0208           LI    R8,CMDLST         SET COMMAND LIST POINTER
0368   0310'
036A   0209           LI    R9,CMDENT-2       SET COMMAND ENTRY POINTER
036C   0320'
036E   C1F8   CMDLP   MOV   *R8+,R7           FETCH COMMAND IN LIST
0370   13F1           JEQ   TOP               IF LIST VALUE = 0, NOT
                  *                           A LEGAL COMMAND
0372   05C9           INCT  R9                INCREMENT ENTRY POINTER
0374   81CA           C     R10,R7            COMPARE ENTERED COMMAND
                  *                           TO LIST
0376   16FB           JNE   CMDLP             IF NOT EQUAL, REPEAT
0378   C259           MOV   *R9,R9            ELSE, COMMAND FOUND,
                  *                           FETCH ENTRY POINT
037A   020A           LI    R10,TOP           COMMAND PROGRAM RETURN
```

Figure 32. Floppy Disk Control Program (Sheet 19 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                PAGE 0020

 037C  0354'
                 *                              ADDRESS
 037E  9807             CB    R7,@ASCIIM        TEST FOR MD,ME, OR
 0380  0092'
                 *                              MX COMMANDS
 0382  13--             JEQ   ADDFCH            IF SO, FETCH ADDRESS ENTRY
 0384  2DA0             AXMT  @TKMSG            PRINT TRACK MESSAGE
 0386  008A'
 0388  D220             MOVB  @TKNUM,R8         FETCH CURRENT TRACK
 038A  80F8
                 *                              NUMBER
 038C  2EC8             HXM2  R8                PRINT TRACK NUMBER
 038E  2E88             HRC2  R8                READ NEW TRACK NUMBER
 0390  0288             CI    R8,77*256         NEW TRACK NUMBER LEGAL?
 0392  4D00
 0394  14DF             JHE   TOP               IF NOT, ABORT
 0396  2CD8             TKST  *R8               STEP HEAD TO NEW TRACK
 0398  1E04             SBZ   SEL               TURN OFF DRIVE
 039A  9807             CB    R7,@ASCIIF        FORMAT COMMAND?
 039C  0081'
 039E  16--             JNE   SECFCH            IF NOT, CONTINUE
 03A0  0459             B     *R9               ELSE, EXECUTE COMMAND
 03A2  2DA0      SECFCH AXMT  @SCTMSG           PRINT SECTOR MESSAGE
 03A4  009A'
 039E**1601
 03A6  D1A0             MOVB  @SECNUM,R6        FETCH CURRENT SECTOR
 03A8  80FA
 03AA  2EC6             HXM2  R6                PRINT CURRENT SECTOR
 03AC  2E86             HRC2  R6                READ NEW SECTOR NUMBER
 03AE  0286             CI    R6,>100           LESS THAN 1
 03B0  0100
 03B2  11D0             JLT   TOP               IF SO, ABORT
 03B4  0286             CI    R6,27*256         GREATER THAN 26?
 03B6  1B00
 03B8  14CD             JHE   TOP               IF SO, ABORT
 03BA  D806             MOVB  R6,@SECNUM        UPDATE SECTOR NUMBER
 03BC  80FA
 03BE  2DA0             AXMT  @NUMMSG           PRINT NUMBER MESSAGE
 03C0  00A5'
 03C2  0205             LI    R5,>100           LOAD DEFAULT NUMBER
 03C4  0100
 03C6  2E85             HRC2  R5                READ NUMBER
 03C8  0985             SRL   R5,8              MOVE TO RIGHT BYTE
 03CA  13C4             JEQ   TOP               IF NUMBER = 0, ABORT
 03CC  0459             B     *R9               EXECUTE COMMAND
 03CE  0208      ADDFCH LI    R8,>8000          LOAD DEFAULT ADDRESS
 03D0  8000
 0382**1325
 03D2  2E88             HRC2  R8                READ FIRST BYTE OF ADDRESS
 03D4  06C8             SWPB  R8                SAVE IN RIGHT BYTE
 03D6  2FA0             XMIT  @BACKSP           BACKSPACE PRINTER
 03D8  0087'
 03DA  2E88             HRC2  R8                READ SECOND BYTE OF ADDRESS
 03DC  06C8             SWPB  R8                CORRECT ADDRESS BYTES
 03DE  0459             B     *R9               EXECUTE COMMAND
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 20 of 28)

```
FLOPPY DISK CONTROL PROGRAM                            PAGE 0021

                     ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
                     ◆
                     ◆      COMMAND CONTROL PROGRAM:  RDASCI,RDHEX
                     ◆
                     ◆      THESE COMMANDS ENABLE THE OPERATOR TO ACCESS
                     ◆      A SPECIFIED NUMBER OF SECTORS BEGINNING AT THE
                     ◆      CURRENT TRACK AND SECTOR LOCATION, PRINTING
                     ◆      THE CONTENTS OF EACH SECTOR IN EITHER ASCII (RA)
                     ◆      OR HEXADECIMAL FORMAT (RH).  IF A DELETED DATA
                     ◆      FIELD IS DETECTED, IT IS REPORTED AND READING
                     ◆      CONTINUES.  IF THE ID FIELD OR DATA MARK ARE
                     ◆      NOT FOUND, OR IF A CRC ERROR OCCURS, THE ERROR
                     ◆      IS REPORTED AND THE COMMAND IS ABORTED.
                     ◆
                     ◆      ENTRY PARAMETERS:      R10 = RETURN ADDRESS
                     ◆                              R7 = COMMAND CHARACTERS
                     ◆                              R5 = NUMBER OF SECTORS
                     ◆                                   TO READ
                     ◆
           03E0' RDASCI EQU   $                    RA COMMAND ENTRY POINT
     0328◆◆03E0'
           03E0' RDHEX  EQU   $                    RH COMMAND ENTRY POINT
     032A◆◆03E0'
     03E0  06C7         SWPB R7                    SWAP COMMAND CHAR BYTES
     03E2  0206  READ   LI    R6,DTAFLD            LOAD DATA FIELD IMAGE
     03E4  80FF
                 ◆                                 POINTER
     03E6  0208         LI    R8,DTARD             LOAD DISK DATA READ
     03E8  7FFC
                 ◆                                 ADDRESS
     03EA  2C80         IDRD 0                     READ ID FIELD
     03EC  DDA0         MOVB @MRKRD,◆R6+           READ DATA MARK
     03EE  7FF8
     03F0  0200         LI    R0,130               REPEAT NEXT INSTRUCTION
     03F2  0082
                 ◆                                 130 TIMES
     03F4  DD98  RDLPL1 MOVB ◆R8,◆R6+              MOVE DISK DATA TO
                 ◆                                 DATA FIELD IMAGE
     03F6  0600         DEC   R0
     03F8  16FD         JNE   RDLPL1
     03FA  1E04         SBZ   SEL                  TURN OFF DRIVE
     03FC  9820         CB    @DTAFLD,@DTMRK       NORMAL DATA MARK?
     03FE  80FF
     0400  00D1'
     0402  13--         JEQ   DMRKOK               IF SO, CONTINUE
     0404  9820         CB    @DTAFLD,@DLDMRK      DELETED DATA MARK?
     0406  80FF
     0408  00CF'
     040A  13--         JEQ   DDMXMT               IF SO, SKIP
     040C  2C60         ERPT @NDMMSG               PRINT ERROR MESSAGE
     040E  0011'
     0410  2F00  DDMXMT NLIN 0                     NEW LINE
     040A◆◆1302
     0412  2DA0         AXMT @DLDMSG               REPORT DELETED DATA MARK
```

9

Figure 32. Floppy Disk Control Program (Sheet 21 of 28)

```
FLOPPY DISK CONTROL PROGRAM                          PAGE 0022

0414   00BC'
0416   C220   DMPFDF MOV   @DTACRC,R8        FETCH READ CRC
0418   8180
0402◆◆1309
041A   2E00          CRCD  0                 RECALCULATE CRC
041C   8220          C     @DTACRC,R8        CRC CORRECT?
041E   8180
0420   13--          JEQ   RDPRT             IF SO CONTINUE
0422   2060          ERRT  @CRCMSG           ELSE, REPORT ERROR
0424   0035'
0426   2F00   RDPRT  NLIN  0                 NEW LINE
0420◆◆1302
0428   04E0          CLR   @DTACRC           CLEAR END OF DATA
042A   8180
              ◆                              FIELD IMAGE
042C   0206          LI    R6,DTABUF         LOAD FIELD IMAGE
042E   8100
              ◆                              POINTER
0430   9807          CB    R7,@ASCIIA        RA COMMAND?
0432   0080'
0434   13--          JEQ   ASCIRD            IF SO, PRINT IN ASCII
              ◆                              FORMAT
0436   0209          LI    R9,8              LOAD LINE COUNT
0438   0008
043A   0208   HXPTLP LI    R8,16             LOAD BYTE COUNT
043C   0010
043E   2F00          NLIN  0                 NEW LINE
0440   2ED6   HXPLP1 HXM2  *R6               PRINT DATA BYTE
0442   0586          INC   R6                INCREMENT DATA POINTER
0444   0608          DEC   R8                DECREMENT BYTE COUNT
0446   16FC          JNE   HXPLP1            IF NOT 0, PRINT NEXT BYTE
0448   1F00          TB    RIN               OPERATOR INTERRUPT?
044A   16--          JNE   READRT            IF SO, ABORT
044C   0609          DEC   R9                DECREMENT LINE COUNT
044E   16F5          JNE   HXPTLP            IF NOT 0, PRINT NEXT LINE
0450   10--          JMP   NXTSCT            CONTINUE
0452   2D96   ASCIRD AXMT  *R6               PRINT DATA FIELD
0434◆◆130E
              ◆                              IN ASCII
0454   2D00   NXTSCT SINC  0                 UPDATE SECTOR NUMBER
0450◆◆1001
0456   1F00          TB    RIN               OPERATOR INTERRUPT?
0458   16--          JNE   READRT            IF SO, ABORT
045A   0605          DEC   R5                DECREMENT SECTOR COUNT
045C   16C2          JNE   READ              IF NOT 0, READ NEXT SECTOR
045E   1E04   READRT SBZ   SEL               TURN OFF DRIVE
044A◆◆1609
0458◆◆1602
0460   045A          B     *R10              RETURN
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 22 of 28)

```
FLOPPY DISK CONTROL PROGRAM                                 PAGE 0023

 0810                    ***********************************************************
 0811                    *
 0812                    *       COMMAND CONTROL PROGRAM:  WRTHEX,WTASCI,WTDDTA
 0813                    *
 0814                    *       THESE COMMANDS ENABLE THE OPERATOR TO WRITE
 0815                    *       A SPECIFIED NUMBER OF SECTORS OF DATA BEGINNING
 0816                    *       AT THE CURRENT TRACK AND SECTOR LOCATION, IN
 0817                    *       EITHER ASCII (WA) OR HEXADECIMAL (WD,WH) FORMAT.
 0818                    *       THE WD COMMAND CAUSES A DELETED DATA MARK TO
 0819                    *       PRECEDE THE DATA, AND THE WA AND WH COMMANDS
 0820                    *       WRITE THE DATA MARK.  IF THE ID FIELD OF
 0821                    *       ANY SECTOR IS NOT FOUND, AN ERROR IS
 0822                    *       REPORTED.
 0823                    *
 0824                    *       ENTRY PARAMETERS:       R10 = RETURN ADDRESS
 0825                    *                                R7 = COMMAND CHARACTERS
 0826                    *                                R5 = NUMBER OF SECTORS
 0827                    *                                     TO WRITE
 0828                    *
 0829  0462  D820  WRTDEL MOVB @DLDMRK,@DTAFLD   LOAD DELETED DATA MARK
       0464  00CF'
       0466  80FF
       0326**0462'
 0830  0468  10--         JMP  WRITE             CONTINUE
 0831        046A' WRTHEX EQU  $                 WH COMMAND ENTRY POINT
       0324**046A'
 0832  046A  D820  WTASCI MOVB @DTMRK,@DTAFLD    LOAD DATA MARK
       046C  00D1'
       046E  80FF
       0322**046A'
 0833  0470  06C7  WRITE  SWPB R7                MOVE SECOND COMMAND
       0468**1003
 0834                    *                       CHARACTER TO LEFT BYTE
 0835  0472  0208  WRITLP LI   R8,DTABUF         LOAD DATA FIELD
       0474  8100
 0836                    *                       IMAGE POINTER
 0837  0476  0200         LI   R0,64             REPEAT 64 TIMES
       0478  0040
 0838  047A  04F8  WTLPL1 CLR  *R8+              CLEAR DATA BUFFER
 0839  047C  0600         DEC  R0
 0840  047E  16FD         JNE  WTLPL1
 0841  0480  0208         LI   R8,DTABUF         LOAD DATA BUFFER POINTER
       0482  8100
 0842  0484  9807         CB   R7,@ASCIIH        WH COMMAND?
       0486  0080'
 0843  0488  13--         JEQ  WRTASC            IF SO, READ ASCII STRING
 0844  048A  C10A         MOV  R10,R4            SAVE RETURN ADDRESS
 0845  048C  020A         LI   R10,WTBRDY        LOAD HRC2 SUBROUTINE
       048E  ----
 0846                    *                       RETURN ADDRESS
 0847  0490  0209         LI   R9,8              LOAD LINE COUNT
       0492  0008
 0848  0494  0206  WTHLP1 LI   R6,16             LOAD BYTE COUNT
       0496  0010
```

Figure 32. Floppy Disk Control Program (Sheet 23 of 28)

```
FLOPPY DISK CONTROL PROGRAM                             PAGE 0024

 0498  2F00          NLIN 0              NEW LINE
 049A  2E98  WTHLP2  HRC2 *R8            READ BYTE
 049C  0588          INC  R8             INCREMENT BUFFER POINTER
 049E  0606          DEC  R6             DECREMENT BYTE COUNT
 04A0  16FC          JNE  WTHLP2         IF NOT 0, READ NEXT BYTE
 04A2  0609          DEC  R9             DECREMENT LINE COUNT
 04A4  16F7          JNE  WTHLP1         IF NOT 0, READ NEXT LINE
 04A6  C284  WTBRDY  MOV  R4,R10         RESTORE RETURN ADDRESS
 048E**04A6'
 04A8  10--          JMP  WTCRCD         CONTINUE
 04AA  0206  WRTASC  LI   R6,128         LOAD CHARACTER COUNT
 04AC  0080
 0488**1310
 04AE  2F00          NLIN 0              NEW LINE
 04B0  2F58  WTASLP  RECV *R8            READ CHARACTER
 04B2  9818          CB   *R8,@ESC       ESCAPE CHARACTER?
 04B4  0083'
 04B6  13--          JEQ  WRITRT         IF SO, RETURN
 04B8  9838          CB   *R8+,@BLANK    NON-PRINTABLE?
 04BA  0084'
 04BC  11--          JLT  WTCRCD         IF SO, END OF SECTOR
 04BE  0606          DEC  R6             DECREMENT CHARACTER COUNT
 04C0  16F7          JNE  WTASLP         IF NOT 0, READ NEXT CHAR
 04C2  2E00  WTCRCD  CRCD 0              GENERATE DATA FIELD CRC
 04A8**100C
 04BC**1102
 04C4  0209          LI   R9,DTAWT       DISK DATA WRITE ADDRESS
 04C6  7FFE
 04C8  0208          LI   R8,DTAFLD      DATA FIELD IMAGE POINTER
 04CA  80FF
 04CC  2C80          IDRD 0              READ ID FIELD
 04CE  0200          LI   R0,16          REPEAT 16 TIMES
 04D0  0010
 04D2  04D9  WTLPL2  CLR  *R9            WRITE LAST 16 BYTES OF
             *                           ID GAP (FIRST BYTE SKIPPED FOR
             *                           BYTE SYNCHRONIZATION)
 04D4  0600          DEC  R0
 04D6  16FD          JNE  WTLPL2
 04D8  D838          MOVB *R8+,@MRKWT    WRITE DATA MARK
 04DA  7F8E
 04DC  0200          LI   R0,130         REPEAT 130 TIMES
 04DE  0082
 04E0  D678  WTLPL3  MOVB *R8+,*R9       WRITE DATA FIELD
 04E2  0600          DEC  R0
 04E4  16FD          JNE  WTLPL3
 04E6  04D9          CLR  *R9            REWRITE FIRST BYTE OF
             *                           DATA GAP
 04E8  2D00          SINC 0              UPDATE SECTOR NUMBER
 04EA  1E04          SBZ  SEL            TURN OFF DRIVE
 04EC  0605          DEC  R5             DECREMENT SECTOR COUNT
 04EE  16C1          JNE  WRITLP         IF NOT 0, WRITE NEXT SECTOR
 04F0  045A  WRITRT  B    *R10           ELSE, RETURN
 04B6**131C
```

▶9

Figure 32. Floppy Disk Control Program (Sheet 24 of 28)

```
FLOPPY DISK CONTROL PROGRAM                               PAGE 0025

                    ****************************************************************
                    *
                    *     COMMAND CONTROL PROGRAM:   FORMAT
                    *
                    *     THIS COMMAND ENABLES THE OPERATOR TO FORMAT
                    *     A NUMBER OF TRACKS BEGINNING AT THE CURRENT TRACK
                    *     AND SPECIFYING THE LAST TRACK.  ALL GAPS,
                    *     TRACK, ID, AND DATA MARKS, AND TRACK AND
                    *     SECTOR NUMBERS ARE WRITTEN.  THE DATA IN THE
                    *     DATA FIELDS IS ALL ZEROES.
                    *
                    *     ENTRY PARAMETERS:       R10 = RETURN ADDRESS
                    *
 04F2   2DA0  FORMAT AXMT @ENDMSG             PRINT END MESSAGE
 04F4   0094'
 032C**04F2'
 04F6   2DA0         AXMT @TKMSG              PRINT TRACK MESSAGE
 04F8   008A'
 04FA   D260         MOVB @TKNUM,R9           FETCH TRACK NUMBER
 04FC   80F8
 04FE   2EC9         HXM2 R9                  PRINT TRACK NUMBER
 0500   2E89         HRC2 R9                  READ LAST TRACK NUMBER
 0502   0289         CI   R9,77*256           LEGAL VALUE?
 0504   4D00
 0506   14--         JHE  FRMTRT              IF NOT, RETURN
 0508   0208         LI   R8,DTAFLD           LOAD DATA FIELD POINTER
 050A   80FF
 050C   DE20         MOVB @DTMRK,*R8+         LOAD DATA MARK
 050E   00D1'
 0510   0200         LI   R0,64               REPEAT 64 TIMES
 0512   0040
 0514   04F8  FFLPL1 CLR  *R8+                CLEAR DATA BUFFER
 0516   0600         DEC  R0
 0518   16FD         JNE  FFLPL1
 051A   2E00         CRCD 0                   CALCULATE THE CRC FOR THE
                    *                         DATA FIELD
 051C   9809  FRMTLP CB   R9,@TKNUM           LAST TRACK LESS
 051E   80F8
                    *                         THAN CURRENT TRACK?
 0520   11--         JLT  FRMTRT              IF SO, RETURN
 0522   0208  FRMT1  LI   R8,>100             LOAD INITIAL SECTOR
 0524   0100
                    *                         VALUE
 0526   0207         LI   R7,SECBUF           LOAD SECTOR BUFFER
 0528   80C0
                    *                         POINTER
 052A   D808  FRIDBL MOVB R8,@SECNUM          UPDATE SECTOR
 052C   80FA
                    *                         NUMBER
 052E   2DC0         CRCI 0                   CALCULATE CRC FOR ID FIELD
 0530   CDE0         MOV  @IDCRC,*R7+         SAVE CRC IN BUFFER
 0532   80FC
 0534   0228         AI   R8,>100             INCREMENT SECTOR NUMBER
 0536   0100
```

9

Figure 32. Floppy Disk Control Program (Sheet 25 of 28)

```
FLOPPY DISK CONTROL PROGRAM                       PAGE 0026

 0538  0288          CI    R8,27*256        LAST SECTOR?
 053A  1B00
 053C  16F6          JNE FRIDBL             IF NOT, REPEAT FOR
              *                             NEXT SECTOR
 053E  0207          LI    R7,SECBUF        LOAD SECTOR BUFFER
 0540  80C0
              *                             POINTER
 0542  0208          LI    R8,>100          LOAD INITIAL SECTOR
 0544  0100
              *                             NUMBER
 0546  2D40          DSON  0                TURN ON DRIVE
 0548  0206   FMINDX LI    R6,DTAWT         DISK DATA WRITE
 054A  7FFE
 054C  04E0          CLR   @INDXWT          WRITE 0 AT INDEX PULSE
 054E  7FFA
 0550  0200          LI    R0,45            REPEAT 45 TIMES
 0552  002D
 0554  04D6   FFLPL2 CLR   *R6              WRITE REST OF POST-INDEX
              *                             GAP
 0556  0600          DEC   R0
 0558  16FD          JNE   FFLPL2
 055A  D820          MOVB  @TKMRK,@TKMWT    WRITE TRACK MARK
 055C  00D2'
 055E  7F9E
 0560  0200   SECTLP LI    R0,32            REPEAT 32 TIMES
 0562  0020
 0564  04D6   FFLPL3 CLR   *R6              WRITE 32 BYTE GAP
 0566  0600          DEC   R0
 0568  16FD          JNE   FFLPL3
 056A  D820          MOVB  @IDMRK,@MRKWT    WRITE ID MARK
 056C  00D0'
 056E  7F8E
 0570  D5A0          MOVB  @TKNUM,*R6       WRITE TRACK NUMBER
 0572  80F8
 0574  D58C          MOVB  R12,*R6          WRITE SECOND BYTE
 0576  D588          MOVB  R8,*R6           WRITE SECTOR NUMBER
 0578  0228          AI    R8,>100          INCREMENT SECTOR NUMBER
 057A  0100
 057C  D58C          MOVB  R12,*R6          WRITE FOURTH BYTE
 057E  D5B7          MOVB  *R7+,*R6         WRITE CRC1
 0580  D5B7          MOVB  *R7+,*R6         WRITE CRC2
 0582  0200          LI    R0,17            REPEAT 17 TIMES
 0584  0011
 0586  04D6   FFLPL4 CLR   *R6              WRITE ID GAP
 0588  0600          DEC   R0
 058A  16FD          JNE   FFLPL4
 058C  0204          LI    R4,DTAFLD        LOAD DATA FIELD
 058E  80FF
              *                             IMAGE POINTER
 0590  D834          MOVB  *R4+,@MRKWT      WRITE DATA MARK
 0592  7F8E
 0594  0200          LI    R0,130           REPEAT 130 TIMES
 0596  0082
 0598  D5B4   FFLPL5 MOVB  *R4+,*R6         WRITE DATA AND CRC
 059A  0600          DEC   R0
```

Figure 32. Floppy Disk Control Program (Sheet 26 of 28)

```
FLOPPY DISK CONTROL PROGRAM                           PAGE 0027

059C  16FD         JNE  FFLPL5
059E  04D6         CLR  *R6                 WRITE PAD BYTE
05A0  0288         CI   R8,>27*256          LAST BYTE?
05A2  2700
05A4  16DD         JNE  SECTLP              IF NOT, FORMAT NEXT SECTOR
05A6  04D6  PREILP CLR  *R6                 WRITE PRE-INDEX GAP
05A8  1F04         TB   INDEX               UNTIL INDEX
05AA  16FD         JNE  PREILP              PULSE OCCURS
05AC  2E40         TINC 0                   STEP HEAD TO NEXT TRACK
05AE  10B6         JMP  FRMTLP              FORMAT NEXT TRACK
05B0  1E04  FRMTRT SBZ  SEL                 TURN OFF DRIVE
0506**1454
0520**1147
05B2  045A         B    *R10                RETURN
            ************************************************************
            *
            *      COMMAND CONTROL PROGRAM:  EXECUT
            *
            *      THIS COMMAND ENABLES THE OPERATOR TO BEGIN
            *      EXECUTION OF A PROGRAM AT ANY LOCATION
            *      IN MEMORY.
            *
            *      ENTRY PARAMETERS:        R8 = ENTRY POINT
            *
05B4  0458  EXECUT B    *R8                 BRANCH TO ENTRY POINT
0332**05B4
            ************************************************************
            *
            *      COMMAND CONTROL PROGRAM:  ENTER
            *
            *      THIS COMMAND ENABLES THE OPERATOR TO ENTER
            *      DATA INTO SEQUENTIAL MEMORY LOCATIONS.
            *
            *      CALLING PARAMETERS:      R8 = BEGINNING MEMORY
            *                                    LOCATION
            *
05B6  0209  ENTER  LI   R9,8                LOAD BYTE COUNT
05B8  0008
0330**05B6
05BA  2F00         NLIN 0                   NEW LINE
05BC  2EC8         HXM2 R8                  PRINT FIRST BYTE OF ADDRESS
05BE  06C8         SWPB R8                  REVERSE BYTES
05C0  2FA0         XMIT @BACKSP             BACKSPACE
05C2  0087
05C4  2EC8         HXM2 R8                  PRINT SECOND BYTE OF ADDRESS
05C6  06C8         SWPB R8                  RESTORE BYTES
05C8  2ED8  ENTLP  HXM2 *R8                 PRINT MEMORY CONTENTS
05CA  2E98         HRC2 *R8                 READ AND STORE NEW VALUE
05CC  0588         INC  R8                  UPDATE ADDRESS POINTER
05CE  0609         DEC  R9                  DECREMENT BYTE COUNT
05D0  13F2         JEQ  ENTER               IF 0, NEW LINE
05D2  10FA         JMP  ENTLP               ELSE, FETCH NEXT BYTE
```

Figure 32. Floppy Disk Control Program (Sheet 27 of 28)

```
FLOPPY DISK CONTROL PROGRAM                            PAGE 0028

                *************************************************************
                *
                *       COMMAND CONTROL PROGRAM:   DUMP
                *
                *       THIS COMMAND ENABLES THE OPERATOR TO
                *       DISPLAY THE CONTENTS OF MEMORY IN
                *       HEXADECIMAL FORMAT.
                *
                *       CALLING PARAMETERS:      R8 = BEGINNING
                *                                     ADDRESS
                *
05D4   C248     DUMP    MOV   R8,R9              LOAD DEFAULT END
032E**05D4'
                *                                ADDRESS
05D6   2E89             HRC2  R9                 READ FIRST BYTE OF
                *                                END ADDRESS
05D8   06C9             SWPB  R9                 SAVE IN RIGHT BYTE
05DA   2FA0             XMIT  @BACKSP            BACKSPACE
05DC   0087'
05DE   2E89             HRC2  R9                 READ SECOND BYTE OF
                *                                END ADDRESS
05E0   06C9             SWPB  R9                 SWAP BYTES
05E2   2F00     DUMPLP  NLIN  0                  NEW LINE
05E4   0207             LI    R7,16              LOAD BYTE COUNT
05E6   0010
05E8   2EC8             HXM2  R8                 PRINT FIRST BYTE OF ADDRESS
05EA   06C8             SWPB  R8                 REVERSE BYTES
05EC   2FA0             XMIT  @BACKSP            BACKSPACE PRINTER
05EE   0087'
05F0   2EC8             HXM2  R8                 PRINT SECOND BYTE OF ADDRESS
05F2   06C8             SWPB  R8                 CORRECT ADDRESS
05F4   2ED8     DMPLP1  HXM2  *R8                PRINT MEMORY CONTENTS
05F6   8209             C     R9,R8              CURRENT ADDRESS = LAST
                *                                ADDRESS
05F8   13--             JEQ   DUMPRT             IF SO, RETURN
05FA   0588             INC   R8                 INCREMENT ADDRESS
05FC   0607             DEC   R7                 DECREMENT BYTE COUNT
05FE   16FA             JNE   DMPLP1             IF NOT 0, PRINT NEXT
                *                                BYTE
0600   1F00             TB    PIN                OPERATOR INTERRUPT?
0602   13EF             JEQ   DUMPLP             IF NOT, PRINT NEXT LINE
0604   045A     DUMPRT  B     *R10               ELSE, RETURN
05F8**1305
                        END

0000 ERRORS

ASM TERM? T
```

Figure 32. Floppy Disk Control Program (Sheet 28 of 28)