



INTEGRATED
COMPUTER
SYSTEMS™

MICROCOMPUTER INTERFACING

WORKBOOK

EDWARD DILLINGHAM, M.E., M.S.E.E.

ASSISTED BY

Dr. David C. Collins

Mr. Eric R. Garen

Mr. Manolito E. Adan

Mr. Scott N. Smith

Don Black, Technical Editor

Published By

INTEGRATED COMPUTER SYSTEMS, INC.

© Copyright 1978

LOS ANGELES

BRUSSELS

TABLE OF CONTENTS

TABLE OF CONTENTS	i
LIST OF ILLUSTRATIONS	vi
1 HARDWARE INTERFACING AND REAL TIME PROGRAMMING	
1.1 Introduction	1-1
1.2 Purpose and Content of the Course	1-1
1.3 Cables and Connections	1-5
1.3.1 Power Connections	1-5
1.3.2 Signal Terminals	1-7
1.4 Interface Hardware and References	1-9
1.4.1 MTS Interface	1-9
1.4.2 Added Memory	1-13
1.4.3 Chip Selects and Resets	1-13
1.4.4 Port 1 and A/D - D/A Converter	1-17
1.4.5 Interrupt System	1-17
1.4.6 Optical Couplers and Power Driver	1-20
1.4.7 Serial Interface Circuit	1-20
1.4.8 Tape Cassette Modem	1-20
2 INPUT/OUTPUT AND INTERRUPTS	
2.1 Port Assignments and Addresses	2-1
2.2 Programming and Using the 8255	2-4
2.3 Port 1A LED's and Drivers	2-10
2.4 MTS Display	2-12
2.5 Input/Output Connections	2-15
2.6 External Inputs 4 and 5	2-16
2.7 Interrupt Flip-Flops and Enables	2-19
2.7.1 Interrupt Sources	2-19
2.7.2 Interrupt Flip-Flops	2-21
2.7.3 Interrupt Status and Enables	2-23
2.7.4 Clearing Interrupts	2-25

2.8	Restart Instructions	2-30
2.8.1	Automatic RST 7	2-30
2.8.2	RST Generation	2-35
2.9	Interrupt Service for EXT 4 and EXT 5	2-39
2.10	Standard Programming for 8255's	2-47
3	INTERVAL TIMERS	
3.1	Intel 8253 Interval Timer	3-1
3.2	Clock, Gate, and Output	3-5
3.3	Timer Modes	3-9
3.4	Mode 0 - Interrupt on Terminal Count	3-13
3.5	Restarting a Counter in Mode 0	3-21
3.6	Reading a Timer	3-27
3.6.1	Measuring a Pulse Duration	3-27
3.6.2	Additional Exercises	3-36
3.6.3	Reading While Counting	3-38
3.7	Mode 2 - Rate Generator	3-41
3.7.1	Use of Mode 2	3-41
3.7.2	Real Time Clock	3-45
3.8	Cascaded Timers	3-53
3.9	Mode 3-Square Wave Generator	3-63
3.9.1	Observing the Output	3-63
3.9.2	Observing the Counting	3-65
3.10	Timer Mode Descriptions	3-68
4	DIGITAL TO ANALOG OUTPUT	
4.1	Methods of D/A Output	4-3
4.2	Pulse Width Modulation	4-5
4.2.1	PWM Output Program	4-6
4.3	Frequency Control	4-28
4.3.1	Audio Tone Generator	4-30
4.3.2	Frequency Modulation Program	4-32
4.3.3	Recorded Music Player	4-36
4.3.4	Music Recording Program	4-53

4.4	Multi-bit Output	4-56
4.5	Analog Voltage Generation	4-60
4.5.1	Binary Summing Circuit	4-60
4.5.2	R-2R Ladder Network	4-66
4.6	Ferranti D/A Converter	4-72
4.6.1	D/A Circuit Input and Output	4-74
4.6.2	D/A Circuit Control Signals	4-74
4.6.3	Generating an Analog Voltage	4-77
4.7	Function Generator	4-79
4.7.1	Voltage Ramps	4-81
4.7.2	Keyboard Controlled Function Generators	4-87
4.7.3	Exponential Function	4-117
5	ANALOG TO DIGITAL INPUT	
5.1	Pulse Interval Measurement	5-3
5.1.1	Measuring a Steady Signal	5-3
5.1.2	Measuring a Multi-Valued Interval	5-6
5.1.3	Measuring Received Pulse Intervals	5-25
5.2	Frequency Measurement	5-27
5.2.1	Logic Level Frequency Measurements	5-27
5.2.2	AC Input Signal	5-31
5.3	A/D Input - Voltage	5-43
5.3.1	Output, Input and Display Subroutine	5-45
5.3.2	Ramping Voltmeter	5-55
5.3.3	Tracking Voltmeter	5-65
5.3.4	Successive Approximation Voltmeter	5-68
5.4	Automatic A/D Input	5-75
5.4.1	Reading A/D Input	5-79
5.4.2	A/D Input with Interrupt	5-85

5.5	Digital Noise Filter	5-90
5.5.1	Filter Program Algorithm	5-91
5.5.2	Program Definitions	5-92
5.5.3	Filter Response	5-107
5.6	Temperature Measurement	5-109
5.6.1	Thermistor Characteristics	5-109
5.6.2	Thermistor Operation	5-113
5.6.3	Thermistor Input Adjustment	5-115
5.6.4	Table Lookup and Interpolation	5-116
5.6.5	Voltage to Temperature Conversion	5-119
5.6.6	Thermometer Program	5-130
5.6.7	Data Logging	5-143
5.6.8	Thermistor Self Heating	5-153
5.6.9	Other Temperature Logging Experiments	5-160
5.6.10	Abbreviated Temperature Lookup	5-160
5.6.11	Thermistor Resistance Matching	5-165
6	CLOSED LOOP CONTROL	
6.1	On-Off Control	6-3
6.1.1	On-Off Control Without Deadband	6-8
6.1.2	On-Off Control With Deadband	6-21
6.1.3	Thermostat With Alarm Limits	6-26
6.1.4	Two-Way Control	6-29
6.2	Proportional vs Integral Control	6-33
6.2.1	Voltage Control Circuit	6-41
6.2.2	Voltage Control by PWM	6-49
6.2.3	Observing Response Time	6-79
6.2.4	Closing the Loop	6-84
6.2.5	Closed Loop Operation	6-101
6.2.6	Closed Loop Response	6-107
6.3	Proportional Plus Integral Control	6-117
6.3.1	Applying Gain to Error Signal	6-118
6.3.2	Subroutine CLOSL Version 2	6-121
6.3.3	Subroutine INTEG Version 2	6-123
6.3.4	Experiments With PI Control	6-132

6.3.5	Full Scale Control and Overflow	6-142
6.4	Proportional - Integral - Differential Control	6-161
6.5	Summary	6-163

7 MOTOR CONTROL

7.1	Optical Disc and Slot Sensor	7-3
7.1.1	Motor, Sensor and Disc Mounting	7-5
7.1.2	Slot Sensor Connections and Test	7-8
7.1.3	Motor Connection	7-13
7.1.4	Motor Characteristics	7-17
7.2	Control System Development	7-21
7.2.1	Speed Measurement	7-25
7.2.2	Interrupt Service	7-29
7.2.3	Initialization	7-36
7.2.4	Main Program Loop	7-39
7.2.5	Keyboard Input Subroutine KYTIM	7-43
7.2.6	Subroutine DECBI	7-47
7.2.7	Subroutines SMULT, SCUML	7-48
7.2.8	Open Loop Operation	7-49
7.2.9	False Speed Indications	7-50
7.3	Closed Loop Motor Control	7-53
7.3.1	Subroutine Speed	7-55
7.3.2	Subroutine Width	7-58
7.3.3	Subroutine Divid	7-61
7.3.4	Summary of Subroutines	7-65
7.3.5	Program Implementation	7-66
7.3.6	Motor Control Program Operation	7-97
7.4	Motor Control by Variable Voltage	7-99
Appendix A	Reference Figures	A-1
Appendix B	MTS - Interface Board Modifications	B-1
Appendix C	Cable and Edge Connector Designations	C-1
Appendix D	Program Cassette Library, Instructions & Listings	D-1
Appendix E	RS232 Program Listing	E-1

LIST OF ILLUSTRATIONS

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1a	Photo of MTS and Interface Board	1-4
1-1b	Interface Training System	1-6
1-2	Microcomputer Interfacing System	1-8
1-3	List of Interface Signals to MTS	1-10
1-4	Added Memory (8400-87ff)	1-12
1-5	I/O Chip Selects and Interrupt Flip-Flop Resets	1-14
1-6	Truth Table for Chip Selects and Resets	1-15
1-7	Port 1 and A/D - D/A Converter	1-16
1-8	Vectored Priority Interrupt System	1-18
1-9	Optical Couplers and Power Driver	1-19
1-10	Serial Interface Circuit	1-21
1-11	Tape Cassette Modem	1-22
2-1	I/O Port Assignments	2-2
2-2	Port Addresses and Assignments	2-3
2-3	Programming the 8255's	2-9
2-4	MTS Keyboard Configuration and Port Assignment	2-7
2-5	Input/Output Connections	2-14
2-6	Interrupt System - Partial Diagram	2-18
2-7	EXT 4 and EXT 5 Connections and Signals	2-20
2-8	Clearing Interrupts - Flowchart	2-26
2-9	Clearing Interrupt Flip Flops	2-28
2-10	Automatic RST 7 Generation	2-33
2-11	Generation of RST Instructions	2-34
2-12	Interrupt Service - RST5, RST6 - Flowchart	2-38
2-13	Status and Command Bytes	2-41
2-14	EXT 4 and EXT 5 Service	2-42
2-15	Standard Programming for 8255's	2-46

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-1	Intel 8253 Interval Timer	3-3
3-2	Timer Clocks, Gates, and Outputs	3-4
3-3	Timer Control Byte Structure	3-10
3-4	Timer Control Bytes	3-12
3-5	Compare Timing Loop with Interval Timer - Flowchart	3-14
3-6	Compare Timing Loop with Interval Timer - Program	3-18
3-7	GETKY Flow Diagram	3-20
3-8	GETKY with Timer	3-22
3-9	GETKY Using Interval Timer	3-25
3-10	Pulse Measurement Connection Point	3-26
3-11	Twos and Tens Complement Counting	3-29
3-12	Time Diagram for Pulse Width Measurements	3-31
3-13	Pulse Width Measurement	3-32
3-14	Pulse Width Measurement	3-33
3-15	Timer and Flip-Flop Operation, Mode 2 rate Generator	3-42
3-16	Time of Day Clock	3-44
3-17	Time of Day	3-48
3-18	RST 5 Interrupt Service	3-46
3-19	Cascaded Timers	3-52
3-20	Cascading Timers with Gate Input	3-54
3-21	Time Delay Program	3-57
3-22	Square Waver Generator - Mode 3	3-62
3-23	Reading the Timer Contents	3-67
3-24	8253 Timer Modes	3-70
3-25	Timing Relationships	3-71
4-1	A/D and D/A Conversion	4-1
4-2	Output Connections for PWM	4-6
4-3	PWM Interrupt Service	4-9
4-4	PWM Test Program	4-11
4-5	PWM Main Program	4-13
4-6	Pulse Width Modulation Program	4-19

<u>Figure</u>	<u>Title</u>	<u>Page</u>
4-7	Conversion of Binary Count to Time	4-24
4-8	Audio Output Program and Circuit	4-29
4-9	List of Concert Pitch Musical Tones	4-31
4-10	Tone Generator - Main Program	4-33
4-11	Tone Generator - Interrupt Service	4-34
4-12	Codes for Musical Notes	4-35
4-13	Tune - Main Flow	4-37
4-14	Tune Interrupt Service - Flowchart	4-39
4-15	Tune Program	4-43
4-16	Tone Table for Chromatic Scale	4-48
4-17	Home on the Range, and the Drunken Sailor	4-51
4-18	Music Recording Program, Hex Key Chart	4-55
4-19	Patch to Display Tone	4-59
4-20	Binary Summing Circuit	4-61
4-21	Numerical Values for Circuit of 4-20	4-62
4-22	Binary Summing Circuit with Op Amp	4-63
4-23	R-2R Ladder Network	4-65
4-24	R-2R Ladder Impedance	4-68
4-25	Equivalent Circuits for Single Bit = 1	4-69
4-26	D/A Converter Output Circuit	4-71
4-27	Ferranti D/A Converter	4-73
4-28	Keyboard to Voltage Program Flow and Circuit Connection	4-76
4-29	Voltage Ramp Generators	4-80
4-30	Triangular Function Generator	4-84
4-31	Keyboard Controlled Function Generator	4-86
4-32	Keyboard Controlled Function Generator	4-88
4-33	Ramp - Dispatch	4-90
4-34	Function - Key Input Processing	4-92
4-35	Timer 0 Interrupt Service	4-96
4-36	TRIWF Function Generator	4-102
4-37	Function Generator	4-109
4-38	Exponential Functions	4-119

<u>Figure</u>	<u>Title</u>	<u>Page</u>
4-39	Successive Charge/Discharge Cycles	4-122
4-40	Key Selection of Waveform	4-125
4-41	EXPV	4-129
4-42	Subroutine BMULT	4-134
4-43	Test Program for EXPV	4-140
4-44	Function Generation	4-142
4-45	Multiplication - Subroutine BMULT	4-150
4-46	Function Subroutine EXPV	4-151
5-1	Pulse Interval Measurement - Flowchart	5-2
5-2	Pulse Interval Measurement - Program	5-4
5-3	Multi-Valued Interval - Flowchart	5-8
5-4	Multi-Valued Interval - Program	5-17
5-5	Frequency Measurement - Flowchart	5-26
5-6	Frequency Measurement - Program	5-28
5-7	Comparator Circuit and Protection	5-32
5-8	AC Frequency Measurement - RST6	5-34
5-9	AC Frequency Measurement - Program	5-38
5-10	Connections for Voltmeter Experiments	5-42
5-11	Output, Input, and Display Subroutine	5-46
5-12	Test Program for OIDSF	5-48
5-13	OIDSF - Program	5-51
5-14	Voltage Ramp Generator - Flowchart	5-54
5-15	D/A Outputs and Inputs	5-55
5-16	Voltage Ramp Generator - Program	5-56
5-17	Ramping Voltmeter - Flowchart	5-58
5-18	Ramping Voltmeter - Program	5-60
5-19	Tracking Voltmeter - Flowchart	5-64
5-20	Tracking Voltmeter - Program	5-67
5-21	Successive Approximation - Signals	5-68
5-22	Successive Approximation Voltmeter - Flowchart	5-70

<u>Figure</u>	<u>Title</u>	<u>Page</u>
5-23	Successive Approximation Voltmeter - Program	5-72
5-24	Ferranti A/D Logic	5-74
5-25	Reading A/D Input - Flowchart	5-78
5-26	Reading A/D Input - Program	5-80
5-27	A/D Input with Interrupt - Flowchart	5-84
5-28	A/D Input with Interrupt - Program	5-86
5-29	Filter Subroutines - Flowchart	5-94
5-30	A/D Input with Filter - Program	5-100
5-31	FILTR Response	5-106
5-32	Thermistor Resistance	5-110
5-33	Voltage Vs Thermistor Temperature	5-112
5-34	Expected Voltage At Room Temperature	5-114
5-35	Temperature Conversion by Integration	5-118
5-36	Thermistor Calibration Data	5-120
5-37	Temperature Subroutine - Flowchart	5-122
5-38	Temperature Subroutine - Program	5-124
5-39	Thermometer Program - Flowchart	5-132
5-40	Thermometer Program	5-134
5-41	Logging Thermometer - Main	5-144
5-42	Logging Thermometer - Review Data	5-146
5-43	Logging Thermometer - Replay	5-147
5-44	Logging Thermometer - Timing	5-148
5-45	Logging Thermometer - Program	5-149
5-46	Self-heating Experiment - Connection & Calibration	5-154
5-47	Thermistor Self-heating - Program	5-157
5-48	Abbreviated Temperature Lookup	5-161
5-49	Thermistor Resistor Matching	5-164
5-50	Thermistor Resistor Matching - Flowchart	5-167
6-1	Connections for Thermostat	6-2
6-2	Connections for On-Off Voltage Control	6-4
6-3	On-Off Control, No Deadband	6-6
6-4	Thermostat Program	6-9

<u>Figure</u>	<u>Title</u>	<u>Page</u>
6-5	Thermostat with Deadband	6-20
6-6	Thermostat with Deadband - Program	6-22
6-7	Heating and Cooling Simulation	6-28
6-8	Connections for Simulation	6-29
6-9	Heating and Cooling Limits	6-30
6-10	Connections for PWM Experiment	6-40
6-11	PWM Voltage - Fixed Period	6-44
6-12	PWM Voltage Control - Main Loop	6-48
6-13	PWM Voltage - Subroutine KYTIM	6-50
6-14	Voltmeter Subroutine	6-56
6-15	PWM Timer Operation	6-60
6-16	PWM Interrupt Service	6-62
6-17	PWM Voltage Control - Program	6-68
6-18	Logging Voltmeter	6-78
6-19	Logging Voltmeter Program	6-80
6-20	PWM - Open Loop Response	6-81
6-21	Subroutines CLOSL, INTEG	6-86
6-22	REG Module of KYTIM	6-94
6-23	Closed Loop Program	6-96
6-24	Open and Closed Loop Waveforms	6-108
6-25	Closed Loop Response Waveform	6-110
6-26	Effect of Total Period	6-112
6-27	Subroutines CLOSL and INTEG	6-120
6-28	CLOSL and INTEG - Program	6-125
6-29	Response with Proportional Control	6-134
6-30	Response with Integral Control	6-136
6-31	Proportional Plus Integral Response	6-138
6-32	Response to Voltage Request	6-140
6-33	LDT1 with Full Scale Control	6-144
6-34	KYTIM - Revised LDT1 - Program	6-146
6-35	Full Scale Response to Voltage Request	6-156

<u>Figure</u>	<u>Title</u>	<u>Page</u>
7-1	Motor and Slot Sensor	7-2
7-2	Motor, Sensor and Disc Mounting	7-4
7-3	Optical Slot Sensor	7-6
7-4	Test for Slot Sensor	7-10
7-5	Motor Connections	7-12
7-6	Motor Connections with External Power	7-14
7-7	Motor Speed vs Voltage	7-16
7-8	Motor Speed vs Duty Cycle Open Loop	7-18
7-9	Motor Control Program Structure	7-20
7-10	Motor Control Interrupt Manager	7-28
7-11	EXT4 Interrupt Service	7-30
7-12	Timer 0 Service	7-34
7-13	Motor Control - Main Loop	7-38
7-14	KYTIM - Input and Dispatch	7-42
7-15	Load Timer 1 Modules	7-46
7-16	Motion Detection with Dual Sensors	7-52
7-17	Subroutine SPEED	7-54
7-18	Subroutine WIDTH	7-57
7-19	Subroutine DIVID	7-60
7-20	Subroutine Register Usage	7-20
7-21	Motor Control - Initialize	7-69
7-22	Patches for Variable Voltage Control	7-101

MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 1

INTRODUCTION

HARDWARE INTERFACING AND REAL TIME PROGRAMMING

1.1 INTRODUCTION

All computer applications involve the connection of the computer to external hardware for input and output. In computational systems these external devices are slaves to the computer, and they exist only to serve the computer. In control applications, however, the computer (or microcomputer) exists to serve the process, and the computer design and programming must be adapted to the process. In this course we will be concerned with control applications: how a microprocessor is connected to equipment it controls, and how it is programmed to meet process requirements.

In most control applications the computer must receive input data, process the data and generate control outputs in a timely fashion in order to achieve its intended goals. Failure to react in the allowed time will result in loss of data and possibly improper control. Real time programming deals with these requirements. Most of the exercises in this course are real time programs.

1.2 PURPOSE AND CONTENT OF THE COURSE

The text and exercises of this course teach the use of programmed, timed, and interrupt driven input and output. These are applied to open and closed loop control problems, with various forms of discrete and analog input signals. Sensor calibration is used to convert a thermistor signal to temperature, and the speed of a motor is measured using an optical sensor. Logarithmic and sinusoidal output signals are generated. A digital noise filter is developed. The student will

This Page Intentionally Left Blank

measure the response time of a closed loop control system, after observing the difference in behavior between open loop and closed loop control. Although no attempt is made to teach servomechanism and feedback theory, the basic ideas of proportional, and integral feedback are presented and used in exercises.

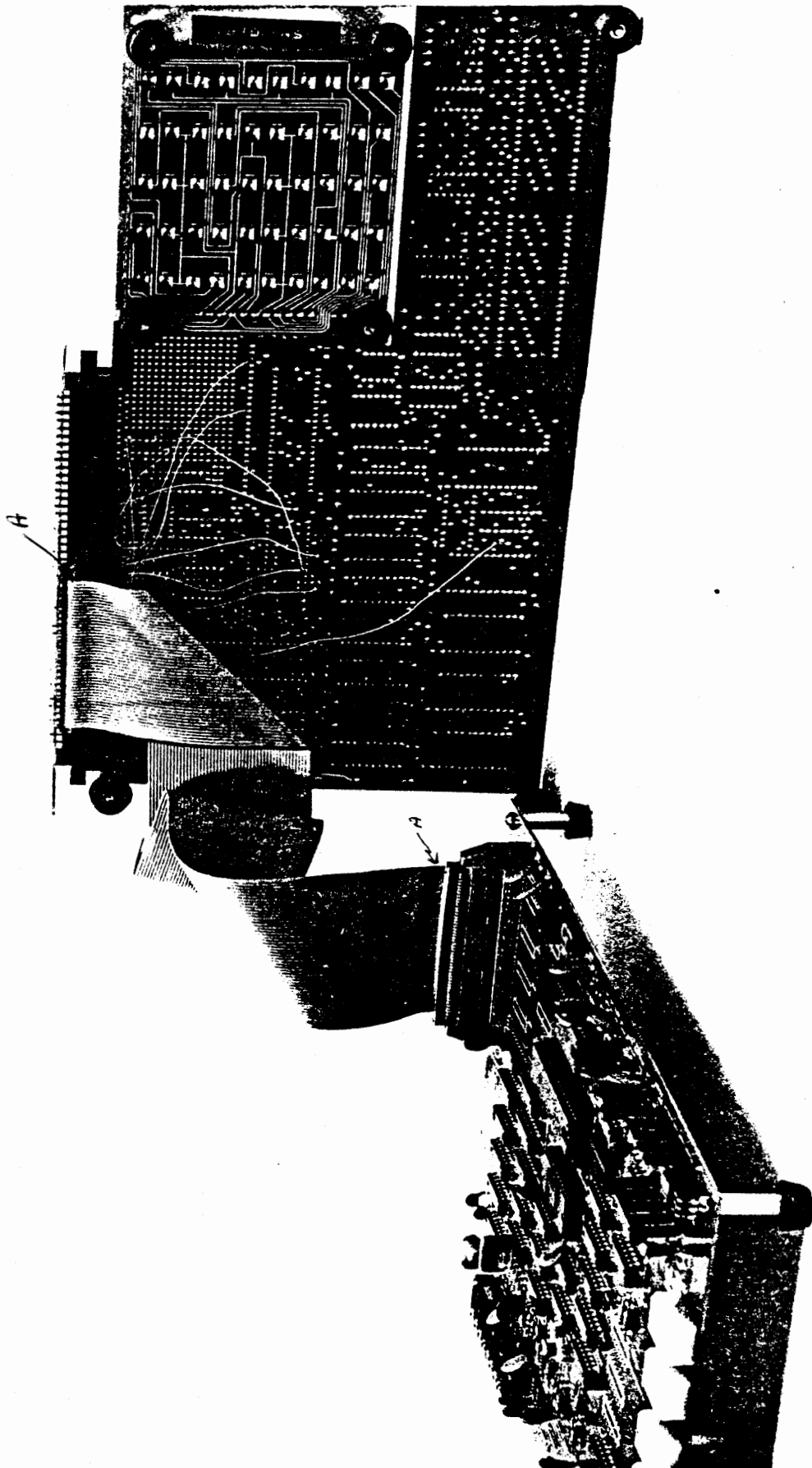
The interface circuit board includes an interrupt system with priorities and vectors. These are explained and used in many exercises having multiple interrupts.

The manufacturers of microprocessors are introducing new LSI chips to make real time control systems easier to design and cheaper to build. The background provided through this course will make such devices comprehensible to the engineer and programmer. One such device, the INTEL 8253 interval timer, is included in the course hardware and extensively treated.

The remainder of this chapter gives an introduction to the hardware of the interface circuit board. Complete schematics are included here, but details of how various parts of the hardware operate are covered along with exercises in later chapters.

FIGURE 1-1

PHOTO OF MTS AND INTERFACE BOARD WITH
MTS TILTED TO SHOW ITS BOTTOM WITH FOLDED CABLE



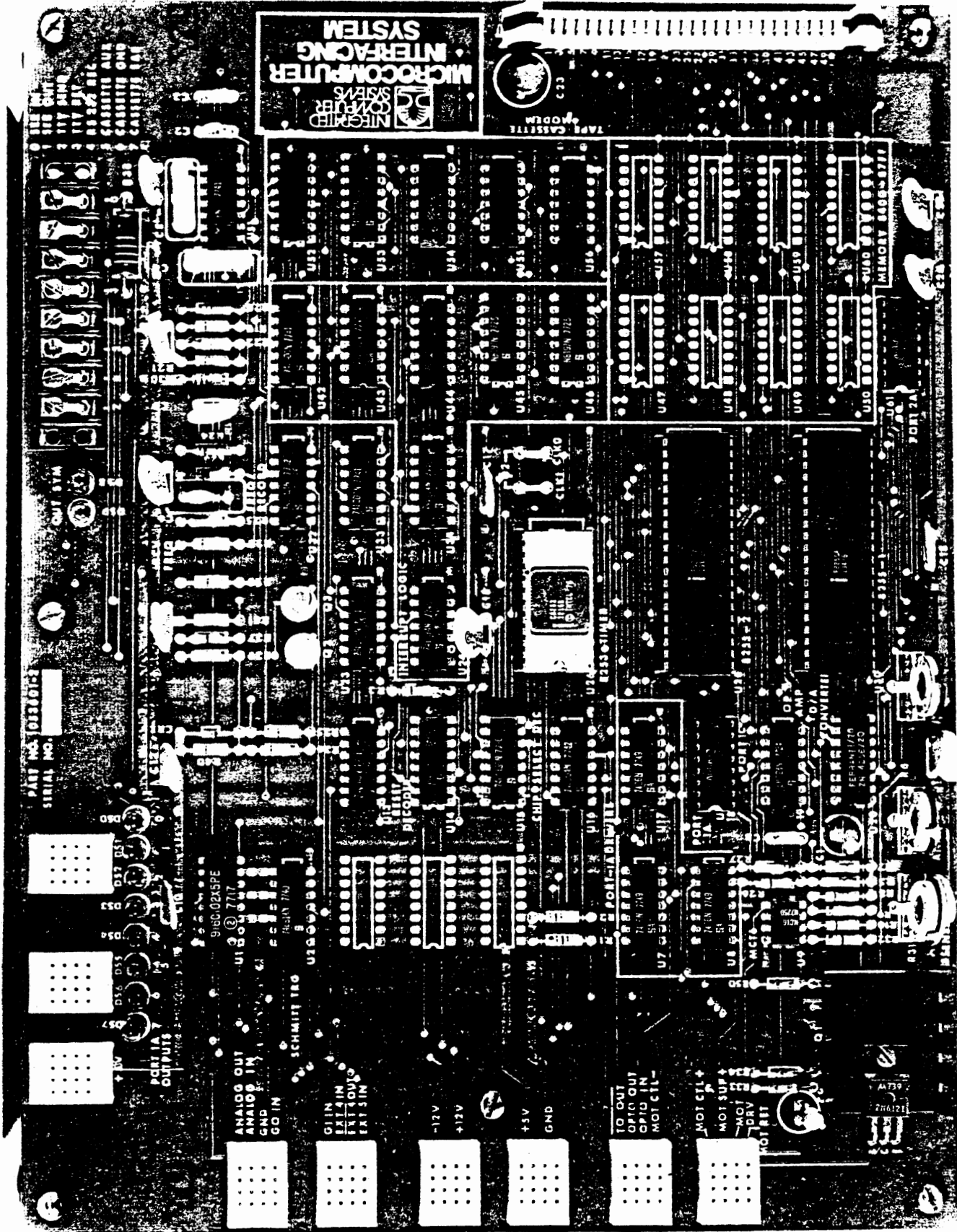
1.3 CABLES AND CONNECTIONS

The interface circuit board is connected to the Microcomputer Training System through a ribbon cable and an adaptor board with an edge connector that plugs onto the MTS. The MTS as originally supplied has only the address bus and data bus available at the edge connector. Various control signals are brought to the edge connector by jumpers. If the MTS and Course 525 have been supplied with Course 536, these jumpers will have been installed by ICS. Otherwise the connections must be made in accordance with Appendix B.

The adaptor circuit board via its connector is plugged into the MTS, with the ribbon cable running under the MTS and folded to come out to the left, as shown in Figure 1-1. The cable is plugged into the connector at the right hand side of the interface board.

1.3.1 Power Connections

Normally the power connections (+5 volts, +12 volts, ground) are made directly to the MTS through wires soldered to pads at the lower left corner. The interface board then obtains the power through the ribbon cable from the MTS. Alternately it is possible to make these connections to the tie block in the center of the left hand side of the interface board. There is no negative supply required, unless the serial interface port for a teletype or an RS232 terminal is to be used.



INTERFACE TRAINING SYSTEM
FIGUP 1-1b

1.3.2 Signal Terminals

Signals used to connect the interface board to external devices, or to connect various functions together for experiments, are made through tie blocks at the left and top edges. The white plastic tie blocks each have four different signals, labelled next to the block. Each row in a block is a common line, making it easy to tie several signals together. Wires or component leads can be inserted directly into these tie blocks. One block, at the upper left corner, has +5 volts at all points to facilitate insertion of pullup resistors.

A row of screw terminals at the upper right provides for connections to serial ports.



MICROCOMPUTER INTERFACING SYSTEM

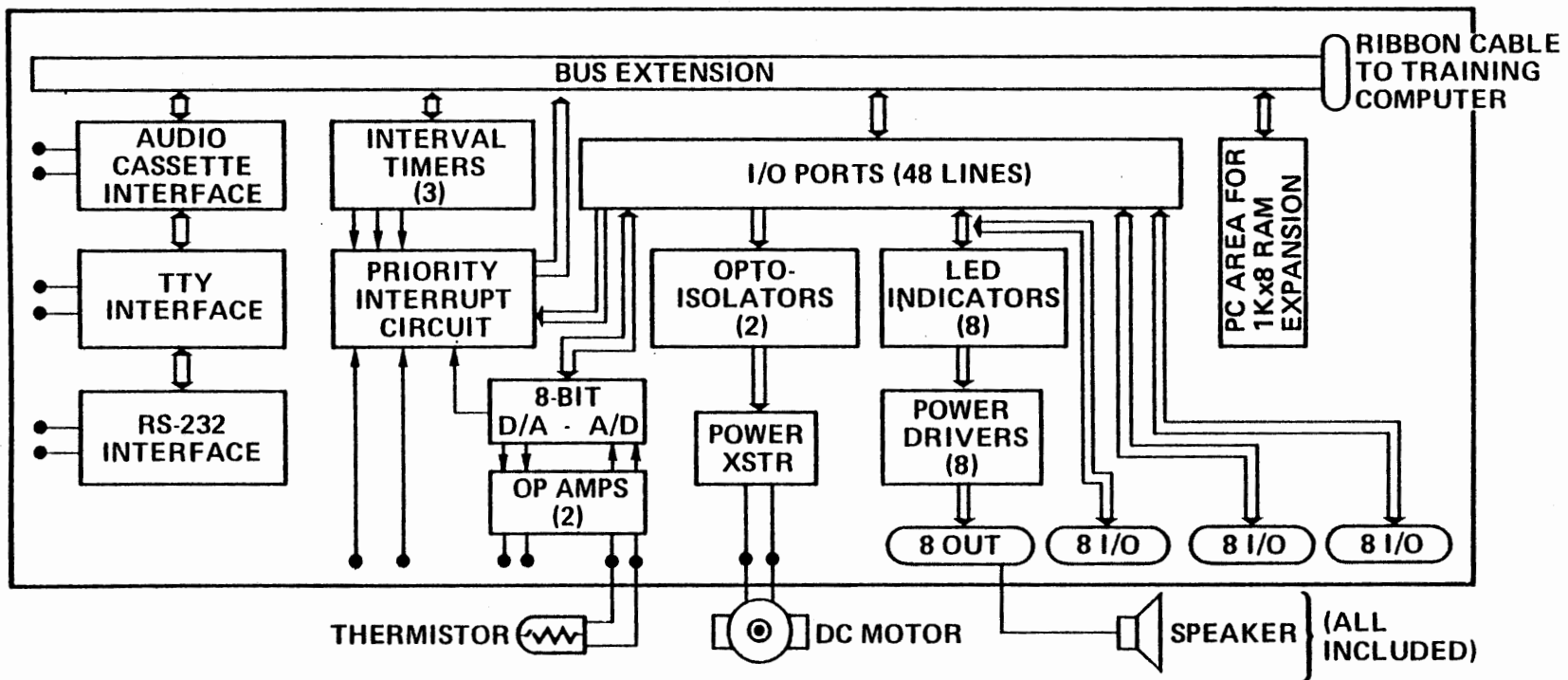


FIGURE 1-2

1.4 INTERFACE HARDWARE AND REFERENCES

An overall block diagram of the interface circuit board is shown in Figure 1-2. Various sections are shown in separate schematic diagrams and described in the chapter referred to below.

1.4.1 MTS Interface

Figure 1-3 lists the signals that are brought out to the MTS via the 50 pin connector with their pin assignments in the connector head. The corresponding connections within the MTS 100 pin edge connector are described in Appendix C.

CONNECTOR PIN	SIGNAL NAME	CONNECTIONS ON INTERFACE BOARD
1	GND	
2	GND	
3	GND	
4	GND	
5	Vcc (+5 Volts)	
6	Vcc (+5 Volts)	
7	GND	
8	+12 Volts	
9	+12 Volts	
10	GND	
11	GND	
12	CLK \emptyset 2	(U26-18) (CLK1 Tiepoint) (CLK2 Tiepoint)
13	GND	
14	AB15	(U44-11)
15	AB7	(MEM-16)
16	AB6	(MEM-1)
17	AB5	(MEM-2)
18	AB4	(MEM-7) (U15-13)
19	AB3	(MEM-6) (U15-14)
20	AB10	(U44-10)
21	AB2	(MEM-5) (U16-9)
22	AB9	(MEM-14)
23	AB1	(MEM-4) (U26-20) (U28-8) (U30-8) (U13-5)
24	AB8	(MEM-15)
25	AB0	(MEM-8) (U26-19) (U28-9) (U30-9) (U13-4)

LIST OF INTERFACE SIGNALS TO MTS

FIGURE 1-3a

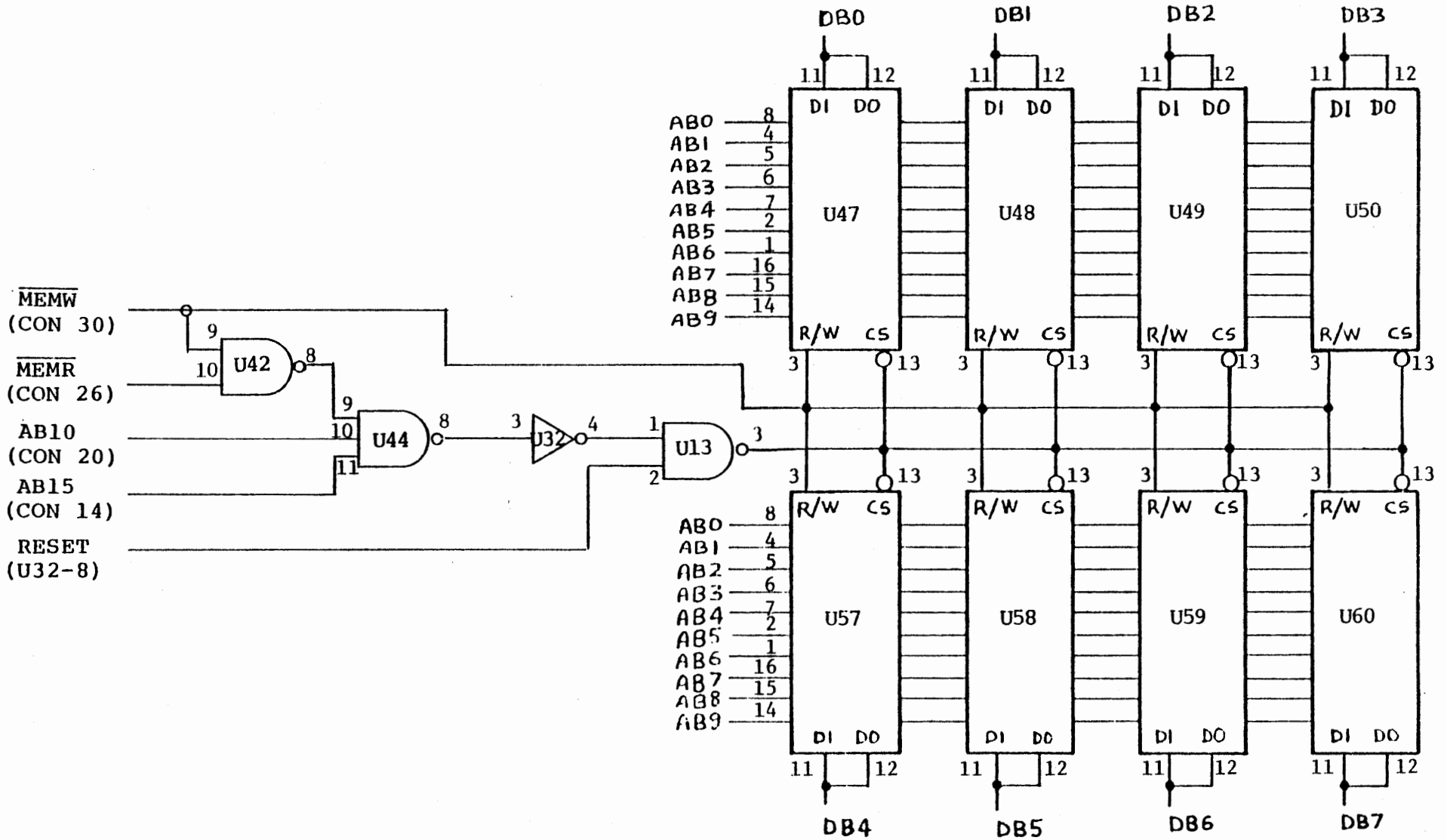
CONNECTOR PIN	SIGNAL NAME	CONNECTIONS ON INTERFACE BOARD
26	$\overline{\text{MEMR}}$	(U42-10)
27	$\overline{\text{RESET}}$	(U32-9) (U13-2)
28	P0B0	(U54-6)
29	P0C0	(U32-5)
30	$\overline{\text{MEMW}}$	(MEM-3) (U42-9)
31	$\overline{\text{INTA}}$	(U46-1) (U45-1)
32	DB7	(U45-9) (U15-2) (U60-11,12) (U26-1) (U28-7) (U30-7)
33	$\overline{\text{IOR}}$	(U28-5) (U30-5) (U26-22)
34	DB6	(U59-11,12) (U26-2) (U28-28) (U30-28) (U45-7)
35	DB5	(U58-11,12) (U26-3) (U28-29) (U30-29) (U45-5)
36	DB4	(U57-11,12) (U26-4) (U28-30) (U30-30) (U45-3)
37	DB3	(U50-11,12) (U26-5) (U28-31) (U30-31) (U46-9) (U14-3)
38	$\overline{\text{INTR}}$	(U33-3)
39	DB2	(U49-11,12) (U26-6) (U28-32) (U30-32) (U46-3) (U14-2)
40	DB1	(U48-11,12) (U26-7) (U28-33) (U30-33) (U46-5) (U14-1)
41	INTC	GND
42	DB0	(U47-11,12) (U26-8) (U28-34) (U30-34) (U46-7)
43	$\overline{\text{IOW}}$	(U28-36) (U30-36) (U26-23) (U15-3)

NOTE: (MEM -) refers to corresponding pins on all memory chips: U47,U48,U49,U50,U57,U58,U59,U60.

LIST OF INTERFACE SIGNALS TO MTS

FIGURE 1-3b

8x2102A-4



ADDED MEMORY (8400 - 87FF)

FIGURE 1-4

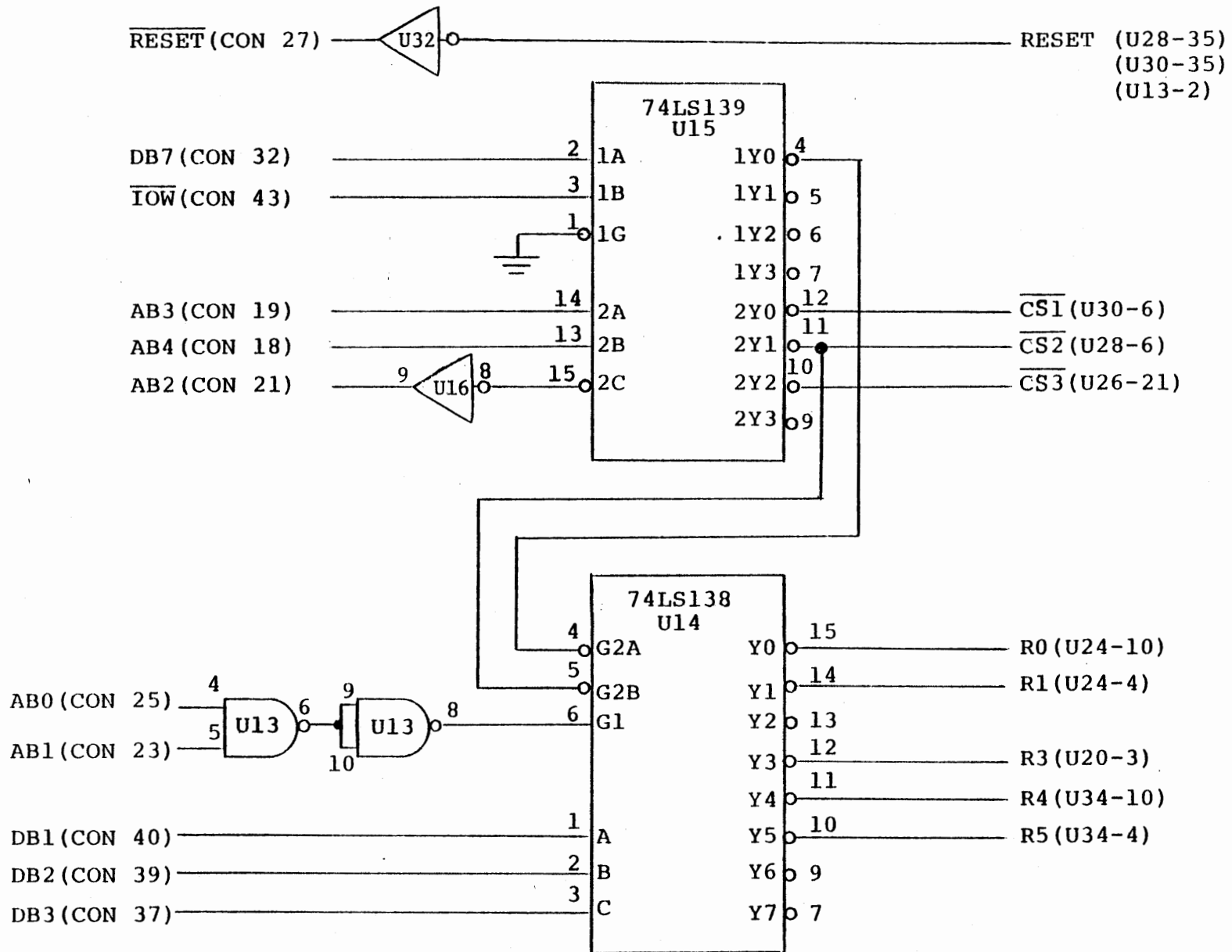
4.2 Added Memory

The interface circuit board provides space for the addition of 1024 bytes of memory, located at addresses 8400-87FF. To use this memory area install eight 2102 or 8102 memory chips. Each chip is organized as 1024 locations of one bit (1024 x 1), so the full 1024 bytes must be installed. Figure 1-4 shows the memory connections. This figure also exemplifies the notation used in other schematics. For example, consider the two signals shown at the lower left, AB 15 and RESET. AB15 is "address bus bit 15". It comes from the ribbon cable at pin 14, designated (CON14). The signal is connected as shown on this schematic to input pin 11 of a three input gate which is part of the chip designated U44 on the ITS board. You will find this chip in the second column from the right on the circuit board.

RESET is a signal generated on the interface board. Its source is an inverter in chip U32, with the output at pin 8. It is connected to input pin 2 of a gate which is part of chip U13.

1.4.3 Chip Selects and Resets

Several bits of the address bus, control bus, and data bus are decoded to generate various chip select and reset signals needed on the interface board. The circuits are shown in Figure 1-5. Figure 1-6 gives a "truth table" listing the results of this decoding circuitry. The addressing of the interface board ports is described in Section 2.1. The purpose and use of the various reset signals are described in Section 2.7.



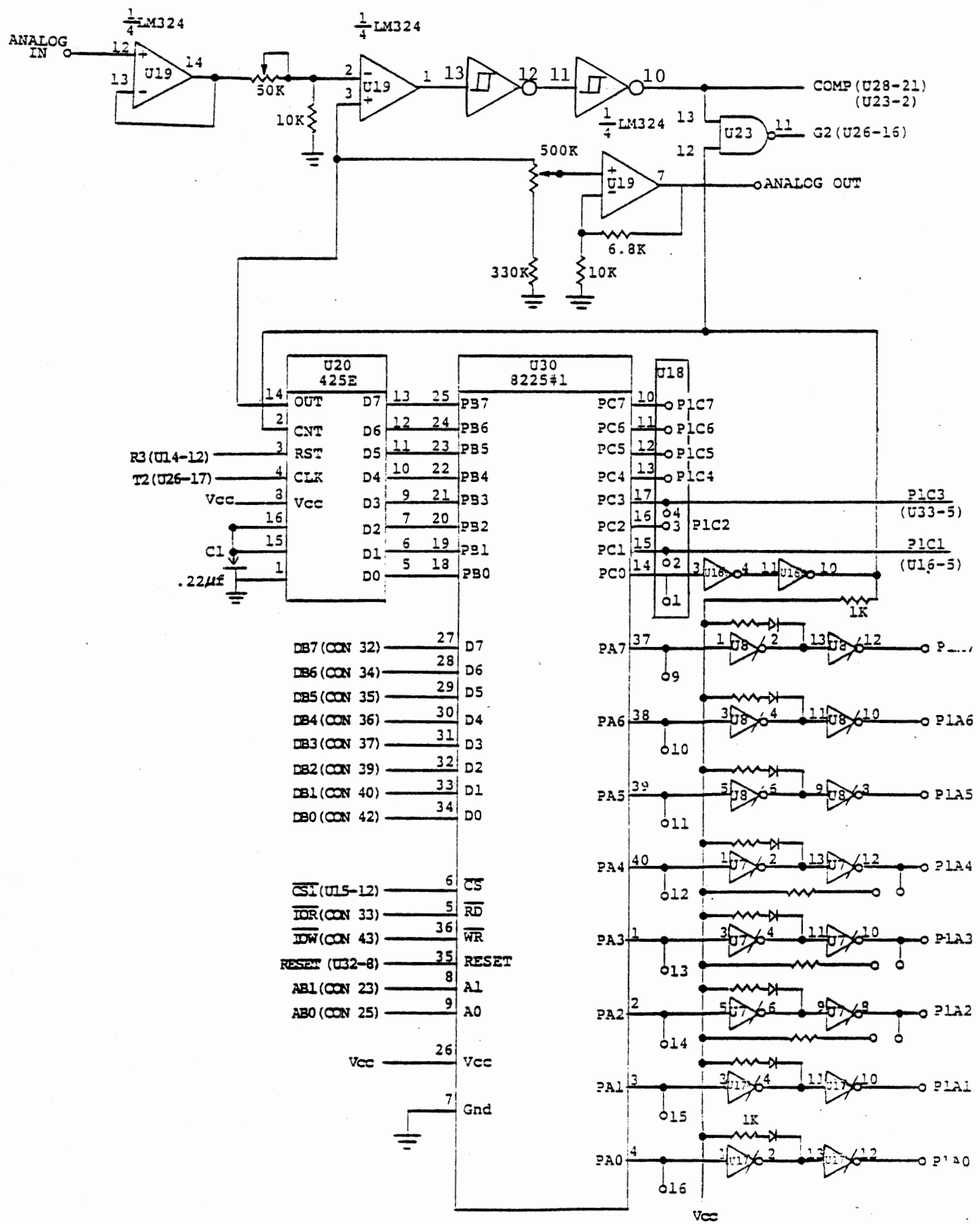
I/O CHIP SELECTS AND INTERRUPT FLIP-FLOP RESETS

FIGURE 1-5

\overline{IOW}	Low Address Bus								Data Bus								Chip Select			Resets					
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	3	2	1	5	4	3	1	0	
X	X	X	X	X	X	0	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	No effect
X	X	X	X	1	0	1	X	X	X	X	X	X	X	X	X	X	0	1	1	1	1	1	1	1	Select Timer
X	X	X	X	0	0	1	X	X	X	X	X	X	X	X	X	X	1	1	0	1	1	1	1	1	Select Port 1
1	X	X	X	0	1	1	X	X	X	X	X	X	X	X	X	X	1	0	1	1	1	1	1	1	Select Port 2
0	X	X	X	0	1	1	0	0	X	X	X	X	X	X	X	X	1	0	1	1	1	1	1	1	Write Port 2A
0	X	X	X	0	1	1	0	1	X	X	X	X	X	X	X	X	1	0	1	1	1	1	1	1	Write Port 2B
0	X	X	X	0	1	1	1	0	X	X	X	X	X	X	X	X	1	0	1	1	1	1	1	1	Write Port 2C
0	X	X	X	0	1	1	1	1	1	X	X	X	X	X	X	X	1	0	1	1	1	1	1	1	Program Port 2
0	X	X	X	0	1	1	1	1	0	X	X	X	0	0	0	X	1	0	1	1	1	1	1	0	Bit Set/Reset 2C0
0	X	X	X	0	1	1	1	1	0	X	X	X	0	0	1	X	1	0	1	1	1	1	0	1	Bit Set/Reset 2C1
0	X	X	X	0	1	1	1	1	0	X	X	X	0	1	0	X	1	0	1	1	1	1	1	1	Bit Set/Reset 2C2
0	X	X	X	0	1	1	1	1	0	X	X	X	0	1	1	X	1	0	1	1	1	0	1	1	Bit Set/Reset 2C3
0	X	X	X	0	1	1	1	1	0	X	X	X	1	0	0	X	1	0	1	1	0	1	1	1	Bit Set/Reset 2C4
0	X	X	X	0	1	1	1	1	0	X	X	X	1	0	1	X	1	0	1	0	1	1	1	1	Bit Set/Reset 2C5
0	X	X	X	0	1	1	1	1	0	X	X	X	1	1	0	X	1	0	1	1	1	1	1	1	Bit Set/Reset 2C6
0	X	X	X	0	1	1	1	1	0	X	X	X	1	1	1	X	1	0	1	1	1	1	1	1	Bit Set/Reset 2C7

Truth Table for Chip Selects and Resets

FIGURE 1-6



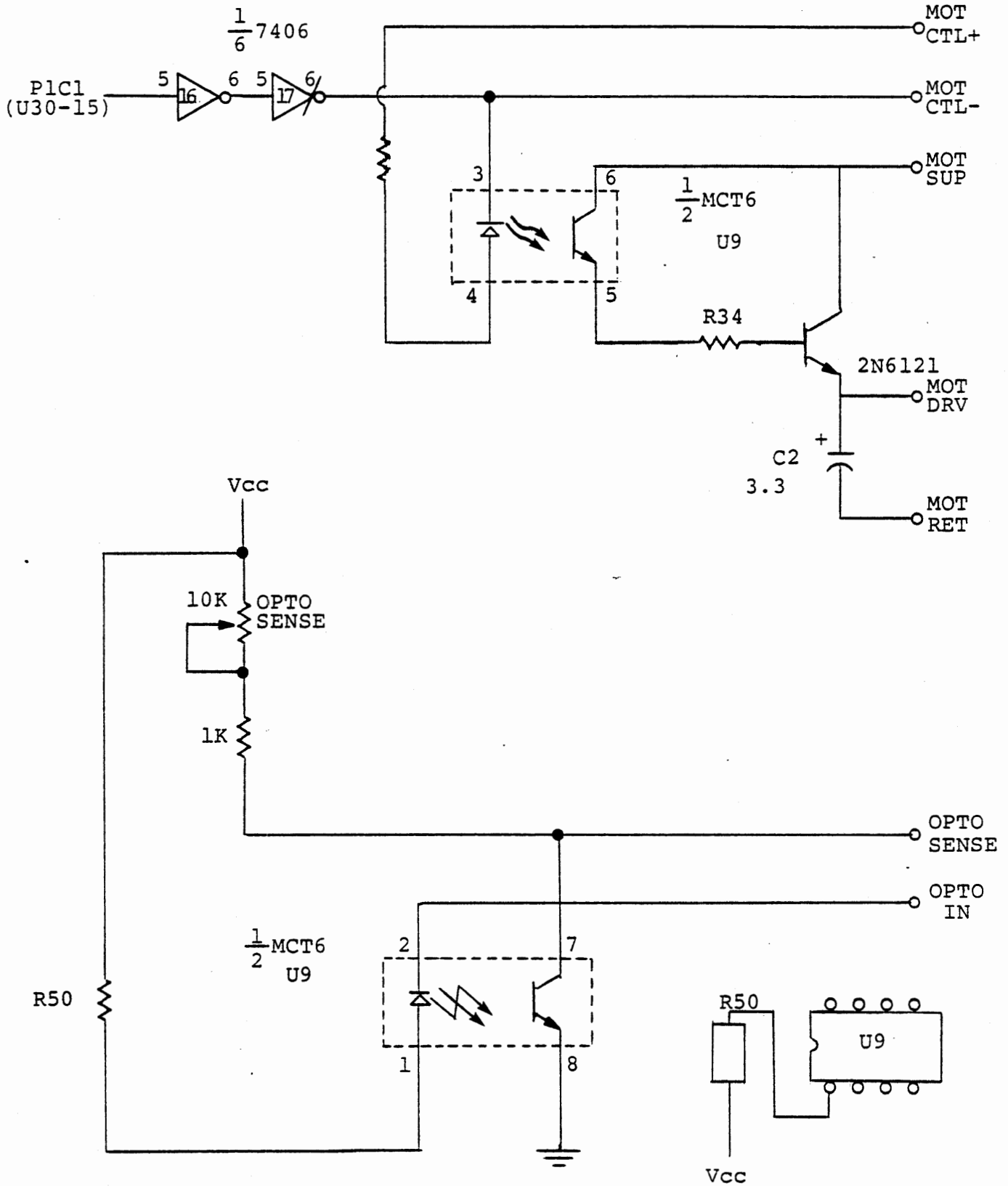
PORT1 AND A/D-D/A CONVERTER
FIGURE 1-7

1.4.4 Port 1 and A/D - D/A Converter

The interface board includes two 8255 I/O devices, with 3 I/O Ports each (A, B & C). One of these designated device (or Port) 1, drives the LED indicators at the top left of the interface board via Port 1A and provides connections for the digital to analog converter via Port 1B. It is shown in Figure 1-7. The discrete outputs are described in Sections 2.1 through 2.3. The digital to analog converter is described in Sections 4.5 and 4.6. The circuitry that makes it function as an analog to digital converter for input is the subject of Sections 5.3 and 5.4. Port 1C is brought to a DIP socket (U18) for general I/O applications.

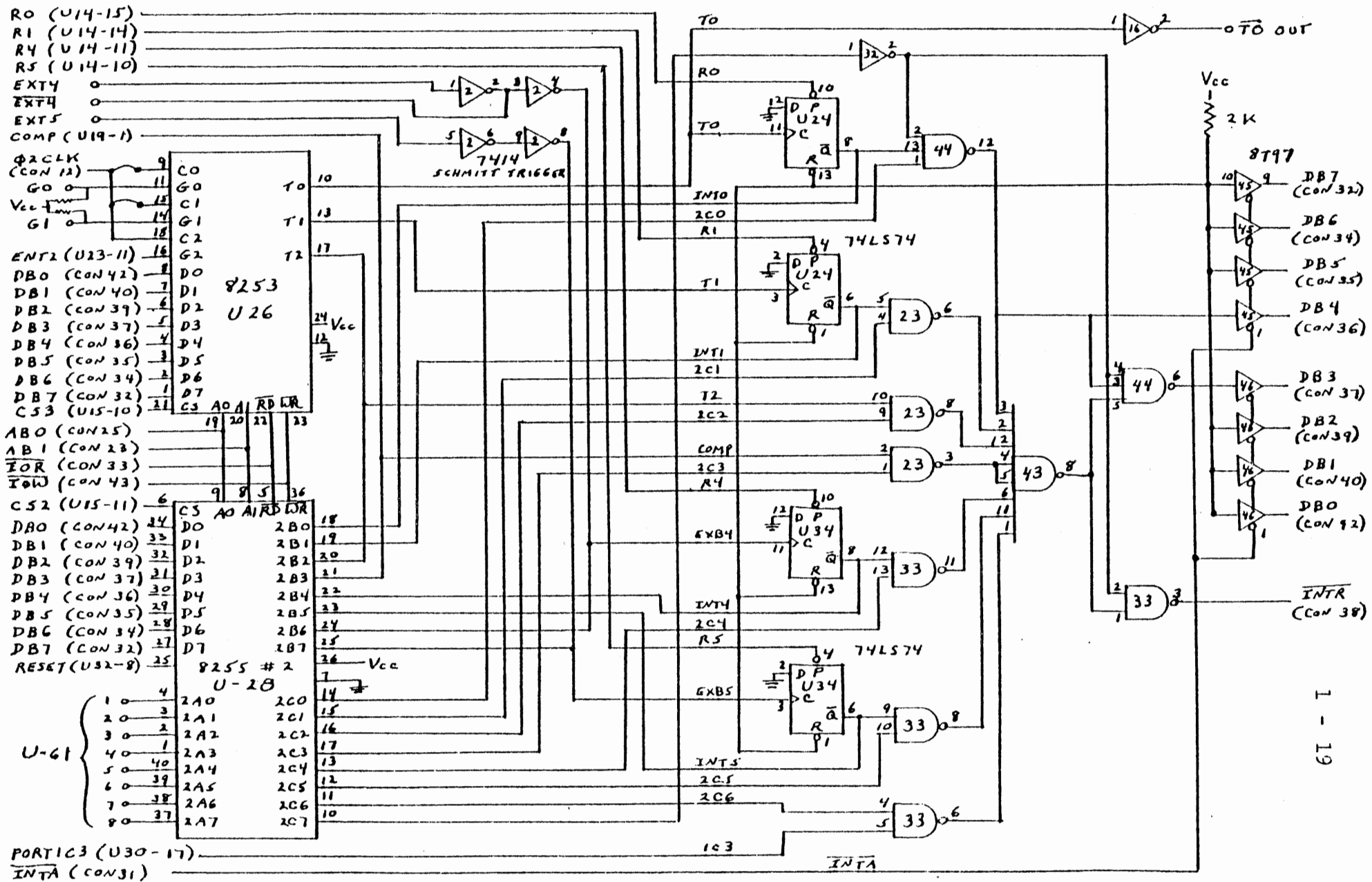
1.4.5 Interrupt System

The second 8255 I/O device is primarily devoted to the interrupt system shown in Figure 1-8 and described in Sections 2.7 through 2.9. The 8253 interval timer is closely tied to the interrupt system, so it is shown in the same schematic, but this device is so important (and so complex) that all of Chapter 3 is devoted to it.



OPTICAL COUPLERS AND POWER DRIVER

FIGURE 1-9



VECTORED PRIORITY INTERRUPT SYSTEM

FIGURE 1-8

1.4.6 Optical Couplers and Power Driver

It is often necessary to provide electrical isolation between the computer and external equipment. Optical couplers use infrared light as a coupling medium for information while giving complete electrical isolation. A optical coupler (Monsanto MCT6) is provided on the circuit board. One coupler is used for output, driving a power transistor mounted on a heat sink. The other coupler is used for input. Figure 1-9 shows these circuits.

1.4.7 Serial Interface Circuit

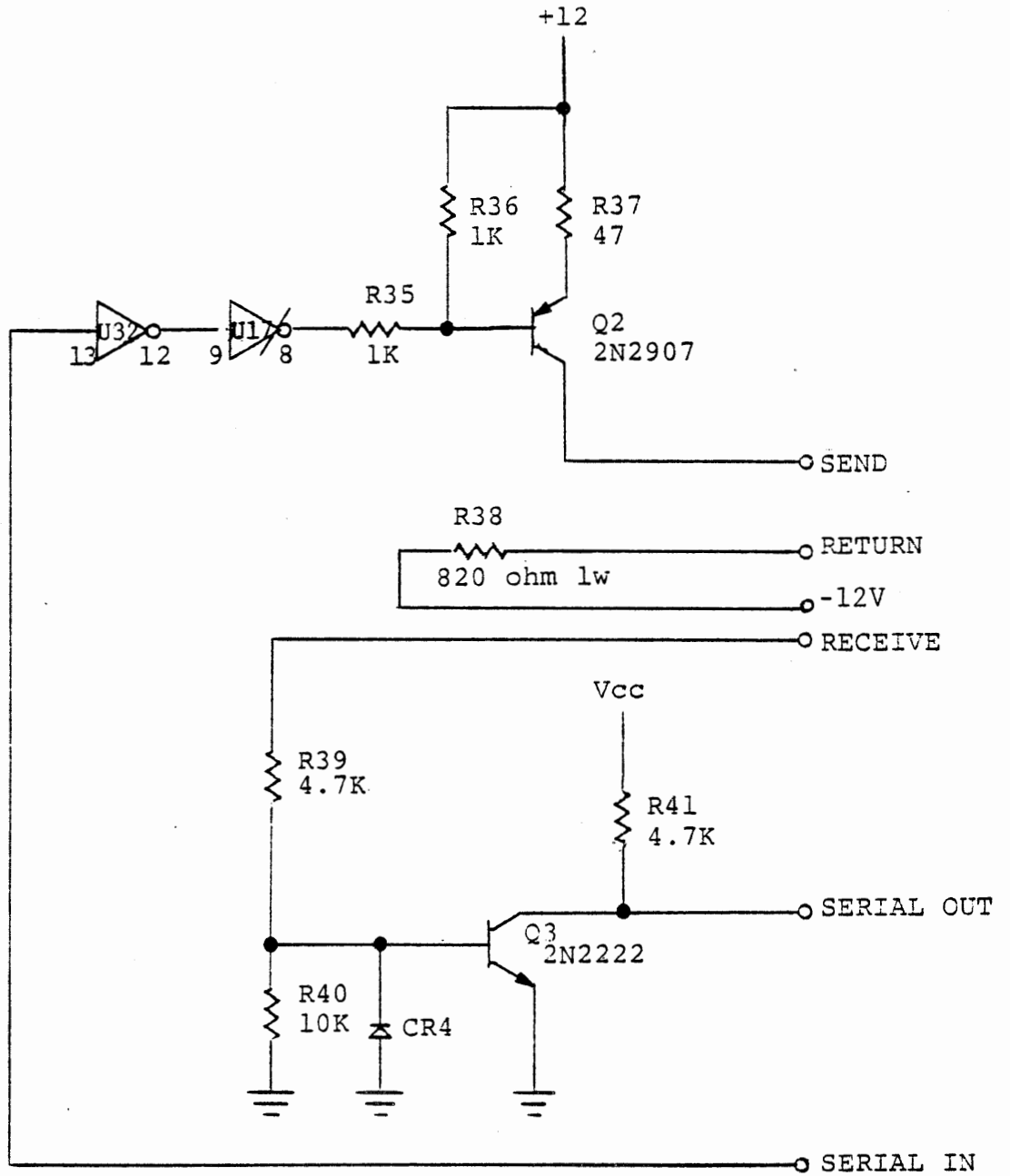
The circuits shown in Figure 1-10 can be used to connect the MTS to a teletype or a terminal such as a CRT that uses RS232 signal levels. The RS232c system's software and board connections are described in Appendix E.

1.4.8 Tape Cassette Modem

Programs (and data) can be stored on or loaded from magnetic tape cassettes, using the digital modem shown in Figure 1-11. This is to be connected to a consumer type audio cassette recorder through the screw terminals labelled AUX and EAR, (located at upper right of ITS board) with a common ground. The monitor program of the MTS provides for recording and loading programs. The programs and connections are described in Appendix D.

1.4.9 Tape Cassette Library

A cassette tape is provided with most of the programming exercise solutions and a few additional programs. It is described in Appendix D.

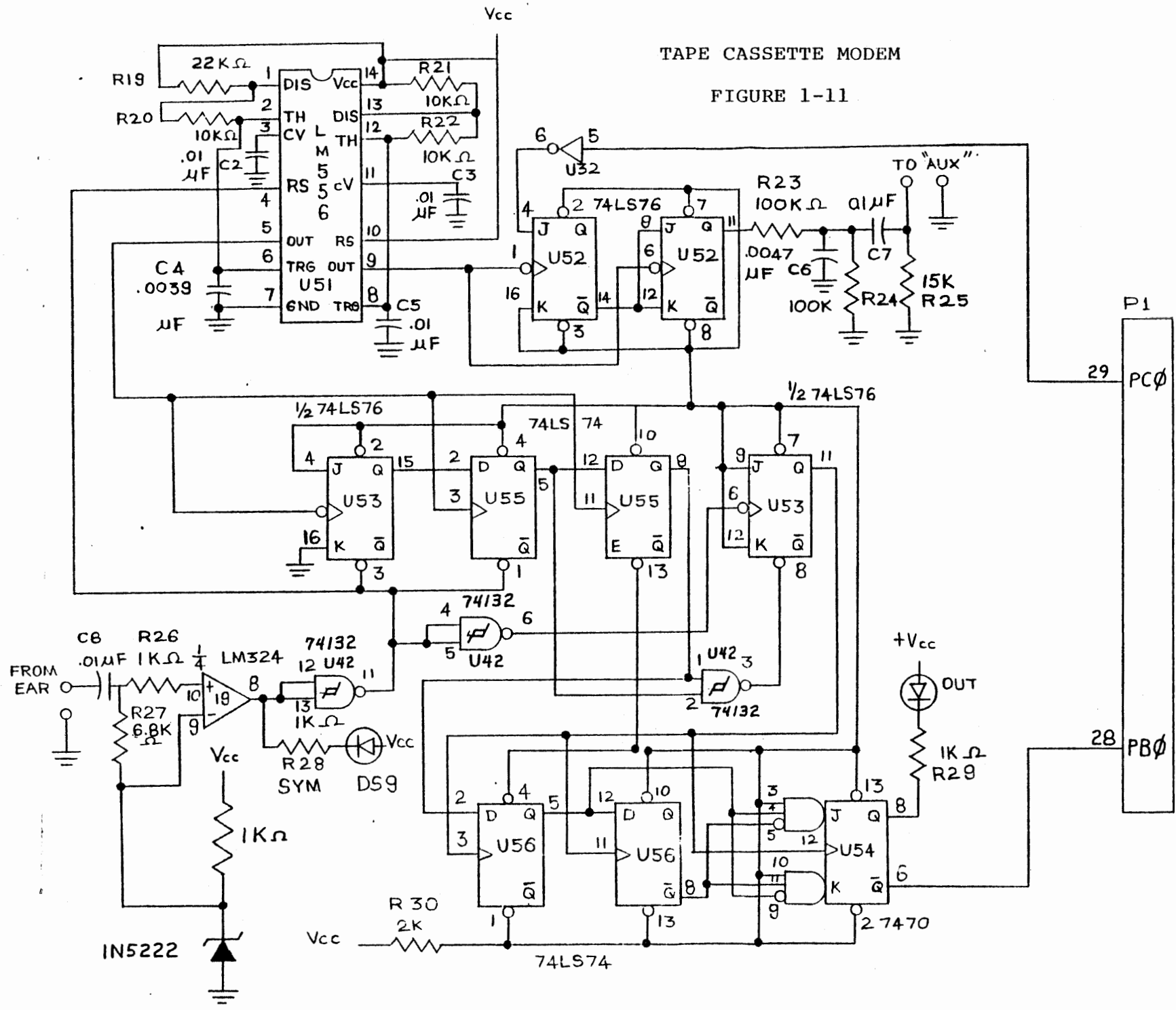


SERIAL INTERFACE CIRCUIT

FIGURE 1-10

TAPE CASSETTE MODEM

FIGURE 1-11



MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 2

INPUT/OUTPUT AND INTERRUPTS

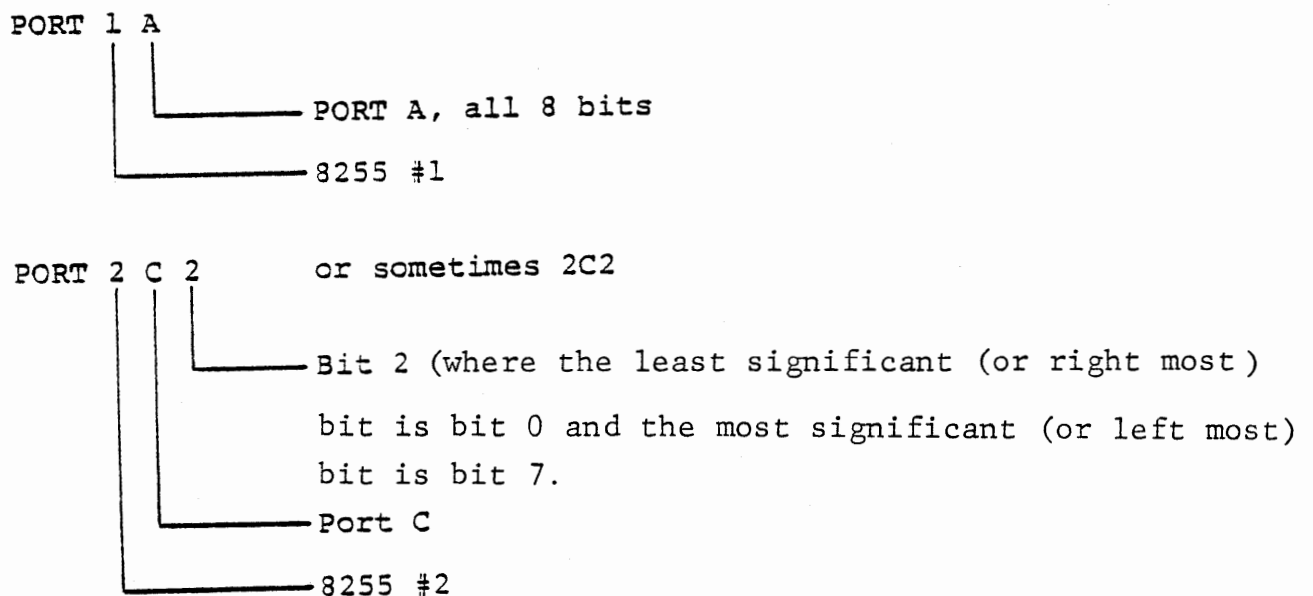
2.0 INPUT/OUTPUT AND INTERRUPTS

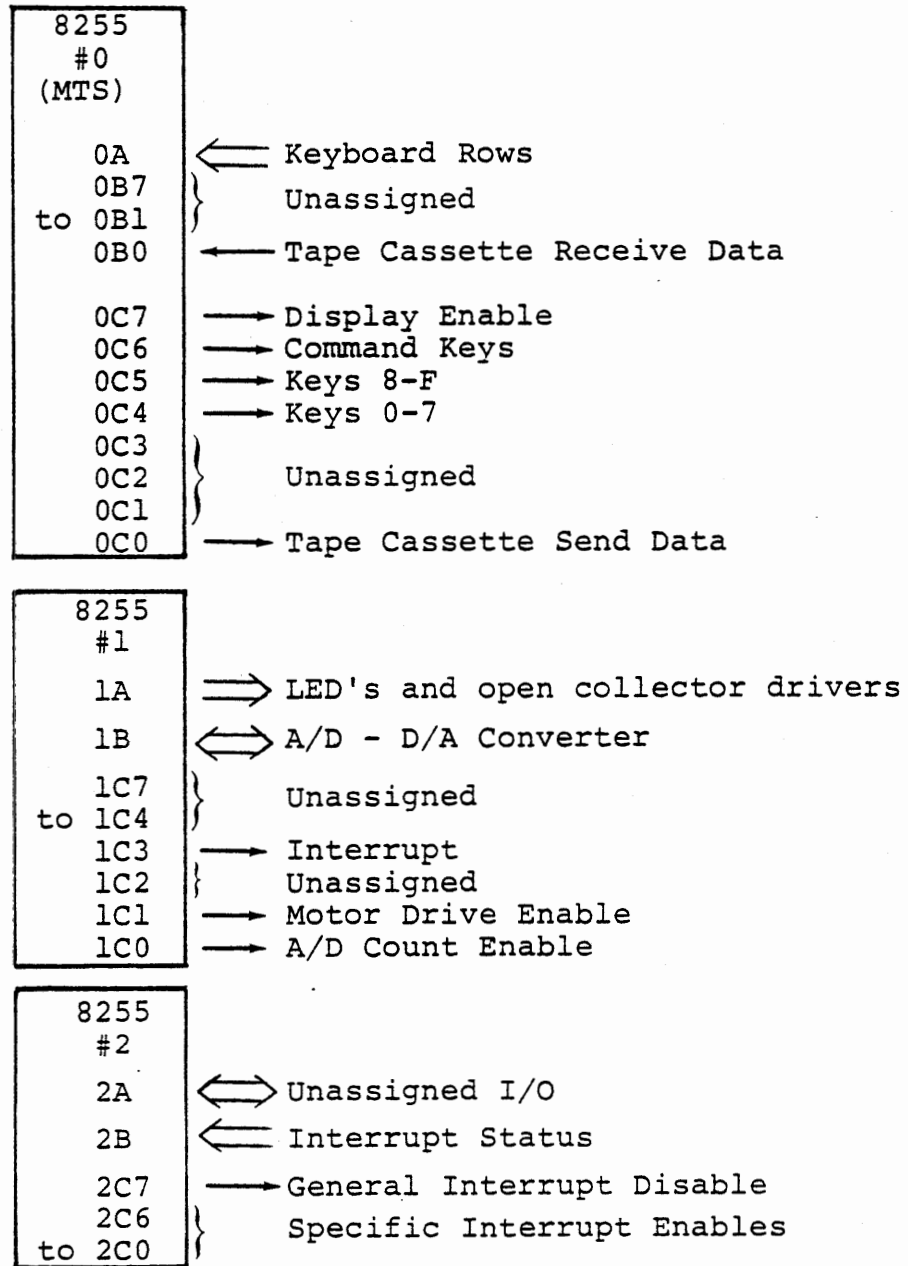
This chapter discusses the provisions made on the experiment board for digital logic level inputs and outputs to the microprocessor. Interval timers, analog signals and optically isolated inputs are discussed in later chapters.

2.1 PORT ASSIGNMENTS AND ADDRESSES

The experiment board includes two 8255 Programmable Peripheral Interface devices. Including the 8255 on the MTS board, a total of 72 bits of input/output is accessible to the 8080 microprocessor. In addition there is an Intel 8253 Interval Timer (see Chapter 3) which is addressed and programmed in much the same way as the 8255 ports. Figure 2-1 shows the port assignments. Figure 2-2 lists the port addresses and assignments and gives a list of programming control bytes suitable for each of the 8255's.

In this table and throughout the course we will refer to input ports by device number, port letter, and sometimes a bit number.





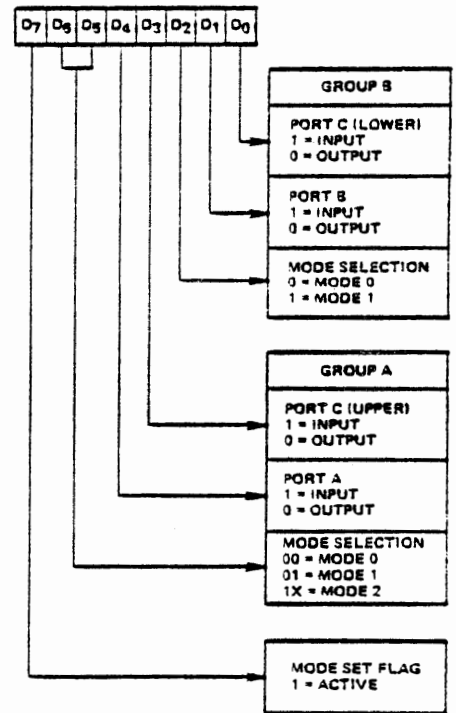
8255 I/O PORT ASSIGNMENTS
Figure 2-1

PORT ADDRESSES AND ASSIGNMENTS

ADDRESS	PORT NAME	FUNCTION	SPECIAL ASSIGNMENTS FOR 0C AND 1C		
00	PORT 0A	MTS Keyboard Input	0C7	Display Control	(1 = On)
01	PORT 0B	Unassigned except 0B0	0C6	Enable Command Keys	(0 = On)
02	PORT 0C	See column at right	0C5	Enable Keys 8-F	(0 = On)
03	CNT 0	Control Port for MTS 8255	0C4	Enable Keys 0-7	(0 = On)
04	PORT 1A	LED and Driver Outputs	0C3	Unassigned	
05	PORT 1B	D/A Output or A/D Input	0C2	Unassigned	
06	PORT 1C	See column at right	0C1	Unassigned	
07	CNT 1	Control Port for 8255 # 1	0C0	Cassette Modem Out	
0C	PORT 2A	Unassigned	0B0	Cassette Modem In	
0D	PORT 2B	Interrupt Status Input	1C7-4	Unassigned	
0E	PORT 2C	Interrupt Enable Output	1C3	Interrupt (If Enabled by 2C6)	
0F	CNT 2	Control Port for 8255 # 2	1C2	Unassigned	
14	TIM 0	Timer 0	1C1	Motor Drive Buffer	(1 = On)
15	TIM 1	Timer 1	1C0	D/A Control (1 = Automatic A/D)	
16	TIM 2	Timer 2			
17	TIM CT	Control Port for 8253			

8255 PROGRAMMING CONTROL BYTES (WRITE TO 8255 CONTROL PORT)

CONTROL BYTE	PORT A	PORT B	PORT C0-C3	PORT C4-C7	USE WITH 8255 #		
					0	1	2
80	Out	Out	Out	Out		D/A	
81	Out	Out	In	Out		●	
82	Out	In	Out	Out		A/D	*
83	Out	In	In	Out		A/D	
88	Out	Out	Out	In		D/A	
89	Out	Out	In	In		●	
8A	Out	In	Out	In		A/D	
8B	Out	In	In	In		A/D	
90	In	Out	Out	Out	*	D/A	
91	In	Out	In	Out	*	●	
92	In	In	Out	Out	*	A/D	*
93	In	In	In	Out	*	A/D	
98	In	Out	Out	In		D/A	
99	In	Out	In	In		●	
9A	In	In	Out	In		A/D	
9B	In	In	In	In		A/D	



* Generally only these control bytes should be used for normal operation.
 ● Forbidden configurations

Mode Definition Format

Figure 2-2

2.2 PROGRAMMING AND USING THE 8255

At system reset all ports of all 8255's are automatically set to input Mode 0. They can be used this way or programmed to other configurations by writing a control byte to the control port of the desired 8255. The monitor program automatically re-configures the 8255 on the MTS board such that ports 0A and 0B are input and 0C is output. It accomplishes this by writing 92 to the control register.

```
3E    MVI A,92          Load A with control byte to make device
92                                0: A = IN; B = IN; C = OUT
D3    OUT CNT           Output A to 8255 0 Control Register
                                (Address 03, with most significant 5 bits
FB                                high: 11111011B.
```

The experiment board 8255's must be set to the desired modes by your program. The first programs we will develop require output in all three ports of 8255 #1. Figure 2-2 gives 80 as the required control byte:

```
3E    MVI A,80          Load A with control byte to make device 1:
80                                A = OUT, B = OUT, C = OUT
D3    OUT CNT1         Set 8255 1 Control Register
07                                (Address 07)
```

Because 8255 2 is largely committed to the interrupt system it usually is programmed for input at port 2B and output at port 2C. Port 2A may be input or output but must be in Mode 0, the normal

direct I/O mode. Most programs developed in this course use the interrupt system, so in general programs should contain (again from Figure 2-2):

```
3E    MVI A,93      (or 82) Device 2: A out, B in, C out
92
D3    OUT CNT2
0F
```

With the 8255 ports programmed, data can be read from or written to the ports by IN or OUT instructions, for example:

```
DB    IN  PORT0A      (A) ← Port 0A
00                                (Keyboard Input)
D3    OUT PORT1A      Port 1A ← (A)
04                                (EXP. Board's LED's)
```

If you write a program containing all of the instructions listed so far, and terminated with a jump back to the input instruction, as in Figure 2-3, the LED indicators on the experiment board will show the keyboard input data.

Figure 2-4 shows the keyboard connections to ports 0C and 0A.

Programming a port to output automatically sets its outputs low.

Therefore, (from Figure 2-4), if a key is pressed, the corresponding bit in port 0A is made low. For example, if the MEM key is depressed, then port 0A0 (which was pulled high by the resistor to V_{cc}) is now pulled low through the MEM key and port 0C6. (The monitor scans the keyboard by alternately making ports 0C4, 0C5 and 0C6 low or 0, thus indentifying the key pressed).

With the program shown, Port 0C is programmed for output, so all keys are enabled (0C4, 0C5 and 0C6 all low), therefore, 0, 8 and MEM will show the same output. The input to the bit of port 0A is high if no key in that column is pressed; if a key is pressed and the bit of port 0C for that row is low, then the input is low. (Review Course 525, Sections 8.1.3 through 8.1.6 if a more detailed description is needed).

PROGRAMMING THE 8255's

2-1
c/7/78

A D D R CODE

CODING SHEET

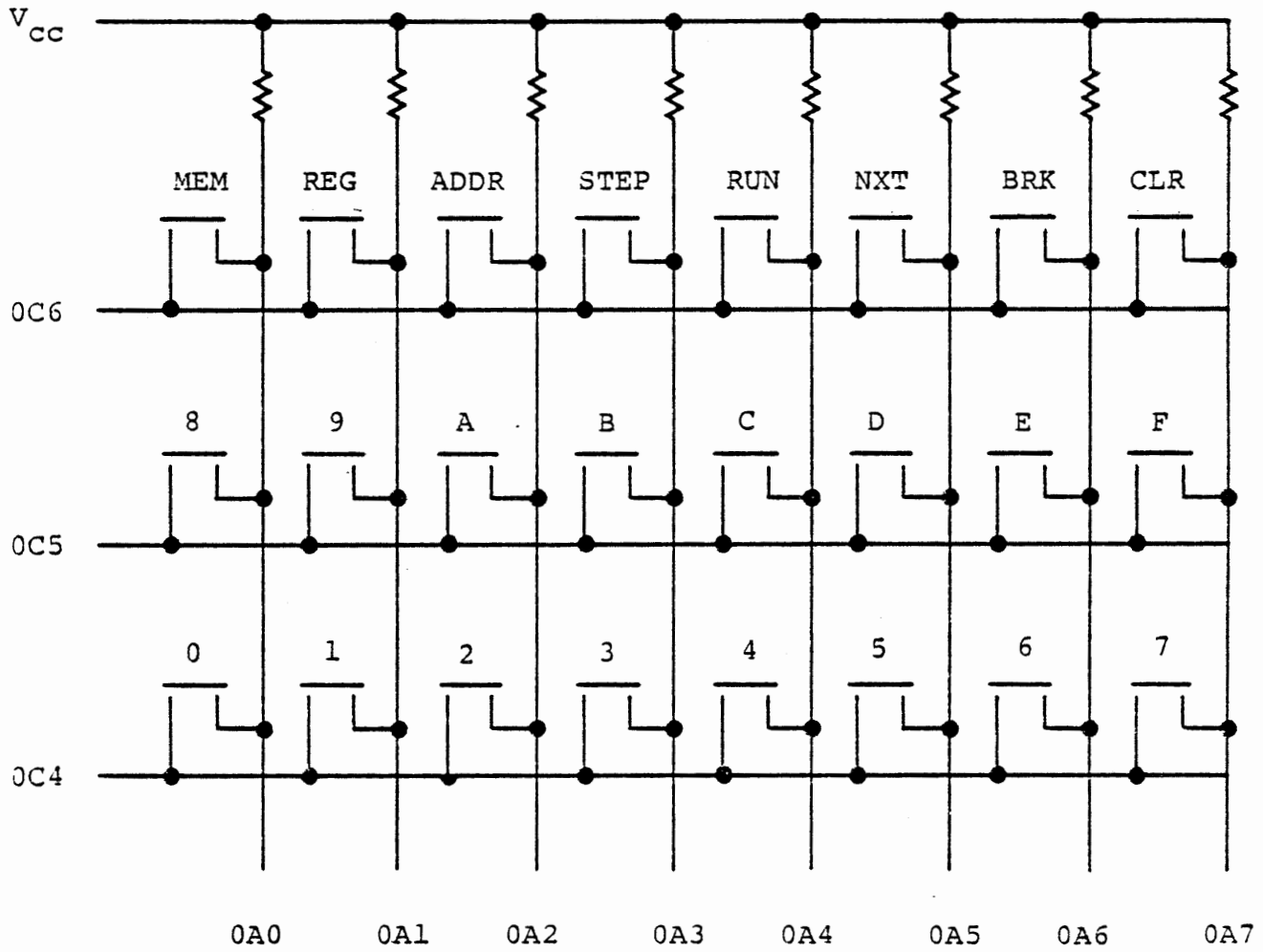
MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	20	0	3E	MVI	A, 92	Program MTS 8255
		1	92			A in B in Cout
		2	D3	OUT	CNT0	
		3	03			
		4	3E	MVI	A, 80	Program 8255 #1
		5	80			A out Bout Cout
		6	D3	OUT	CNT1	
		7	07			
		8	3E	MVI	A, 92	Program 8255 #2
		9	92			A in B in Cout
	A		D3	OUT	CNT2	
	B		0F			
	820	C	DB	IN	PORT0A	Read keyboard
		D	00			
		E	D3	OUT	PORT1A	Display in LED's
		F	04			
8	21	0	C3	JMP	820C	Back to input
		1	0C			
		2	82			
		3				
		4				
		5				
		6				
		7				
		8				
		9				
	A					
	B					
	C					
	D					
	E					
	F					
8		0				
		1				
		2				
		3				
		4				
		5				
		6				
		7				
		8				

Figure 2-3

This Page Intentionally Left Blank

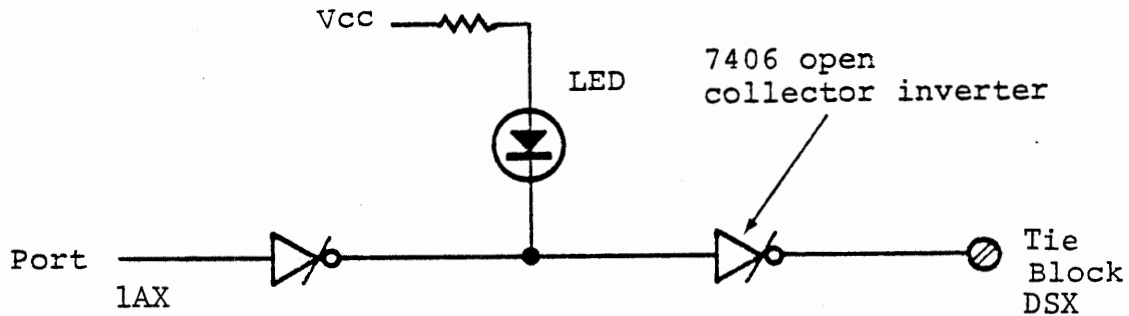


MTS KEYBOARD CONFIGURATION
AND PORT ASSIGNMENT

Figure 2-4

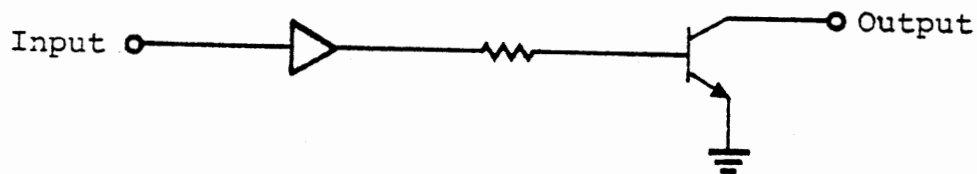
2.3 PORT 1A LED'S AND DRIVERS

Port 1A (address 04) drives eight sets of open collector inverters and LED's.

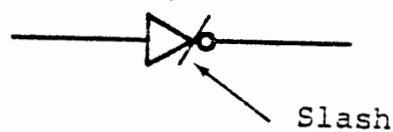


Each bit of the port drives an identical circuit. The LED indicates the state of the output, i.e. illuminated if a one is output. The terminal block output follows the port. The state is low if a zero is output and open if one is output.

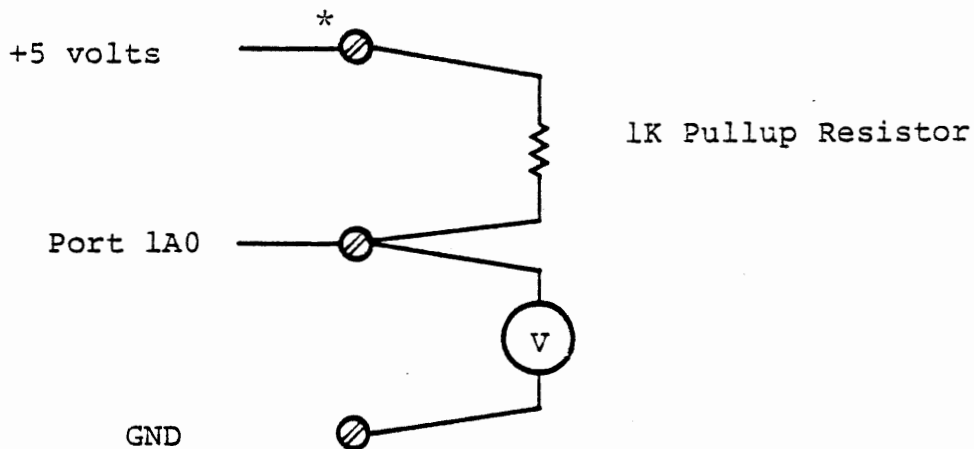
An open collector buffer is a TTL amplifier whose output comes from a transistor with no internal connection to its collector. It is approximately equivalent to the circuit below. When the input is a logic 1 (>2v.) current flows into the transistor, thus turning it on and effectively connecting the output to ground. However, when the input is a logic 0 (0 v.), no current flows into the transistor's base. Therefore, it is off and the output is "floating" (i.e. connected to neither ground nor +5v.).




An open collector inverter is shown on a schematic diagram by:



(The slash indicates open collector). Note that the open collector output gives a signal only if it is pulled up through some load or pullup resistor to a positive voltage, which may be as high as 30 volts. The output is capable of sinking 40 ma to 0.7 volts. Connect a voltmeter from one of the port 1A output drivers to ground. It will show 0 volts whether the LED is on or off. Now connect a pullup resistor to +5 volts, and the voltmeter will display either 0V or 5V depending on the state of the Port 1A0 output bit.



For output bits 1A2, 1A3, and 1A4 (marked DS2, DS3, and DS4 on the ITS board) pullup resistors (in U1) are available on the circuit board, but not connected. They can be connected by soldering jumpers between two pads in front of the LED's for those three bits.

* NOTE Throughout this text the illustrations represent ITS board TIE Block connect points with the symbol . These refer to one of the labelled rows within the white Tie blocks.

2.4 MTS DISPLAY

The seven segment displays on the MTS are operated by a direct memory access system. Whatever data are written to memory locations 83F8 through 83FF are automatically displayed in the eight digits. (Review Course 525, Sections 8.3 through 8.3.1 for more detail). The DMA channel must be enabled by a high output at port 0C7. Since programming a port to output automatically sets all bits low, the display was disabled when you programmed the MTS 8255 0. Prove this by adding STA 83F8 before the jump instruction in your program. Even though you have written the same data to the DMA display area of the MTS memory as you wrote to the LED's, the display will remain blank.

A good way to turn the display on is by use of the bit set/reset function of the 8255. This allows a single bit of port C to be changed without affecting any other bit. Enter this at the end of your program (after OUT PORT1A):

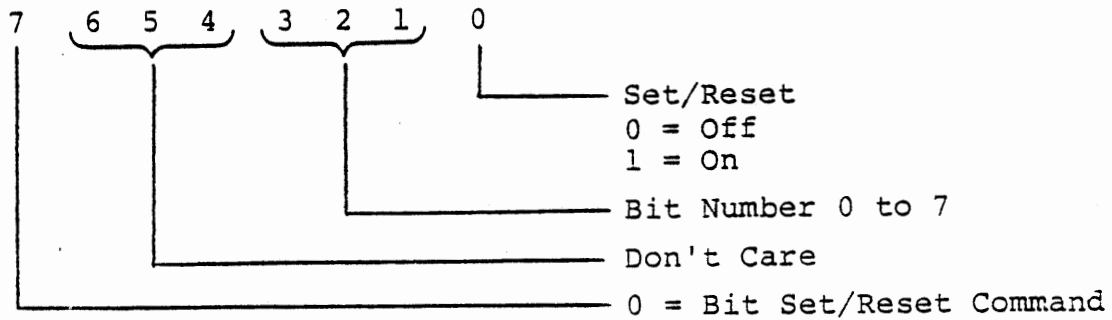
```

32   STA      83F8      Write keyboard data to display
F8
83
3E   MVI      A,0F      Set bit 7 in port 0C
0F
D3   OUT      CNT0
03
C3   JMP      820C      Jump back to input instruction
0C

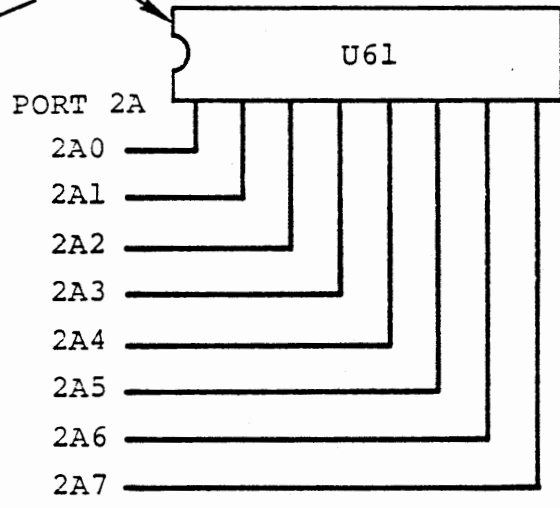
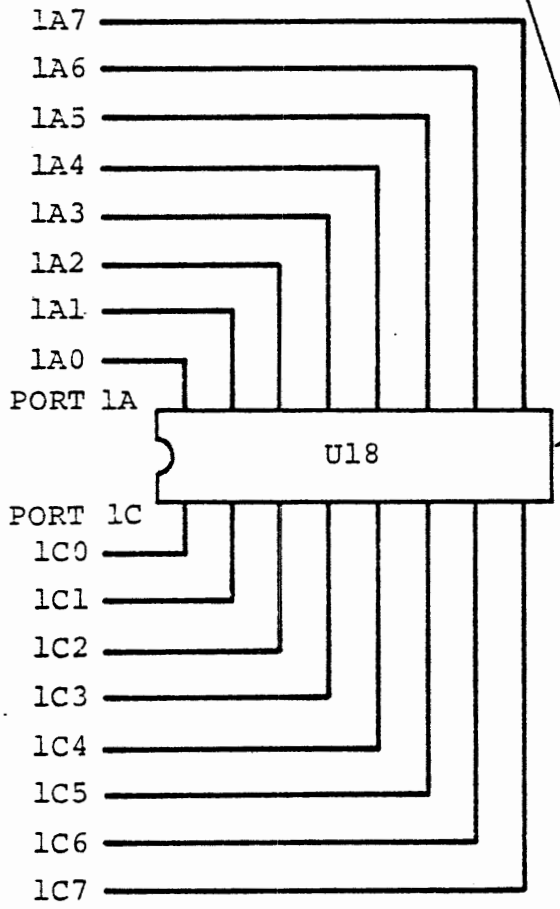
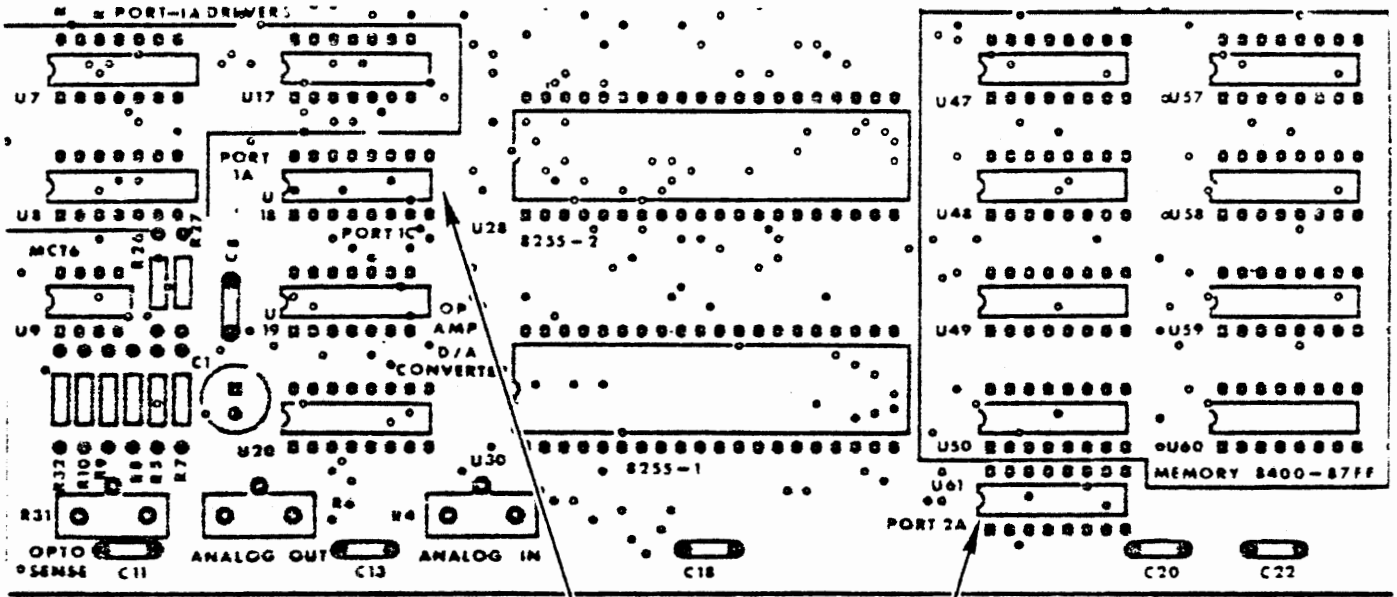
```

Note that the bit set/reset control byte is written to the control port, not to port C.

The bit set/reset control byte has the form:



We will use the bit set/reset command frequently. Be sure you understand it. The bit set/reset command is discussed in Course 525, Section 8.1.4.



INPUT/OUTPUT CONNECTIONS

Figure 2-5

2.5 INPUT/OUTPUT CONNECTIONS

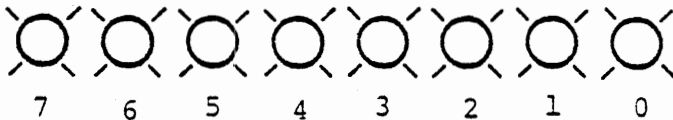
In addition to the terminal block outputs driven by the buffers, port 1A, port 1C, and port 2A are directly connected to empty DIP sockets, shown in Figure 2-5. A cable that plugs into this socket can be obtained (available from Augat as part number 7P16-3T24-1). This allows connection of these ports to any suitable device for input or output. None of the experiments described here require these connections, but when you develop interfaces to other equipment they may be needed.

2.6 EXTERNAL INPUTS 4 AND 5

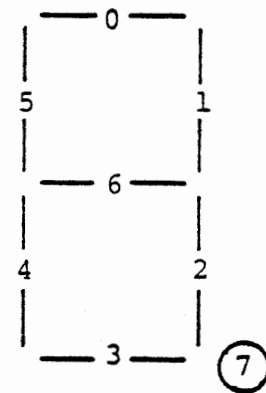
Two terminal block connections labelled EXT 4 and EXT 5 are provided for the external inputs needed in many of the experiments in this course. These inputs are part of the interrupt system, which will be described in Section 2.7. They can also be read as single input bits. They are connected to Port 2B, bits 6 and 7 respectively (Figure 2-7).

EXERCISE:

Read the external input bits and display them with the LED's. Change the program of Section 2.4 to read from Port 2B instead of from Port 0A. (Refer to Figure 2-2 to find the address). Since the EXT4 and EXT5 inputs are connected to Port 2B bits 6 and 7, the modified program will repeatedly input these bits and display them (along with random data in the other bit positions) in both the LED's and in the left digit of the MTS display. Connect a clip lead to the

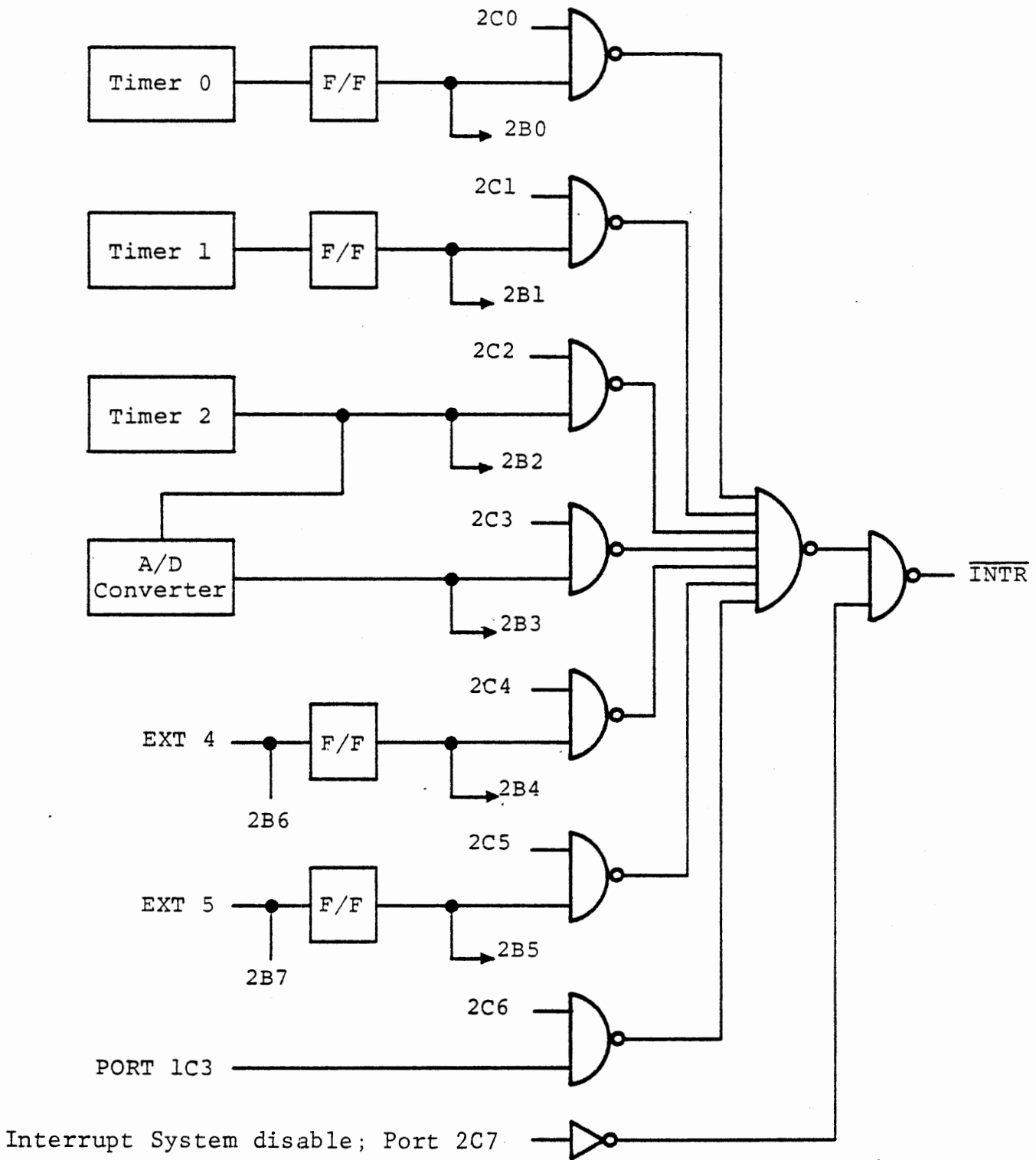


EXT4 input and touch it to ground to see bit 6 change in the display.



TTL circuits demand sharp rising and falling edges. To ensure suitable signals and to prevent spurious noise from causing interrupts, EXT4 and EXT5 are brought into Schmitt Trigger inverters (in chip U2, a 7414) (see Figure 2-7). The signals are then inverted again and used to clock flip-flops in the interrupt system, discussed in the

next section. The inverted signal, $\overline{\text{EXT4}}$, is available at a terminal labelled $\overline{\text{EXT4}}$ OUT. Connect this to EXT 5 IN, so that EXT 5 will be inverted from EXT 4. Now when you connect EXT 4 input to ground both signals will change. They will be displayed (with the program of Section 2.4) in bits 6 and 7 of the LED's.



INTERRUPT SYSTEM - PARTIAL DIAGRAM
Figure 2-6

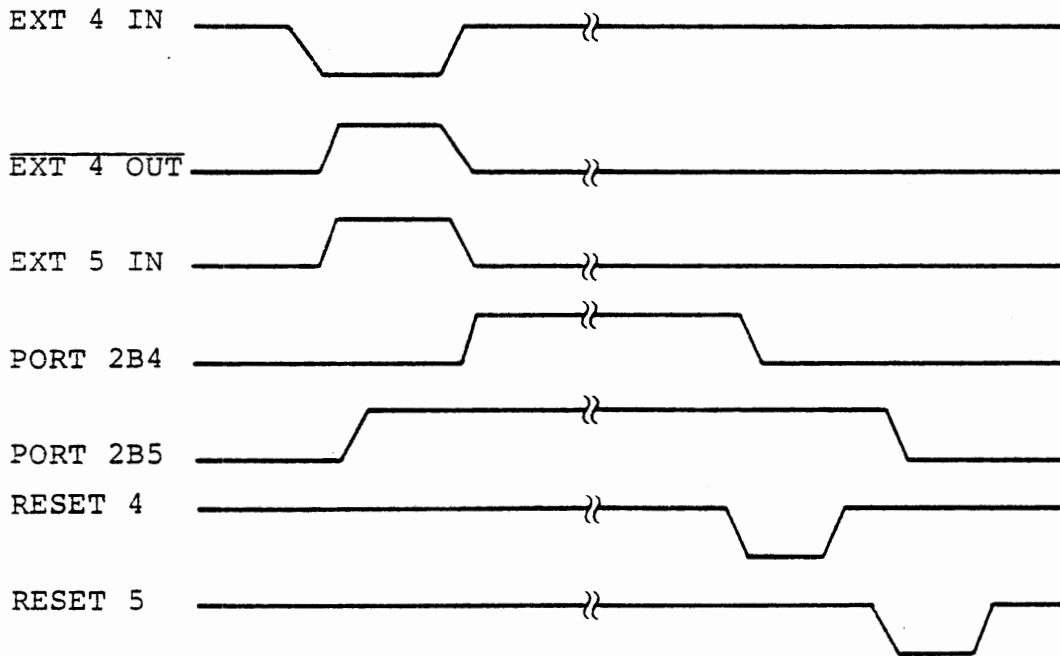
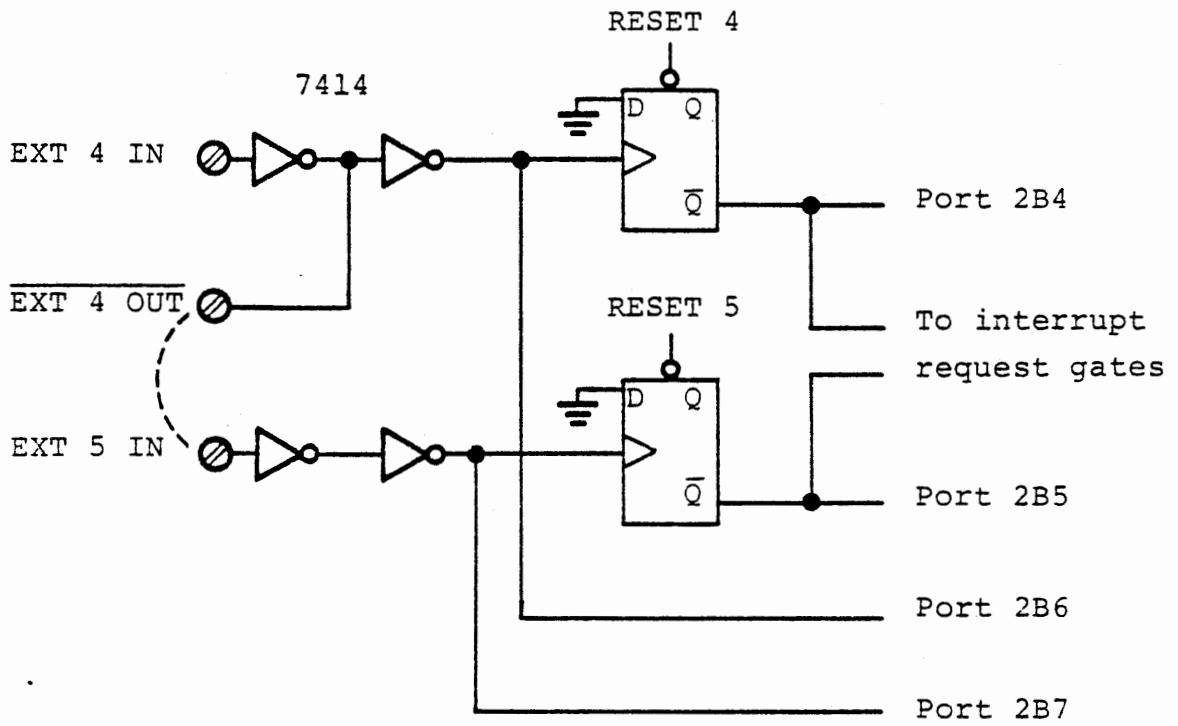
2.7 INTERRUPT FLIP-FLOPS AND ENABLES

Most of the experiments in this course use the interrupt capability of the 8080. The student should be familiar with Section 8.4 through 8.6 in Course 525 . A review of those sections may be advisable at this point.

2.7.1 Interrupt Sources

The MTS will accept repeated interrupts generated by its own hardware when the AUTO/STEP toggle switch is set to STEP. We will also refer to these as "monitor interrupts" since their purpose is to invoke monitor functions such as single stepping or breakpoints. In addition, and independent of the AUTO/STEP switch, the MTS will accept interrupts generated by the experiment board. (Once an interrupt has occurred, or if a DI instruction is executed in the program, no other interrupt can occur until an EI instruction and one following instruction have been executed).

Figure 2-6 is a partial diagram of the interrupt logic of the experiment board. Interrupts can occur in response to signals generated by the interval timers (Chapter 3), by strobed input or output using port 1C, and by the EXT 4 and EXT 5 input ports.



EXT 4 AND EXT 5 CONNECTIONS AND SIGNALS

Figure 2-7

2.7.2 Interrupt Flip-Flops

The purpose of an interrupt system is to provide for processing of an occasional and perhaps fleeting event while allowing the computer to carry on other tasks that can be put aside temporarily when an interrupt occurs. To permit the processor to recognize a possibly very brief signal, the rising edges of the EXT 4 and EXT 5 inputs set flip-flops, which actually provide the interrupt data to the processor. Each flip-flop is reset only by a specific command from the processor under program control. This insures that each interrupt signal is retained until it has been processed. Figure 2-7 shows the connections of these flip-flops in detail.

(To save components the experiment board uses negative logic here. The flip-flop is actually set to 0 by the input signal and preset to 1 by the reset command. Since the inverted output of the flip-flop is used, the signal becomes true (=1) at the input rising edge and false (=0) at reset.) These "reset" signals are software generated as described below.

Similar flip-flops are connected to the outputs of timer 0 and timer 1, to allow interrupts on narrow pulses from these timers. The interrupts from the A/D converter and port 1C3 are latched by their sources so flip-flops are not needed. Only timer 2 output is unlatched. It is used primarily in connection with the A/D converter or in a timing mode in which it latches its own input.

This Page Intentionally Left Blank

2.7.3 Interrupt Status and Enables

All of the interrupt sources, except port 1C3, are taken to port 2B as inputs. After an interrupt has occurred, the program can read this port to determine the source of the interrupt. We refer to the content of this port as the interrupt status byte. The program can, of course, read this port at any time. The data are not dependent on the interrupt. Refer again to Figure 2-6 to see these connections.

Although the hardware is designed to permit interrupts from many different sources, most programs will be concerned with only one or a few of these. To prevent any reaction to an undesired interrupt, each interrupt source is gated with an output bit from port 2C. The processor will be interrupted only if an event occurs and its enable bit at port 2C is set high. These gates also are shown in Figure 2-6.

The interrupt sources, their positions in the interrupt status byte at port 2B, and their enable bits from port 2C are listed on the next page.

Source	Interrupt Enable Bit (Active High)	Interrupt Status Bit
Timer 0 Flip-Flop	2C0	2B0
Timer 1 Flip-Flop	2C1	2B1
Timer 2 (no Flip-Flop)	2C2	2B2
A/D Comparator	2C3	2B3
EXT 4 Flip-Flop	2C4	2B4
EXT 5 Flip-Flop	2C5	2B5
Port 1C3	2C6	1C3
General Disable	2C7	
EXT 4 Direct (no interrupt)		2B6
EXT 5 Direct (no interrupt)		2B7

Note that Timer 0, Timer 1, EXT 4, and EXT 5 generate interrupts only through their flip-flops, which are set by rising edges. In processing the interrupt, the flip-flop will be reset (see section 2.7.4), so it will not generate new interrupts until another rising edge occurs.

Port 2C7 is a general disable for all external interrupts. When it is high, only monitor interrupts can occur. At system reset all ports are forced to input mode thus floating the signal lines. To the logic this appears as a high signal so port 2C7 automatically inhibits external interrupts. Whenever port 2C is programmed for output, all bits are automatically set low. Now, bits 2C0 through 2C6 inhibit the individual interrupt sources. No external interrupt can occur until 8255 2 has been programmed and specific bits of 2C have been set high.

2.7.4 Clearing Interrupts

The interrupt flip-flops for Timer 0, Timer 1, EXT 4, and EXT 5 are reset by the act of either setting or resetting the corresponding interrupt enable bit. This must be done by the bit set/reset command written to CNT2 (OF). Writing a byte to port 2C does not affect the interrupt flip-flop (see Figure 1-5). This logic design has three purposes:

- a) After an interrupt from one source has been processed, its flip-flop can be cleared without affecting any other source which may have received a signal while the previous interrupt was being serviced.
- b) A previously disabled source can be enabled and its flip-flop cleared so that only future events will generate interrupts.
- c) A previously disabled source can be enabled without clearing its flip-flop (by writing to Port 2C instead of CNT 2) so that a previous event can generate an interrupt.

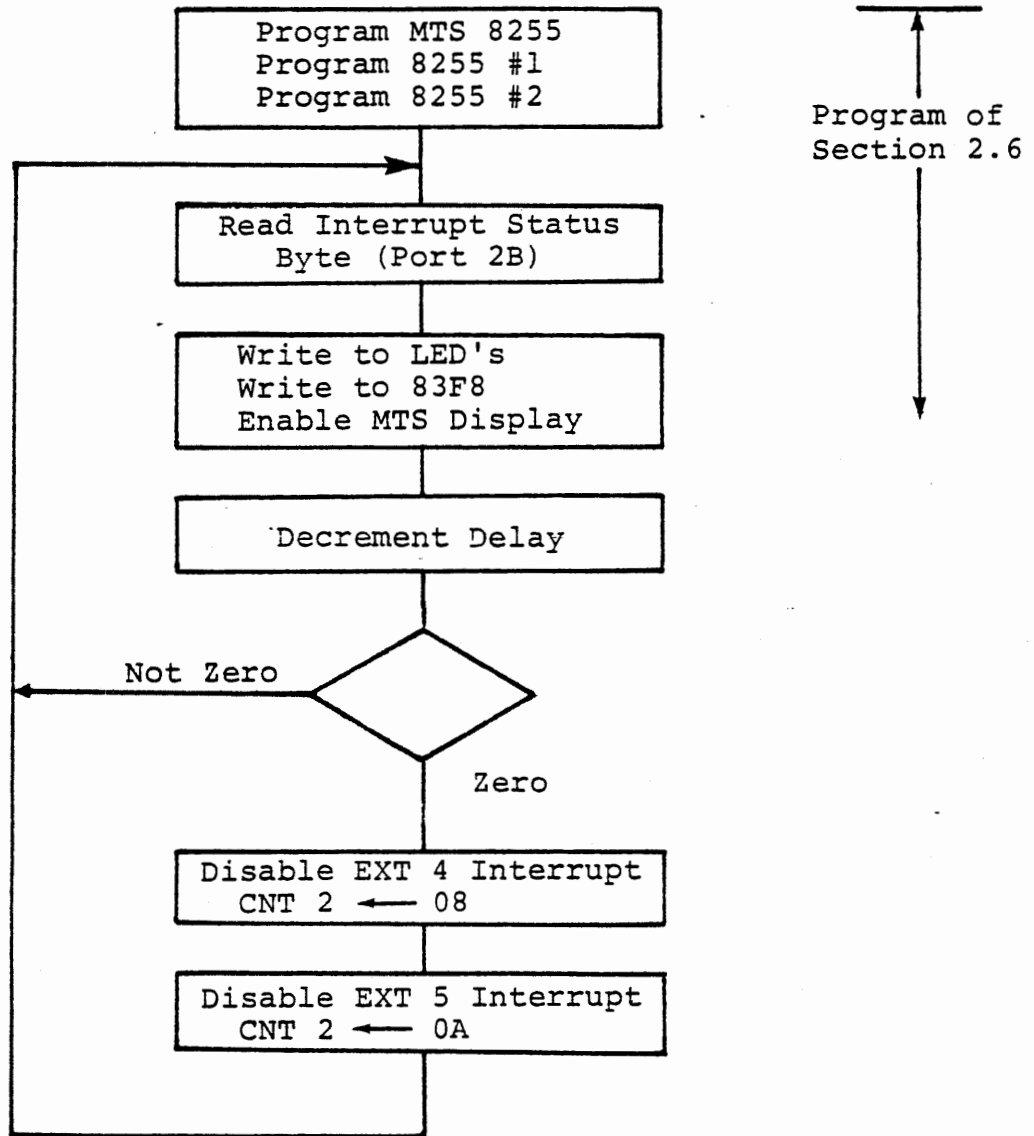


Figure 2-8

EXERCISE:

We will demonstrate the setting and clearing of the interrupt flip-flops for EXT 4 and EXT 5, using the connection already set up (Figure 2-7), and extending the program of Section 2.6. The program will display EXT 4 and EXT 5 flip flops as well as the direct inputs and demonstrate clearing the flip flops. The routine will provide a delay period during which the inputs can be controlled manually and will be displayed. At the end of the delay, it will clear the flip flop by a disable command. LED's DS6 and DS7 will display EXT 4 and EXT 5 direct inputs, while LED's DS4 and DS5 will display the state of EXT 4 and EXT 5 flip flops, respectively. Figure 2-8 shows the program.

During a delay period the interrupt status byte is repeatedly read and displayed. At the end of the delay, the EXT 4 and EXT 5 flip-flops are cleared by disabling their control bits in port 2C. Now if you ground the EXT 4 input during the delay period, its inverted output ($\overline{\text{EXT4 OUT}}$) will become high and set the EXT5 flip flop. These signals will be displayed. If you remove the ground, the EXT4 flip-flop will be set. If both flip flops are set (both LED's on), it is due to "bouncing" the jumper contact more than once during a delay. In fact it is very difficult to make or break the connection so cleanly that you do not set both flip-flops, but it is possible. This phenomenon of seeing both rising and falling edges when a contact is opened or closed is called "contact bounce", and must be considered when a computer is connected to switches.

CLEARING INTERRUPT FLIP FLOPS

	A	D	D	R	CODE						
CODING SHEET	8	2	0		3E	MVI	A,	92			Program MTS 8255's
					92						
					D3	OUT		CNT0			
					03						
					3E	MVI	A,	80			Program 8255 # 1
					80						
					D3	OUT		CNT1			
					07						
					3E	MVI	A,	92			Program 8255 # 2
					92						
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT		CNT2			
	B				0F						
	820	C			DB	IN		PORT2	B		Read interrupt status byte
		D			0D						
		E			D3	OUT		PORT1	A		Output to LED's
		F			04						
	8	2	1	0	32	STA		83F8			Output to MTS Display
					F8						
					83						
					3E	MVI	A,	0F			Enable MTS Display
INTEGRATED COMPUTER SYSTEMS					0F						
					D3	OUT		CNT0			
					03						
					2B	DCX		H			Delay with two byte countdown in (HL)
					7C	MOV	A,	H			
					B5	ORA		L			
					C2	JNZ		820C			Repeat input and display
					0C						
					82						
					00	NOP					
				00	NOP						
				00	NOP						
8											

Figure 2-9a

CLEARING INTERRUPTS continued

		A	D	D	R	CODE							
CODING SHEET	8	2	2	0	3	E	M	V	I	A, 08	Disable and clear		
											EXT4 Interrupt		
		1				08							
		2				D3	O	U	T	C	N	T	Z
		3				0F							
		4				3E	M	V	I	A, 0A	Disable and clear		
		5				0A					EXT5 Interrupt		
		6				D3	O	U	T	C	N	T	Z
		7				0F							
		8				C3	J	M	P	820C	Back to input		
MICROCOMPUTER TRAINING SYSTEM	9				0C								
	A				82								
	B												
	C												
	D												
	E												
	F												
	8												
		0											
		1											
		2											
		3											
		4											
		5											
		6											
	INTEGRATED COMPUTER SYSTEMS		7										
		8											
		0											
		1											
		2											
		3											
		4											
		5											

Figure 2-9b

2.8 RESTART INSTRUCTIONS

The 8080 provides for eight "restart" instructions RST0, RST1 --- RST7. (See pages 8-55 through 8-67 in Course 525). The MTS monitor program permits five of the RST instructions to be used:

RST 0 Corresponds to system reset, generated by power up or the reset key.

RST 4 Enters the monitor program as though the STEP key has been used.

RST 5 Jumps directly to address 8228

RST 6 Jumps directly to address 8230

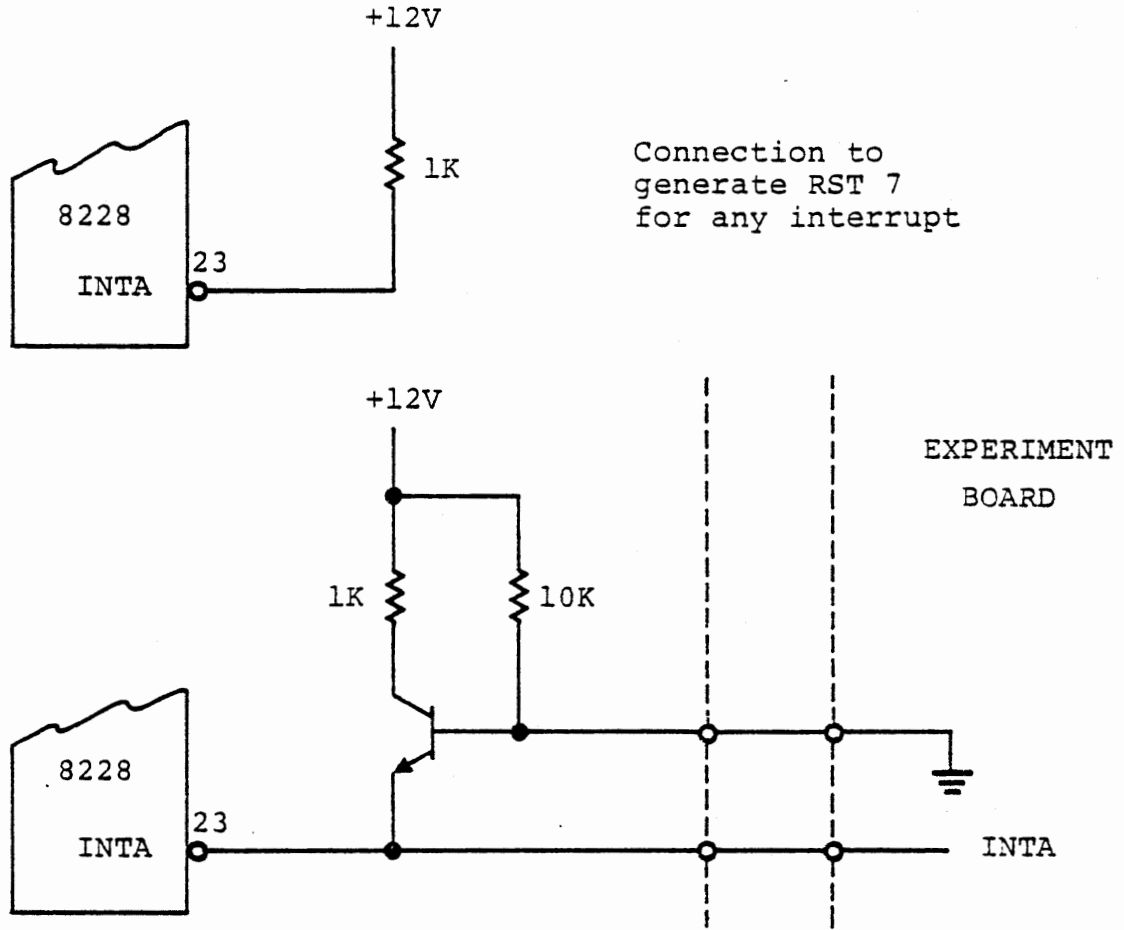
RST 7 Enters the monitor to test for a breakpoint or the STEP key.

2.8.1 AUTOMATIC RST 7

The MTS hardware as originally delivered automatically generates RST 7 when any interrupt occurs. A modification must be made to permit insertion of other restart instructions by external interrupts. This is done by ICS on MTS boards supplied with Course 536. The 8228 system controller enters the RST 7 instruction if its INTA output is pulled up to +12 volts through a resistor. The modification shown in

Figure 2-10 permits the automatic RST 7 generation to occur if the experiment board is not connected, because the transistor is turned on and INTA is pulled up. When the experiment board is connected to the MTS, the transistor is turned off by its grounded base. INTA is now an output signal from the 8228, indicating that the processor has acknowledged an interrupt request and expects the interrupt system to place an instruction on the data bus.

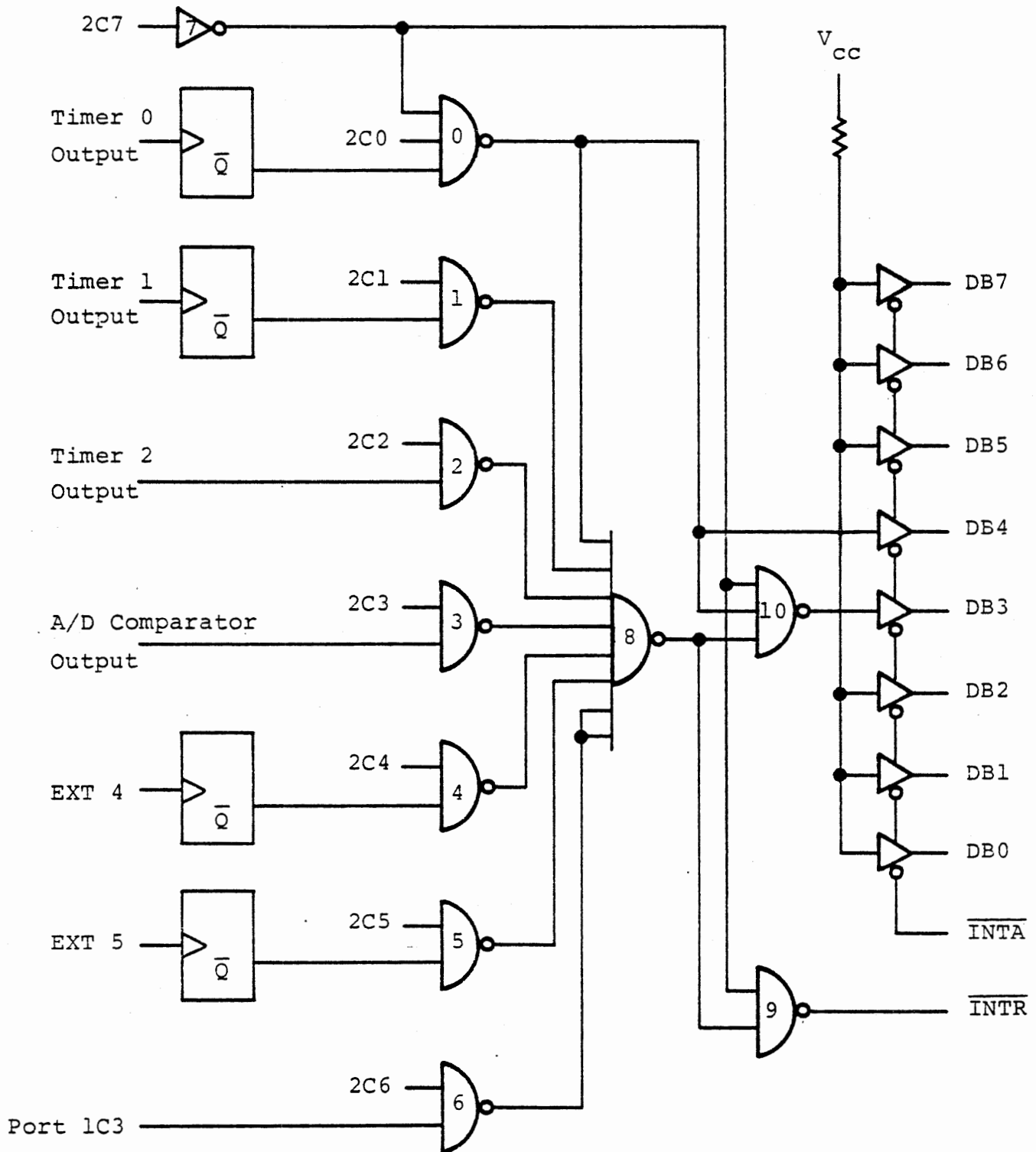
This Page Intentionally Left Blank



Modified connection generates RST 7 automatically if experiment board is not connected, but when experiment board is present it must generate all RST instructions.

AUTOMATIC RST 7 GENERATION

Figure 2-10



(Numbers in gates are for reference to text, and do not indicate chip numbers.)

GENERATION OF RST INSTRUCTIONS

Figure 2-11

2.8.2 RST Generation

The logic for generating interrupt request and restart instructions is shown in Figure 2-11. Port 2C7 is the general disable for interface board interrupts. When it is high (or floating), none of the interface board sources can generate an interrupt request. If the monitor hardware generates an interrupt request, all data bus bits will be high, giving an RST 7 interrupt.

When port 2C7 is low, the interface board interrupt sources can be enabled by the other bits of port 2C. If any interrupt source is high and its corresponding enable bit in port 2C is high, the NAND gate (0, 1, 2---6) output becomes low, forcing the output of gate 8 high. Now gate 9 generates the interrupt request, which is OR gated on the MTS board with the monitor interrupt request, so in STEP mode an interrupt occurs on every user instruction, but in AUTO mode an interrupt occurs only if 2C7 is low and one of the NAND gates 0-6 is low.

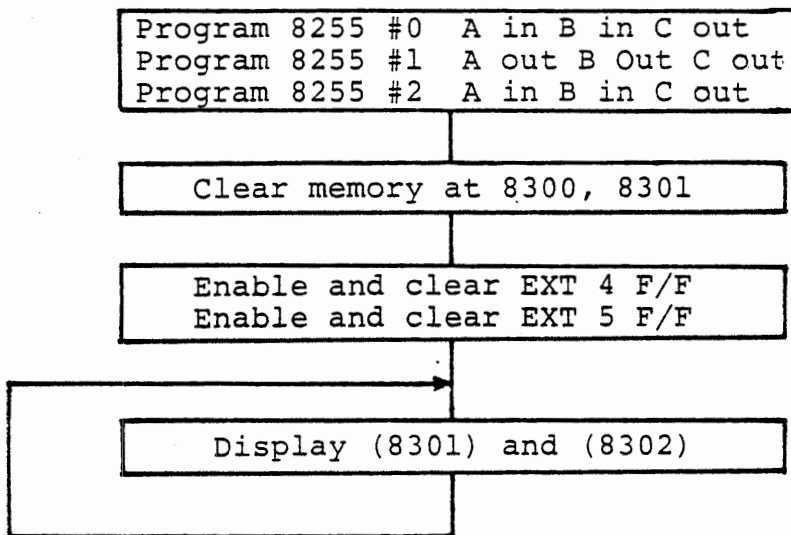
When INTA is output by the 8228 in response to the interrupt request, the tri-state buffers are enabled to drive the data bus (DB0-DB7). Six of these are always high; DB3 and DB4 are controlled by the gates. The following possible combinations exist:

- * 2C7 high. The interrupt request was generated by the MTS hardware. Gate 0 and gate 10 outputs are both high, giving 11111111 on the data bus. This is an RST 7 instruction.

- * 2C7 low, timer 0 flip flop and 2C0 high. Gate 0 output low, forcing gate 10 output high. This gives 11101111 on the data bus, a RST 5 instruction.

- * 2C7 low, timer 0 flip-flop or 2C0 low. Gate 0 output is high. Now if any other interrupt source and its enable bit are high, gate 10 is low, giving 11110111, RST 6, on the data bus.

- * 2C7 low but no enabled source high. Again the interrupt request has come from the monitor; gate 0 and gate 10 are high, and RST 7 is generated.



INTERRUPT SERVICE FOR RST 6

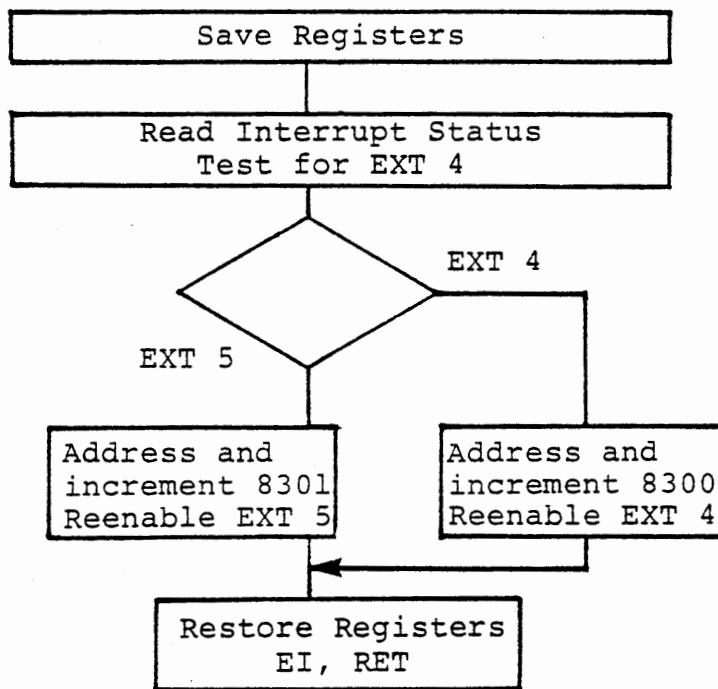


Figure 2-12

2.9 INTERRUPT SERVICE FOR EXT 4 AND EXT 5

EXERCISE

Develop a program to count the number of times the EXT 4 input is connected to ground and released. Program the 8255's as in the preceding sections. Clear two bytes of variable memory at 8300 and 8301. Enable EXT 4 and EXT 5 interrupts (using the bit set command). Write a main program with a repetitive loop that loads and displays the two bytes from variable memory: high order byte for EXT 4, low order byte for EXT 5.

Write an interrupt service routine at 8230 to distinguish EXT 4 from EXT 5. Set the interrupt enable bit (to clear the flip-flop) and increment a count of number of interrupts. Use location 8300 for EXT 4 and 8301 for EXT 5. A flow diagram appears in Figure 2-12.

Figure 2-13 lists the status bytes resulting from the various interrupt sources and the command bytes to disable or re-enable the interrupts.

A solution to the programming problem is given in Figure 2-14a and 2-14b. In 2-14c an alternate interrupt service routine is shown to demonstrate two programming tricks. It is only necessary to save registers that will be used. Here only H, L, A and flags are use. When a conditional jump is to be made based on a yes-no decision, it is often more efficient to assume one result before making the jump. Her we can replace a three byte LXI 8301 (at 8246) with a single byte INX H, and we can omit the JMP 824E (at 8243). Such tricks are often

powerful, but should be introduced only after a successful program has been written. Patching the program would be difficult in this situation.

STATUS AND COMMAND BYTES

INTERRUPT SOURCE	STATUS BYTE OBTAINED BY IN PORT 2B (see Note 2)									COMMAND BYTE WRITTEN BY OUT CNT 2 (see Note 1)		
	BINARY									HEX	DISABLE	ENABLE
Timer 0	0	X	X	X	X	X	X	1	01	00	01	
Timer 1	0	X	X	X	X	X	1	X	02	02	03	
Timer 2	0	X	X	X	X	1	X	X	04	04	05	
A/D Comparator	0	X	X	X	1	X	X	X	08	06	07	
EXT 4	0	X	X	1	X	X	X	X	10	08	09	
EXT 5	0	X	1	X	X	X	X	X	20	0A	0B	
Port 1C3	(see Note 3)										0C	0D

Note 1: Disable or enable command byte must be output to CNT 2 to clear the interrupt flip flop for Timer 0, Timer 1, EXT 4, or EXT 5. Disable or enable for A/D Comparator clears the interrupt in automatic A/D mode only.

Note 2: The hex values shown assume all other bits are 0. ANI (hex value) will give zero if the interrupt is not present.

Note 3: Port 1C3 does not appear in the status byte. It is read as XXXX1XXX by IN PORT1C. It is cleared by reading PORT1A in strobed input mode (mode 1 or mode 2) or by writing to PORT1A in strobed output mode (mode 1 or mode 2). Otherwise it can be cleared or set by writing 06 or 07 to CNT1. The interrupt enable for Port 1C3 is cleared or set by writing 0C or 0D to CNT2, but this does not change the data at Port 1C3.

STATUS AND COMMAND BYTES

Figure 2-13

MAIN FOR EXT 4 AND EXT 5 SERVICE

2 - 42

		A	D	D	R	CODE				
CODING SHEET	8	20	0	3E		MVI	A,	92	Program MTS 8255	
			1	92						
			2	D3		OUT		CNT0		
			3	03						
			4	3E		MVI	A,	80	Program 8255 #1	
			5	80						
			6	D3		OUT		CNT1		
			7	07						
			8	3E		MVI	A,	92	Program 8255 #2	
			9	92					(sets all enable	
MICROCOMPUTER TRAINING SYSTEM		A	D3		OUT			CNT2	bits low)	
		B	0F							
		C	21		LXI	H,		0000	Clear two bytes	
		D	00						of memory at	
		E	00						8300 and 8301	
		F	22		SHLD			8300		
		8	21	0	00					
				1	83					
				2	3E		MVI	A,	09	Enable EXT4
				3	09					interrupt and
INTEGRATED COMPUTER SYSTEMS			4	D3		OUT		CNT2	clear its Flip Flop	
			5	0F						
			6	3E		MVI	A,	0B	Enable EXT5	
			7	0B					interrupt and	
			8	D3		OUT		CNT2	clear its Flip Flop	
			9	0F						
		821	A	2A		LHLD			8300	
			B	00						
			C	83						
			D	CD		CALL			DWORD	
		E	D1							
		F	02							
	8	22	0	C3		JMP		821A		
			1	1A						
			2	82						
			3							
			4							
			5							
			6							
			7							
			8							

Figure 2-14a

INTERRUPT SERVICE FOR EXT 4 AND EXT 5

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE					
8	2	3	0	FS	PUSH	H	PSW		Save registers
			1	ES	PUSH	H			
			2	DS	PUSH	H	D		
			3	CS	PUSH	H	B		
			4	DB	IN		PORT 2 B		Read interrupt
			5	OD					status byte
			6	E6	ANI		10		Test for EXT 4
			7	10					
			8	CA	JZ		8246		Jump if not EXT 4
			9	46					
			A	82					
			B	21	LXI	H,	8300		Address and
			C	00					increment counter
			D	83					for EXT 4
			E	34	INR	M			
			F	3E	MVI	A,	09		Reenable EXT 4
8	2	4	0	09					and clear flip flop
			1	D3	OUT		CNT 2		
			2	0F					
			3	C3	JMP		824E		Jump to exit
			4	4E					
			5	82					
8	2	4	6	21	LXI	H,	8301		Address and
			7	01					increment counter
			8	83					for EXT 5
			9	34	INR	M			
			A	3E	MVI	A,	0B		Reenable EXT 5
			B	0B					and clear flip flop
			C	D3	OUT		CNT 2		
			D	0F					
8	2	4	E	C1	POP	B			
			F	D1	POP	D			
8	2	5	0	E1	POP	H			
			1	F1	POP	PSW			
			2	FB	EI				
			3	C9	RET				
			4						
			5						
			6						
			7						
			8						

Figure 2-14b

SHORTER INTERRUPT SERVICE FOR EXT 4, EXT 5 2 - 45

		A	D	D	R	CODE						
CODING SHEET	8	2	3	0		FS		PUSH	H	PSW		Save only registers
				1		ES		PUSH	H			that will be used
				2		DB		IN		PORT 2 B		Read interrupt
				3		OD						status byte
				4		EG		ANI		10		Test for EXT 4
				5		10						
				6		21		LXI	H	8300		Address counter
				7		00						for EXT 4
				8		83						
				9		3E		MVI	A	09		Reenable byte
MICROCOMPUTER TRAINING SYSTEM	A					09						for EXT 4
	B					C2		JNZ	8241		Jump if input	
	C					41					was EXT 4/	
	D					82						
	E					23		INX	H		Address EXT 5 Count	
	F					3E		MVI	A	0B	Reenable byte	
	8	2	4	0		0B						for EXT 5
		8	2	4	1		D3		OUT	CNT 2		
				2			0F					
				3			34		INR	M		
			4			E1		POP	H			
			5			F1		POP	PSW			
			6			FB		EI				
			7			C9		RET				
INTEGRATED COMPUTER SYSTEMS	8											

Figure 2-14c

Program 8255's - 1B out

3E	MVI	A,80
80		
D3	OUT	CNT1
07		
3E	MVI	A,92
92		
D3	OUT	CNT2
0F		

Program 8255's - 1B in

3E	MVI	A,82
82		
D3	OUT	CNT1
07		
3E	MVI	A,92
92		
D3	OUT	CNT2
0F		

STANDARD PROGRAMMING FOR 8255'S

Figure 2-15

2.10 STANDARD PROGRAMMING FOR 8255'S

In Figure 2-12 we programmed the 8255's as follows:

8255	0	A in	B in	C out
8255	1	A out	B out	C out
8255	2	A in	B in	C out

Almost all of the exercises in this course will use either that programming or the same, except for port 1B.

8255	1	A out	B in	C out
------	---	-------	------	-------

In most program flow diagrams hereafter, we will show either

Program 8255's - 1B out

Program 8255's - 1B in

This is to imply the programming above, with the assumption that the user program need not program 8255 0 since the monitor sets it in the required condition. Figure 2-15 shows the program steps. This is duplicated in Appendix A. You may want to post it in a convenient place.

MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 3

INTERVAL TIMERS

3. INTERVAL TIMERS

Timing functions are extremely common in computers used in real time applications and communications. Timing can be achieved by program loops but with two major limitations. The precision of the timing is limited to the length of the loop, (commonly of the order of four or more instructions) and the computer can do nothing else while it is timing. Hardware timers overcome these limitations at moderate cost.

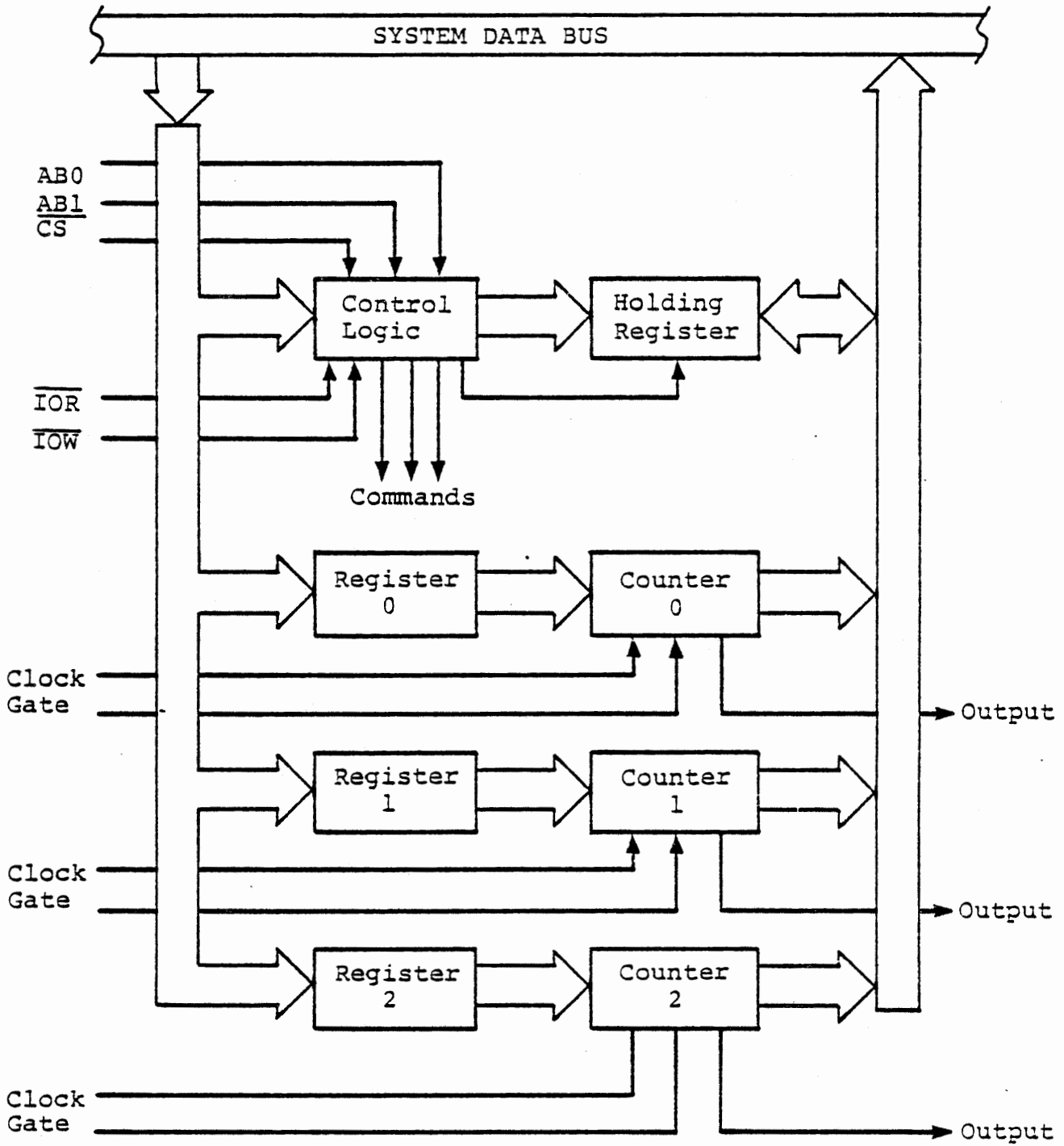
3.1 INTEL 8253 INTERVAL TIMER

The 8253 provides three identical, independent 16 bit timers. Each timer comprises (Figure 3-1) a 16 bit counter and a 16 bit storage register (accessible by IN and OUT instructions), control logic and flip flops, a clock input, gate input, and an output. Each of the timers occupies one I/O port address for reading the counter and loading the register. A fourth I/O port provides for controlling all of the counters. Various operation modes exist which may be selected by writing a control byte to the control port.

Initiating a timer's operation always involves two steps. First the timer "mode" must be specified to the control register. Second, the timer count-down value is initialized. Typically this requires the following sequence.

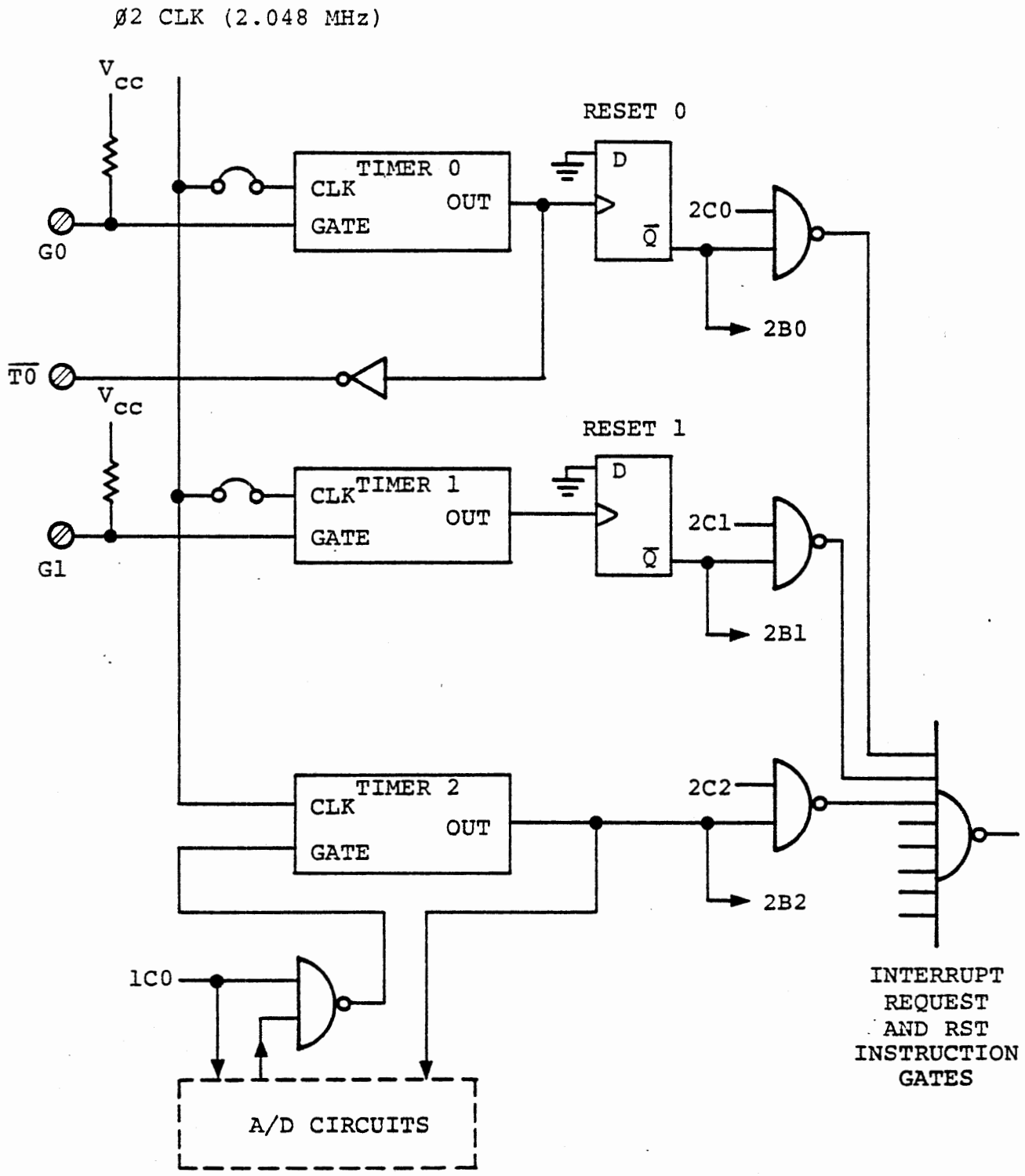
MVI A, control byte	Write control byte to
OUT TIMCT	timer control port, to set mode.
MVI A, low data byte	
OUT TIMER	Load time data to storage register
MVI A , high data byte	
OUT TIMER	Load high time data byte

After the count value has been initialized the timer will run under control of its clock and gate inputs, and will generate a particular output signal depending on which timer mode was pre-specified. The output waveform could be used as a timed interrupt to the microprocessor, a low speed clock for an external circuit, or a variety of other applications in a microcomputer system, as we will see throughout the course.



INTEL 8253 INTERVAL TIMER

Figure 3-1



TIMER CLOCKS, GATES AND OUTPUTS

Figure 3-2

3.2 CLOCK, GATE, AND OUTPUT

Each timer receives a clock input and decrements the content of its counter at the falling edge of the clock. On the experiment board all clock inputs are normally connected to the system 2.048 MHz clock, but this connection can be altered to permit use of an external clock input. (See Figure 3-2.)

The gate input to each timer starts, enables or disables its counting, depending on the selected mode. On the experiment board these inputs for timer 0 and timer 1 are pulled high by resistors so that counting is normally enabled, but these gate inputs are also accessible at terminals for external control. The gate input to timer 2 is connected to the analog to digital converter circuitry because timer 2 is often used in A/D operations (as discussed in Chapter 5). To enable timer 2 for other functions its gate input must be forced high by setting port 1C0 low.

The output of a timer goes high to indicate the end of a time interval. The time at which it goes low depends on the mode selected. On the experiment board the outputs of timer 0 and timer 1 set flip flops in the interrupt system, exactly like the EXT 4 and EXT 5 flip flops. Timer 2 output has no flip flop. It is directly gated with an interrupt enable bit into the interrupt system, and it is also used to drive a counter in the A/D converter.

The output of timer 0, as well as setting an interrupt flip-flop, also drives an inverter whose output is available at a terminal for use by external hardware.

Because the system clock is nominally 2.048 MHz it is very easy to relate binary counts to decimal times. The following table lists some useful values.

Binary Count	Decimal Count	Time
(Hexadecimal)		(milliseconds)
0100	256	0.125
0200	512	0.250
0400	1024	0.500
0800	2048	1.000
1000	4096	2
1800	6144	3
2000	8192	4
2800	(10240)*	5
3000	(12288)	6
3800	(14336)	7
4000	(16384)	8
4800	(18432)	9
5000	(20480)	10
A000	(40960)	20
F000	(61440)	30
0000	(65536)	32
		Time (seconds)
1F40	8000	1/256
0FA0	4000	1/512
07D0	2000	1/1024

* The timer cannot be loaded with decimal values greater than 9999, so binary counting must be used.

Relating Counts to Time

This Page Intentionally Left Blank

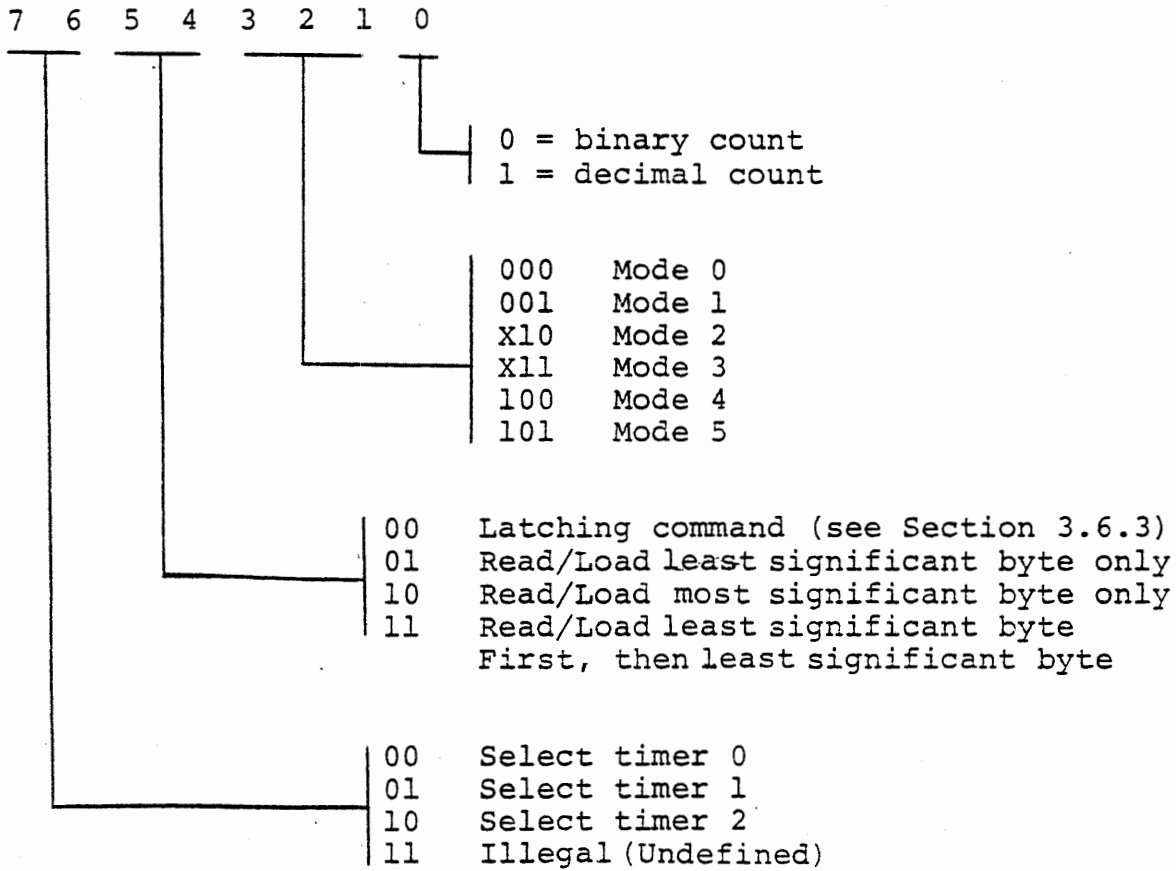
3.3 TIMER MODES

Any of the three interval timers can operate in any of six modes (0-5). Most of the experiments here use mode 0 or mode 2. The other modes are intended principally for interfacing the timer directly with external hardware rather than through the program. The modes are listed below, and defined in subsequent sections along with experiments. A summary of the modes is given in section 3.10.

Mode 0	Interrupt on Terminal Count
Mode 1	Programmable One Shot
Mode 2	Rate Generator
Mode 3	Square Wave Generator
Mode 4	Software Triggered Strobe
Mode 5	Hardware Triggered Strobe

Within each of these modes the user has some additional options. The counters are 16 bits long, and can be loaded with two bytes of data, less significant byte first. Two other options (which must be selected when the mode is programmed) are to load only the less significant byte or to load only the more significant byte. In either of these cases, the other byte is set to 00, and counting proceeds on both bytes.

Timer Control Byte Structure



TIMER CONTROL BYTE STRUCTURE

Figure 3-3

The timers can count in binary or decimal, as selected when the mode is programmed.

The mode and options are selected for any one of three timers by writing a byte to the control port of the 8253.

```
3E      MVI A,CONTROL BYTE
XX
D3      OUT TIMCT
17
```

Figure 3-3 shows the bit structure of the control byte. Figure 3-4 lists the most commonly used control bytes for each of the three timers.

Timer 0	Mode					
	0	1	2	3	4	5
Latch	00	00	00	00	00	00
Read/Load LSB	10	12	14	16	18	1A
Read/Load MSB	20	22	24	26	28	2A
Read/Load Both (LSB first)	30	32	34	36	38	3A

Timer 1	Mode					
	0	1	2	3	4	5
Latch	40	40	40	40	40	40
Read/Load LSB	50	52	54	56	58	5A
Read/Load MSB	60	62	64	66	68	6A
Read/Load Both (LSB first)	70	72	74	76	78	7A

Timer 2	Mode					
	0	1	2	3	4	5
Latch	80	80	80	80	80	80
Read/Load LSB	90	92	94	96	98	9A
Read/Load MSB	A0	A2	A4	A6	A8	AA
Read/Load Both (LSB first)	B0	B2	B4	B6	B8	BA

Control Bytes shown set binary counting

Add 1 for decimal counting

Write control byte to TIMCT, Port 17

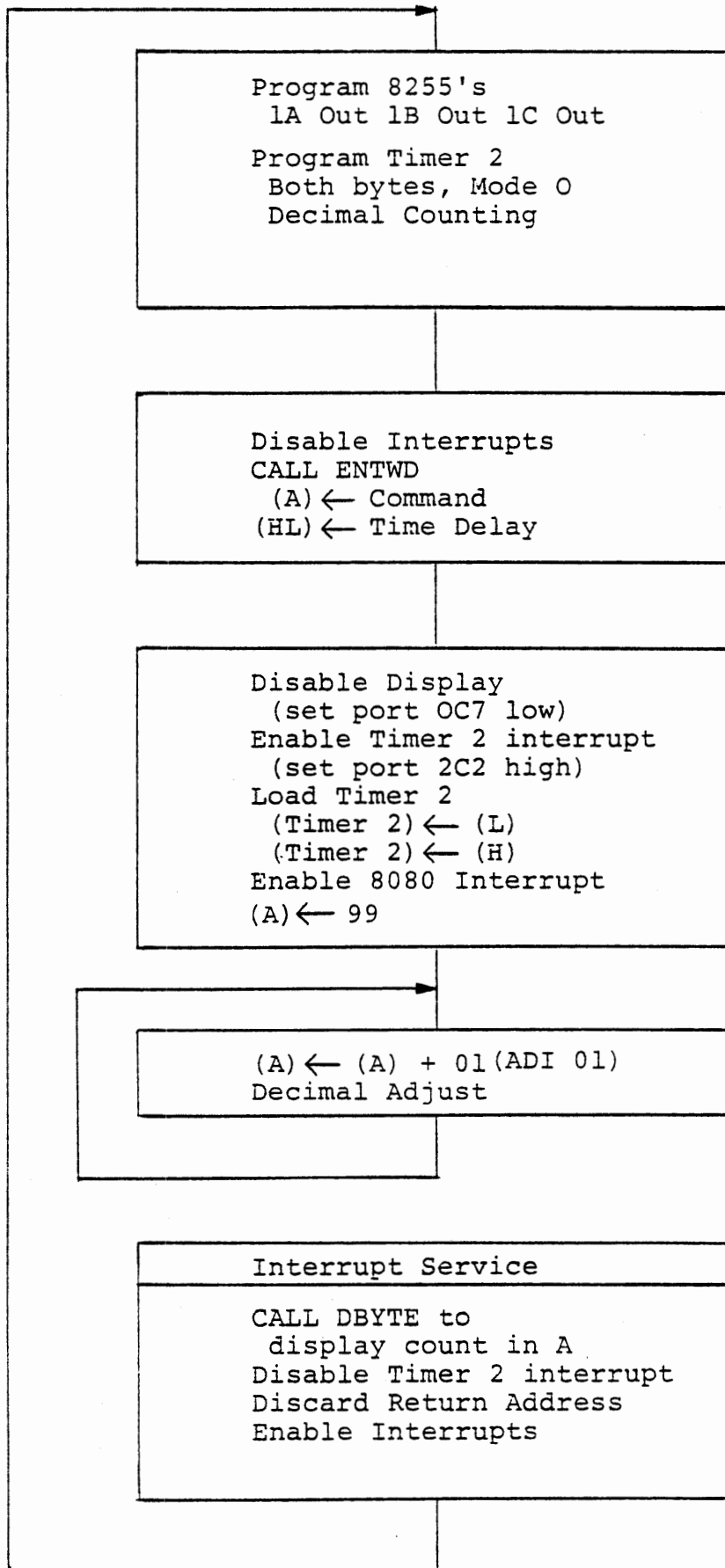
Latching control byte does not affect mode

TIMER CONTROL BYTES

FIGURE 3-4

3.4 MODE 0 - INTERRUPT ON TERMINAL COUNT

When a timer is set to mode 0, its output goes low. When it has been loaded (with one or two bytes as required by the mode select option), and its gate input is high, it will decrement the count at each falling edge of the clock. When the count reaches zero, the output goes high. Mode 0 is intended to generate a time delay whose duration and starting time are set by the program. In the following exercise we compare a programmed timing loop with an interval timer. Figure 3-5 shows the program flow diagram.



COMPARE TIMING LOOP WITH INTERVAL TIMER
FIGURE 3-5

EXERCISE

This program accepts a time delay value from the keyboard, and starts timer 2 in mode 0 with this value. After enabling interrupts it enters a counting loop. At the interrupt generated by timer 2 it displays the value reached by the counting loop. The interrupt service discards the return address (by POP H) and jumps to start since the function of the main program is finished when the interrupt occurs.

Note that timer 2, which has no external flip-flop in the interrupt system, is appropriately used here because in mode 0 its output goes low when it is programmed to mode 0 or when it is loaded, goes high and stays high at the end of the interval.

The addresses, programming control bytes, and interrupt enable/disable bytes are found in Figures 2-2, 2-13, and 3-4. Duplicate copies of these are found in Appendix A. You may want to post them for ready reference.

The delay loop should be:	ADI	01	9 clocks
	DAA		5
	JMP		13
			27 clocks

The interval timer will count 27 times as fast as the programmed timing loop. When you run the program, find the smallest delay value you can enter that results in a zero in (A). This represents the time taken to reach the DAA instruction after the second byte is loaded to

This Page Intentionally Left Blank

the timer. (Run the program in AUTO mode to make the time measurements.)

You should be able to add 27 to that value and get a count of 01. (We programmed the timer and the loop for decimal counting to make the arithmetic easier.) Each added value of 27 in the delay should result in one added count in the result. At some intermediate values you will see hex values in the display because the interrupt occurred after ADI 01 but before DAA. At 2711 you should obtain a count of 99. With delays from 2712 to 2720 the count will be 9A, and at 2721 it will be 00.

If the display is not disabled during counting the programmed timing loop will be slower because of the hold states introduced by the DMA channel for the display.

Try running the program in STEP mode (but with the RUN key). Now you can measure the time taken by the monitor. Insert some breakpoints that will never be reached and observe the effect.

3-18

COMPARE TIMING LOOP WITH INTERVAL TIMER

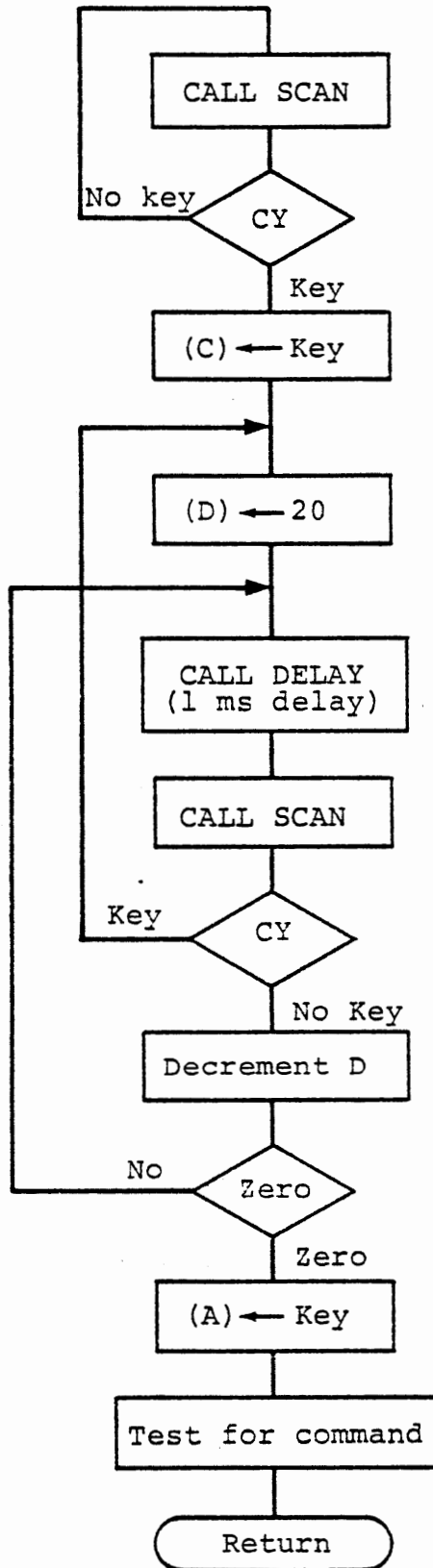
	A	D	D	R	CODE					
CODING SHEET	8	2	0	0	3E	MVI	A,	80	Program 8255#1	
					80				A out Bout Count	
					D3	OUT		CNT1		
					07					
					3E	MVI	A,	92	Program 8255#2	
					92				A in Bin Count	
					D3	OUT		CNT2		
					0F					
					3E	MVI	A,	B1	Program Timer 2	
					B1				Both bytes	
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT		TIMCT	Mode 0	
	B				17				Decimal	
	820	C			F3	DI			For step mode	
		D			CD	CALL		ENTWD		
		E			46					
		F			03					
	8	2	1	0	3E	MVI	A,	0E	Set Port OCT low	
					0E				to disable display	
					D3	OUT		CNT0		
					03					
INTEGRATED COMPUTER SYSTEMS					3E	MVI	A,	05	Set Port 2C2 high	
					05				to enable timer 2	
					D3	OUT		CNT2	interrupt (but	
					0F				8080 interrupts	
					7D	MOV	A,	L	} still disabled)	
					D3	OUT		TIM2		
		A			16				} Load time delay	
		B			7C	MOV	A,	H		} to timer 2
		C			D3	OUT		TIM2		
					D	16				
				E	FB	EI				
				F	3E	MVI	A,	99		
8	2	2	0	99						
8	2	2	1	C6	ADI		01			
				2	07					
				3	27	DAA				
				4	C3	JMP		8221		
				5	21					
				6	82					
				7						
				8						

FIGURE 3-6 a

INTERRUPT SERVICE FOR TIMING COMPARISON 3 - 19

	A	D	D	R	CODE														
CODING SHEET	8				0														
					1														
					2														
					3														
					4														
					5														
					6														
					7														
MICROCOMPUTER TRAINING SYSTEM	8	2	2	8	C3	JMP	8230												
					9	30													
					A	82													
					B														
					C														
					D														
					E														
					F														
		8	2	3	0	CD	CALL	DBYTE	Display Count										
					1	95													
					2	02													
					3	3E	MVI	A, 04	Disable Timer 2										
					4	04			interrupt										
					5	D3	OUT	CNT2											
					6	0F													
					7	E1	POP	H	Discard return										
				8	FB	EI		address											
				9	C3	JMP	8200	Jump to start											
INTEGRATED COMPUTER SYSTEMS					A	00													
					B	82													
					C														
					D														
					E														
					F														
		8				0													
						1													

FIGURE 3-6 b



GETKY FLOW DIAGRAM

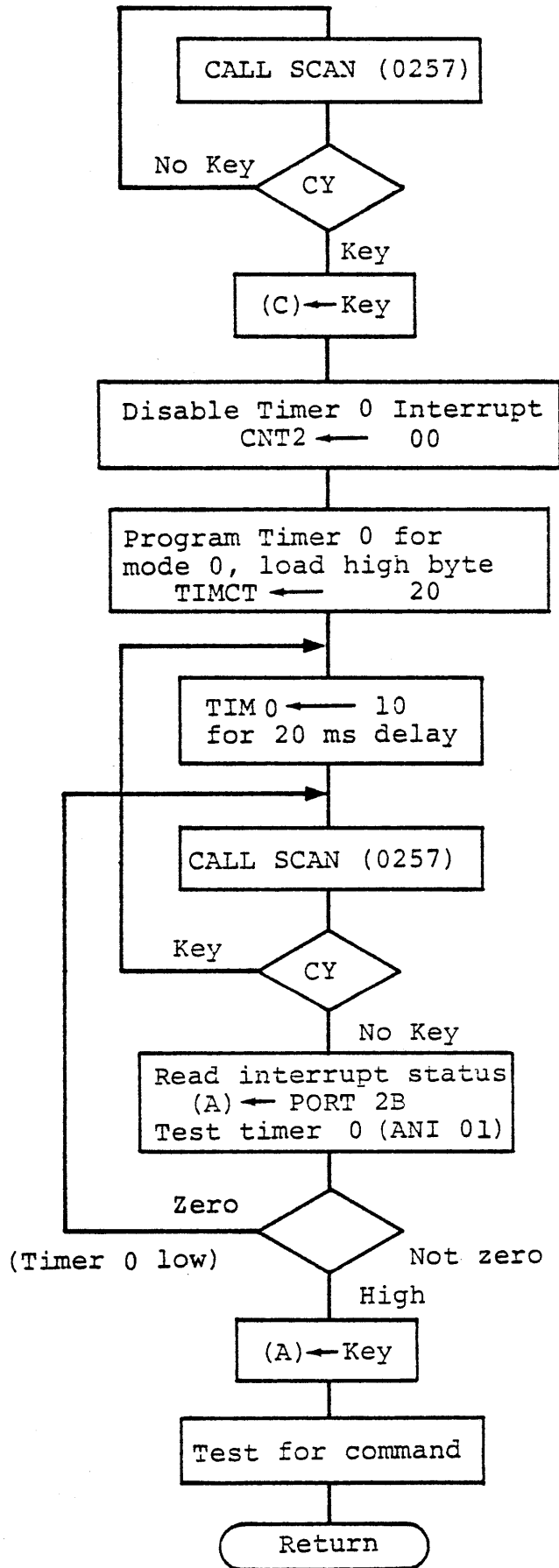
Figure 3-7

3.5 RESTARTING A COUNTER IN MODE 0.

When a counter is running in mode 0 it can be stopped and restarted by loading a new time count. The output will remain low while this is done, and go high only when the most recently loaded count reaches zero.

EXERCISE

The monitor subroutine GETKY is used to get a single keyboard entry. After a key has been pressed and read, it waits until the key has been released for 20 milliseconds to protect against contact bounce, which might otherwise cause a single key operation to be read as two or more operations. Figure 3-7 is a flow diagram for GETKY. Scan is the subroutine that actually reads the keyboard, returning with carry cleared if no key is pressed. If a key is pressed SCAN returns the hex value in (A) and carry set.



GETKY WITH TIMER

Figure 3-8

The delay loop in GETKY is a nuisance when a program is being run with the monitor enabled for breakpoints (i.e. with the MTS toggle switch at STEP) because the delay is exaggerated by a factor of 42 or more. If you are not familiar with the problem, load and run this program, first in AUTO mode and then in STEP.

```

      8200  CD    CALL GETKY
          01  3D
          02  02
          03  CD    CALL DBYTE
          04  95
          05  02
          06  C3    JMP 8200
          07  00
          08  82

```

We will develop a substitute for GETKY that uses timer 0 instead of a delay loop. Figure 3-8 is a flow diagram for this program. The diagram is generally the same as figure 3-7 except for the delay functions. Write this subroutine and call it instead of GETKY in the program above.

Note that here we restart timer 0 each time SCAN finds the key still present, but let it run when the key is released. Timer 0 output never goes high until the timer is decremented to zero.

We have disabled the timer 0 interrupt because this program tests timer 0 itself and does not want an interrupt to occur. Other

interrupts are allowed. It is necessary to disable the timer interrupt (using the bit reset function) to clear the flip flop, because it is the flip flop output that is read in port 2B, not the direct output from the timer. Timer 1 can be used in the same way. Make that substitution and see that the program still works. It can also work with timer 2, but port1C must be programmed for output and bit 1C0 set low, otherwise timer 2 may be inhibited from counting by the A/D circuitry.

GETKY

USING INTERVAL TIMER

A D D R CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	240	CD	CALL	SCAN	Read keyboard
	1	57			
	2	02			
	3	D2	JNC	8240	Repeat until key
	4	40			is pressed
	5	82			
	6	4F	MOV	C, A	(C) ← key
	7	3E	MVI	A, 00	Disable timer 0
	8	00			interrupt
	9	D3	OUT	CNT2	
A	0F				
B	3E	MVI	A, 20		Program timer 0
C	20				load high byte
D	D3	OUT	TIMCT		mode 0, binary
E	17				
824	F	3E	MVI	A, 10	TIM0 ← 10
825	0	10			for 20 ms delay
	1	D3	OUT	TIM0	
	2	14			
825	3	CD	CALL	SCAN	Read keyboard
	4	57			
	5	02			
	6	DA	JC	824F	If key still pressed
	7	4F			go restart timer
	8	82			If key released
	9	DB	IN	PORT2B	read interrupt
A	0D				status
B	E6	ANI	01		Test for timer 0
C	01				bit high
D	CA	JZ	8253		If not high
E	53				go read keyboard
F	82				again.
8	0	79	MOV	A, C	(A) ← key
	1	FE	CPI	10	Test for command
	2	10			Sets CY if hex
	3	C9	RET		No CY if command
	4				
	5				
	6				
	7				
	8				

Figure 3-9

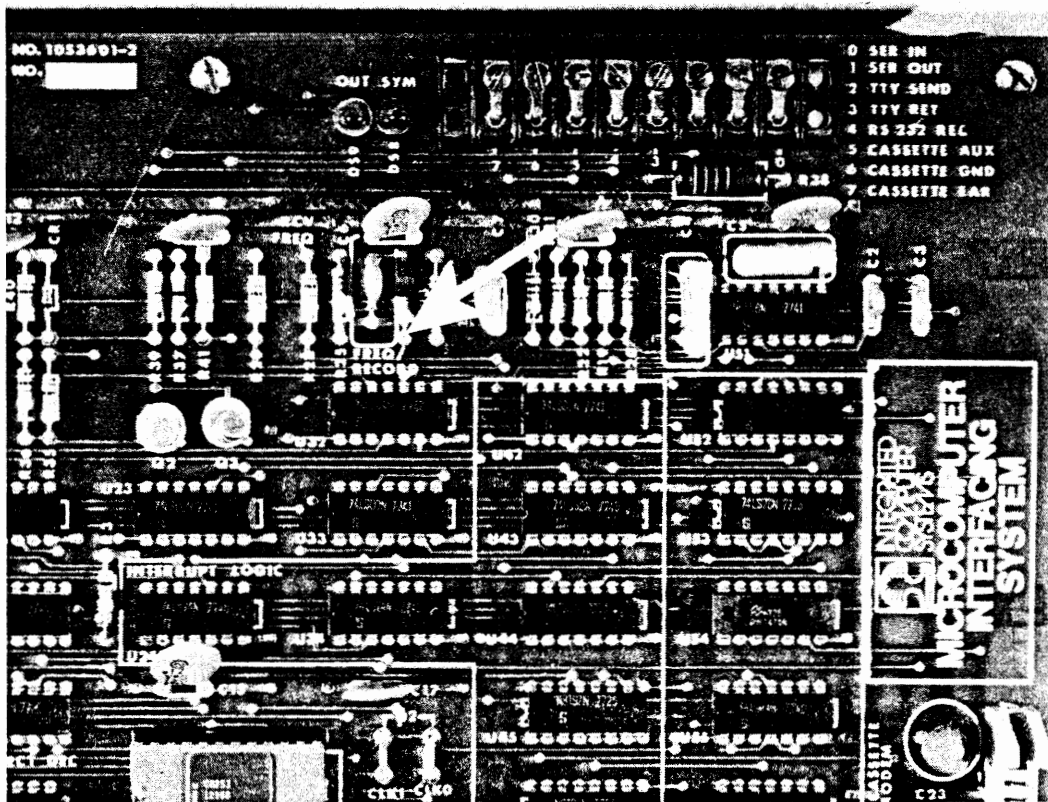
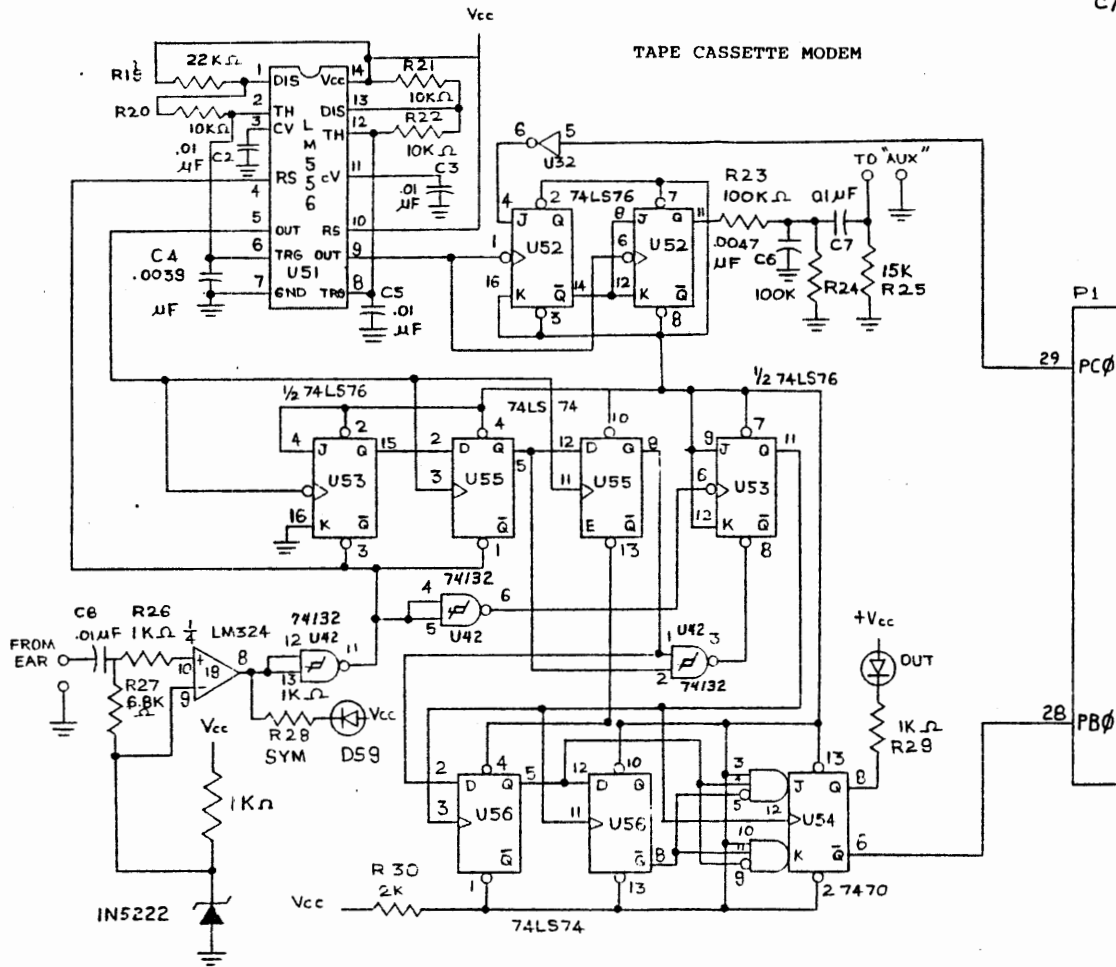


FIGURE 3-10

3.6 READING A TIMER

A timer can be read as well as loaded. The exercises of this section make use of that facility.

3.6.1 Measuring a Pulse Duration

EXERCISE

In mode 0 (also modes 2, 3 and 4) counting continues only while the gate input is high. We can use this to measure the width of a pulse. A useful signal source is the cassette modem output. Connect one end of a clip lead as indicated in Figure 3-10 (bottom of R23) to pick up the signal, and connect it to the gate input (G1 IN) for timer 1 and also at the EXT 4 input. The modem output is nominally 1200 Hz if port OCO output is low, and twice that when port OCO is high. We will write a program to select the frequency by keyboard input, measure the width of the high portion of the output signal, and display that width. The width is displayed in decimal clock pulse units. Divide the clock count by the clock frequency (2.048 MHz) to determine the input pulse width. Alternately, since the signal we are measuring is a square wave, obtain the frequency by $1024000/\text{count}$.

To measure the pulse width we will initially load timer 1 with zero, while the input signal is low. After the signal has gone high and then returned to low we will read the counter.

```

XRA  A           Enter zero to
OUT  TIM1        both bytes of
OUT  TIM1        the timer
      '          Wait for input
      '          to go high and
      '          then low
IN   TIM1        Read the timer
MOV  L,A         content into
IN   TIM1        registers H,L
MOV  H, A

```

Although we could clear the timer to zero with a single byte load, if it were so programmed, we would then be restricted to a single byte read.

The process above reads and stores the content of the timer, but since it counts down this result is the twos or tens complement of the actual time, as shown in Figure 3-11.

Binary Counting		Decimal Counting	
Positive Count	Timer Data	Positive Count	Timer Data
0000	0000	0000	0000
0001	FFFF	0001	9999
0002	FFFE	0002	9998
0003	FFFD	0003	9997
0004	FFFC	0004	9996
0005	FFFB	0005	9995
0006	FFFA	0006	9994
0007	FFF9	0007	9993
0008	FFF8	0008	9992
0009	FFF7	0009	9991
000A	FFF6	0010	9990
000B	FFF5	0011	9989
000C	FFF4	0012	9988
000D	FFF3		
000E	FFF2		
000F	FFF1		
0010	FFF0		
0011	FFEF		
00FF	FF01	0099	9901
0100	FF00	0100	9900
0101	FEFF	0101	9899
0FFF	F001	0999	9001
1000	F000	1000	9000
1001	EFFF	1001	8999
FFFF	0001	9999	0001
0000	0000	0000	0000

TWOS AND TENS COMPLEMENT COUNTING

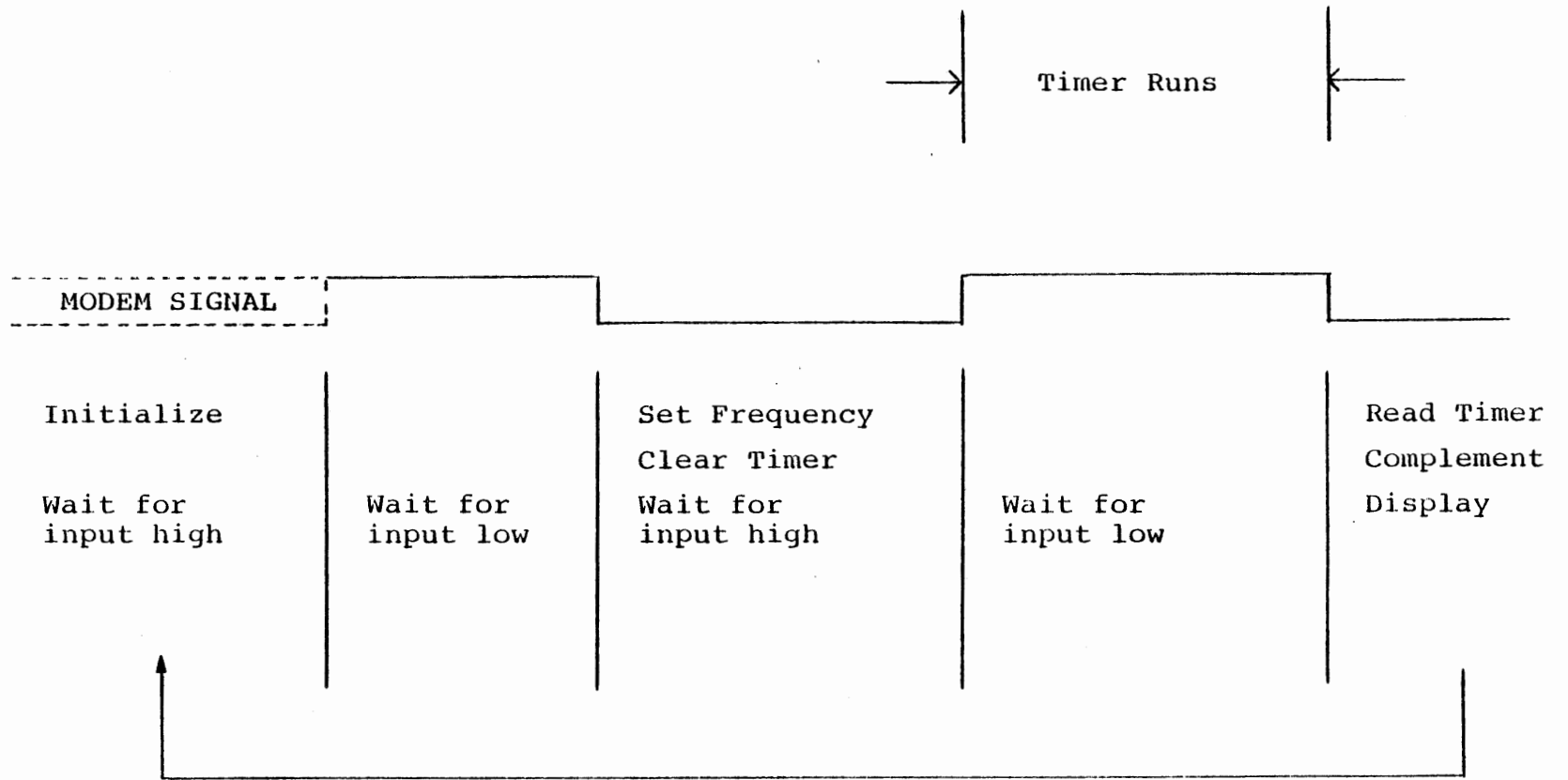
Figure 3-11

The twos complement can most easily be converted by complementing the byte as it is read and then adding one to the two byte result.

```
IN    TIM1
CMA
MOV   L,A
IN    TIM1
CMA
MOV   H,A
INX   H
```

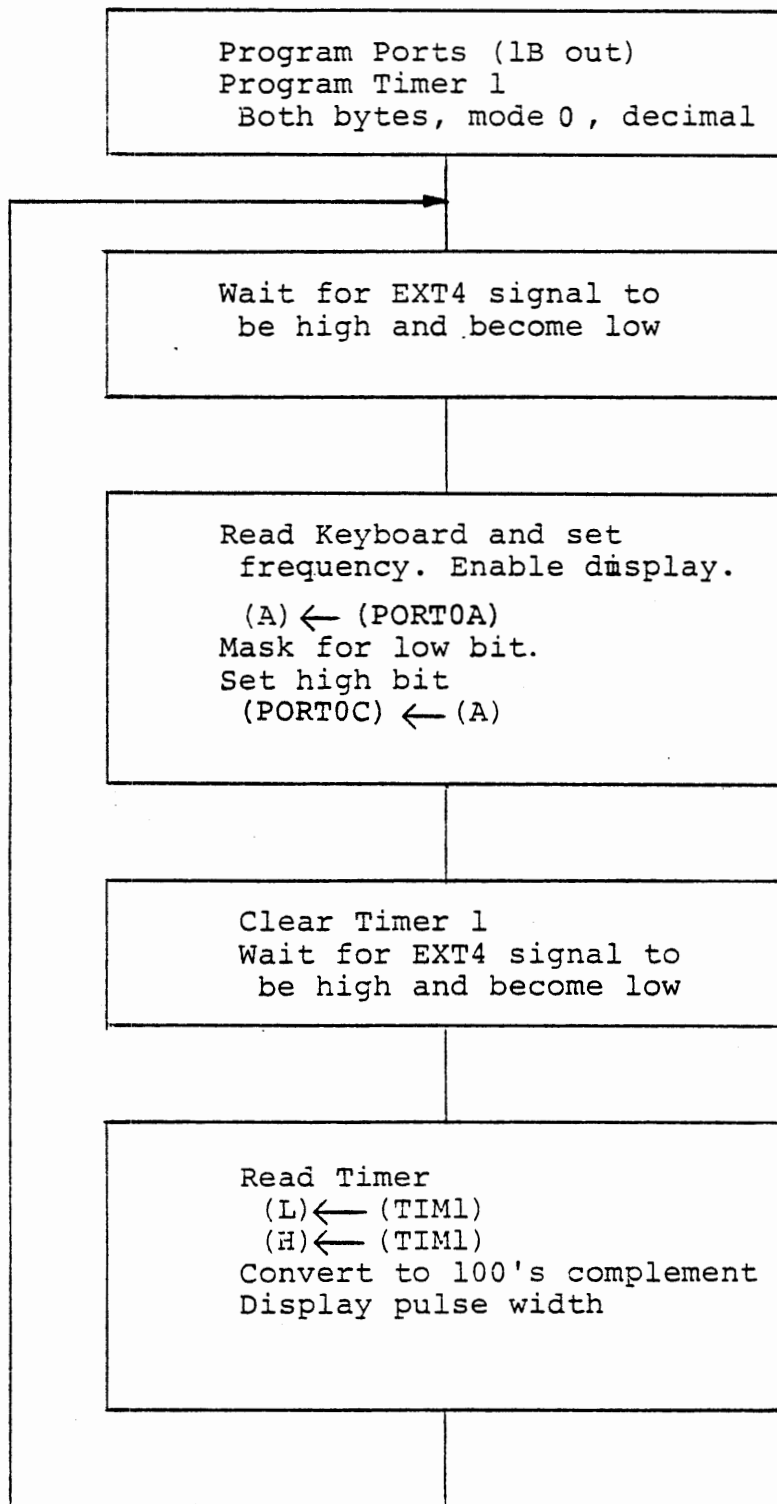
The tens complement is needed if we use decimal counting. Refer to course 525, figure 10-33, for a subroutine to convert a two byte decimal value to its tens or hundreds complement.

Although the counter in mode 0 will only run while its gate input is high, it gives no direct indication to the program when it stops. Figure 3-12 shows how the computer will react to the input signal. Figure 3-13 is a flow diagram for the program. A program solution is given in Figure 3-14 for decimal counting.



TIME DIAGRAM FOR PULSE WIDTH MEASUREMENT

FIGURE 3-12



PULSE WIDTH MEASUREMENT

FIGURE 3-13

PULSE WIDTH MEASUREMENT

	A	D	D	R	CODE						
CODING SHEET	8	2	0	0	3E	MVI	A,	80			Program 8255#1
					80						A out B out Cout
					D3	OUT	CNT1				
					07						
					3E	MVI	A,	92			Program 8255#2
					92						
					D3	OUT	CNT2				
					0F						
					3E	MVI	A,	71			Program Timer 1
					71						Both bytes
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT	TIMCT				Mode 0
	B				17						Decimal
	8	2	0	C	CD	CALL	WTHL				Wait until input
				D	30						has been high
				E	82						and becomes low
				F	DB	IN	PORT0A				Read keyboard
	8	2	1	0	00						
				1	F6	ANI	01				Mask for low bit
				2	01						and set high bit
				3	F6	ORI	80				to set frequency
INTEGRATED COMPUTER SYSTEMS				4	80						and enable display
				5	D3	OUT	PORT0C				
				6	02						
				7	AF	XRA	A				Clear Timer
				8	D3	OUT	TIM1				
				9	15						
				A	D3	OUT	TIM1				
				B	15						
				C	CD	CALL	WTHL				Wait through high
				D	30						part of pulse.
			E	82						Timer will run	
			F	00	NOP						
8			0								
			1								
			2								
			3								
			4								
			5								
			6								
			7								
			8								

FIGURE 3-14a

PULSE WIDTH - cont'd

		A	D	D	R	CODE				
CODING SHEET	8	2	2	0	DB	IN	TIMI			Read timer
		1			15					It is stopped
		2			6F	MOV	L, A			because gate
		3			DB	IN	TIMI			input is low
		4			15					
		5			67	MOV	H, A			
		6			CD	CALL	HUNCP			Take hundreds
		7			40					complement for
		8			82					decimal time
		9			CD	CALL	DWORD			Display time
MICROCOMPUTER TRAINING SYSTEM	A				D1					
	B				02					
	C				C3	JMP	820C			Loop
	D				0C					
	E				82					
	F				00	NOP				
	8	2	3	0	DB	IN	PORT2B	WTHL		
		1			0D					Read interrupt
		2			E6	ANI	40			status byte and
		3			40					test EXT4 input
	4			CA	JZ	8230			at bit 6. Wait	
	5			30					until EXT4 high	
	6			82						
INTEGRATED COMPUTER SYSTEMS	8	2	3	7	DB	IN	PORT2B			Read interrupt
		8			0D					status byte
		9			E6	ANI	40			and wait until
		A			40					EXT4 is low
		B			C2	JNZ	8237			
		C			37					
		D			82					
		E			C9	RET				
		F			00	NOP				
		8			0					
	1									
	2									
	3									
	4									
	5									
	6									
	7									
	8									

FIGURE 3-146

HUNDREDS

COMPLEMENT - TWO BYTES

A	D	D	R	CODE						
8	2	4	0	3E	MVI	A,	9A			Represents 100
			1	9A						
			2	95	SUB	L				Subtract LSD
			3	C6	ADI	00				Add 0 to make
			4	00						DAA valid
			5	27	DAA					
			6	6F	MOV	L,	A			
			7	3E	MVI	A,	99			
			8	99						Subtract higher
			9	CE	ACI	00				digits from 99
			A	00						but add in CY
			B	94	SUB	H				from LSD
			C	C6	ADI	00				Add 0 to make
			D	00						DAA valid
			E	27	DAA					
			F	67	MOV	H,	A			
8	2	5	0	C9	RET					
			1							
			2							
			3							
			4							
			5							
			6							
			7							
			8							
			9							
			A							
			B							
			C							
			D							
			E							
			F							
			8	0						
			1							
			2							
			3							
			4							
			5							
			6							
			7							
			8							

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

ENTER WITH
 (HL) = DECIMAL VALUE
 RETURN
 (HL) = HUNDREDS COMPLEMENT
 (A) = HIGH BYTE

REGISTERS B, C, D, E
 PRESERVED

FIGURE 3-14c

3.6.2 Additional Exercises

Two other ways of recognizing the state of the input signal are possible. One method uses EXT 4 and EXT 5 interrupts but is merely a simple extension of the preceding program. The other reads the timer to determine whether it is running. You should develop the program of 3.6.2.2 yourself. Exercise 3.6.2.1 is optional.

3.6.2.1 Awaiting an Interrupt

EXERCISE

The program in figure 3-14 can be modified to use the EXT 5 interrupt in place of the WTHL wait subroutine at 8230H.

Connect an additional jumper from EXT4 OUT to EXT5 IN, and enable the EXT5 interrupt which will occur at the falling edge of the signal. We will load Timer 1 the first time this interrupt occurs, and then wait for a second interrupt. When that occurs we will read the timer, which will have counted down during the time that the signal pulse was high.

The processor can be forced to stop operations by the HLT (76H) instruction. When this is encountered in a program the processor enters a wait state, and does not execute any further instructions until an interrupt occurs. At this time the interrupt service routine is executed and then control returns to the next instruction following the HLT.

Replace the WTHL subroutine with an interrupt service routine that does the following:

```
Save PSW
Reenable and clear the interrupt
Restore the PSW
EI
Return
```

To enable the interrupt initially, call this service routine instead of WTHL. Follow the call by HLT to wait for the first falling edge. Since RST6 is exactly equivalent to CALL 8230, you can insert both of these instructions and a NOP in place of CALL WTHL.

Now replace the second CALL WTHL by HLT, NOP, NOP. This will cause the processor to wait for the second falling edge. The remainder of the main program is unchanged.

3.6.2.2 Reading an Active Timer

EXERCISE

You can re-code the original program (figure 3-14) to read the timer while it is running.

Use the original WTHL subroutine to detect the first falling edge (The timer interrupt should not be enabled). In place of the second call to WTHL, call a new subroutine that does the following:

```
    Read Timer 1 (both bytes)
    Save the result
    Read Timer 1 again (both bytes)
    Compare with previous result
    Repeat until the result changes, indicating
        that the timer is running
    Repeat until the result no longer changes,
        indicating that the timer has stopped
```

NOTE: use of an internal subroutine may shorten this program to less than 30D bytes.

3.6.3 Reading While Counting

In the exercise of 3.6.1, the timer is read only when counting has been inhibited by a low input at the gate. In 3.6.2.2, the counter is read while it is counting. The final measurement which is displayed was taken after counting stopped.

The IOR signal that places input data on the data bus extends across at least one phase 2 clock cycle. In the MTS, which inserts a wait state in every memory or I/O cycle, it extends across two clock pulses. Since the timer runs from the phase 2 clock, it is guaranteed that the two lowest bits will change during the time that the counter outputs are driving the bus, and possible that all 16 bits will change. The data thus received by the 8080 while the data bits are changing must be considered garbage. The 8253 provides a facility for accurately reading a timer while it is counting. There is one 16 bit register which can be synchronously loaded with the content of any one counter, upon command from the processor. A subsequent IN (or two IN's for two bytes) addressed to the same counter will access the latching register rather than the counter itself. The latching control bytes were included (though not defined) in Figures 3-3 and 3-4.

Timer Control Byte Digits

Control Byte				
Binary	Hex	Timer		Operation
0000	XXXX	00	0	Copy timer into latching
0100	XXXX	40	1	register before reading
1000	XXXX	80	2	

Like the mode set control bytes, these are sent by OUT TIMCT. To read a running timer that is programmed for two byte read and load the following sequence is used.

```

3E   MVI  A, 40           Latch control byte for timer 1
40
D3   OUT  TIMCT          Write to timer control
17
DB   IN   TIM1           Read latched data from timer 1
15
6F   MOV  L,A
DB   IN   TIM1           Store in (HL)
15
67   MOV  H,A

```

Note that the IN instructions are still addressed to timer 1, and two reads are still required if the timer is programmed for two byte load and read.

EXERCISE

Develop a program to demonstrate that invalid data may be read from a running counter if the latching operation is not used.

3.7 MODE 2 - RATE GENERATOR

Probably the most common use of the interval timer is generation of a signal or an interrupt at precisely repeated intervals. This is useful for:

- * Generating a slower clock for an external device that cannot use the 2.048 MHz system clock.

- * Measuring times or generating timing functions too great for the 32 millisecond capacity of a 16 bit counter.

- * Servicing inputs or outputs on a schedule rather than by interrupts.

- * Keeping track of real time.

3.7.1 Use of Mode 2

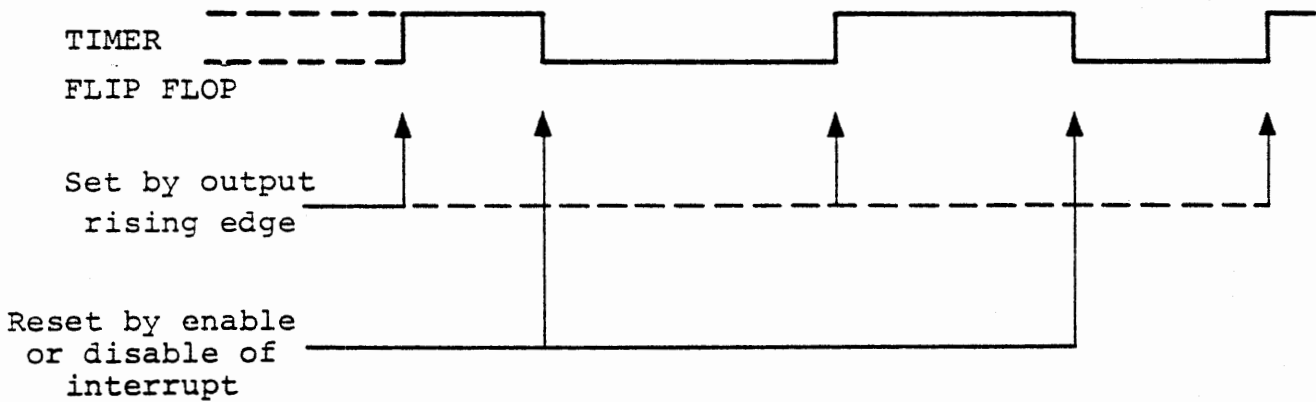
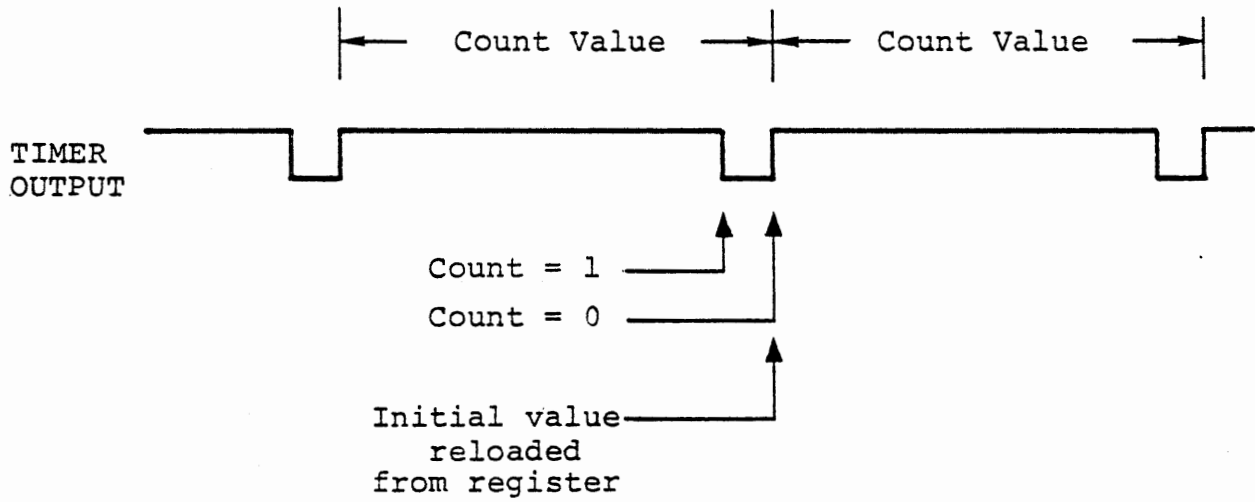
Mode 2 is programmed by writing a control byte to the timer control port in accordance with Figure 3-4. For instance, to program timer 1 for a two byte load, mode 2, binary:

```
3E MVI A, 74
```

```
74
```

```
D3 OUT TIMCT
```

```
17
```

TIMER AND FLIP FLOP OPERATION

MODE 2 - RATE GENERATOR

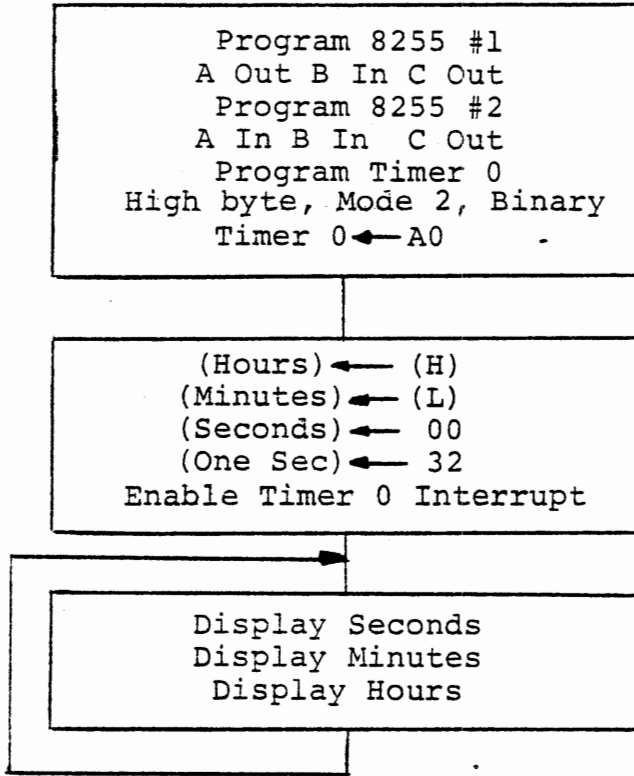
Figure 3-15

This immediately sets the output high if it was not high. Counting starts when a count value is loaded, provided the gate input is high. Counting is inhibited if the gate input is low.

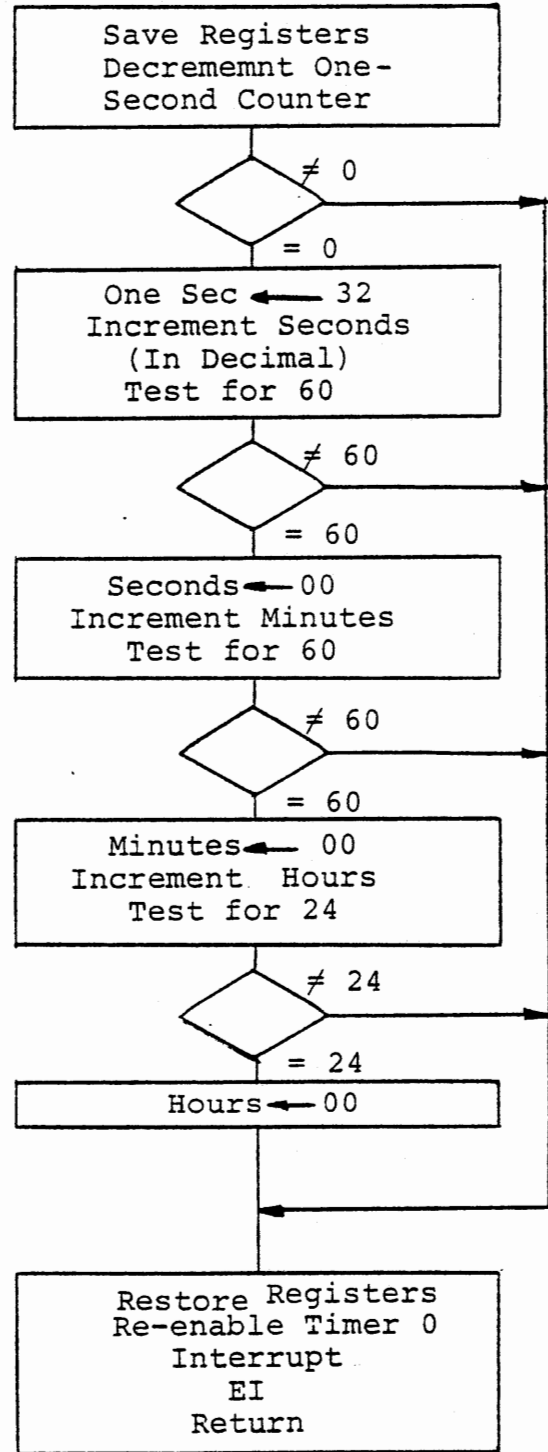
The output remains high while counting, until the count value reaches 0001, when the output goes low. At the next falling edge of the clock, the output goes high and the initial value is reloaded from the count register into the counter. Thus, a half microsecond pulse is output once for each counting cycle. The timer need not be reloaded, and it will give the pulse at a precisely repeated interval even if the interrupt service is delayed.

Figure 3-15 shows the relationship between the timer output and its flip flop. Note that the half microsecond pulse from the timer, if it were directly connected to interrupt request, might or might not generate an interrupt, depending on the state of the 8080 at that moment. Therefore, the interrupt must be taken from the flip flop of timer 0 or timer 1. Timer 2 cannot reliably generate an interrupt in mode 2 (nor in modes 4, 5 or 6, for the same reason).

After the flip flop has generated an interrupt it must be reset by setting or resetting the corresponding enable bit at port 2C0 or 2C1, before an EI instruction is given. Otherwise repeated interrupts will be generated, filling the stack until the program is destroyed.



MAIN TIME DISPLAY PROGRAM



RST 5 INTERRUPT SERVICE

TIME OF DAY CLOCK

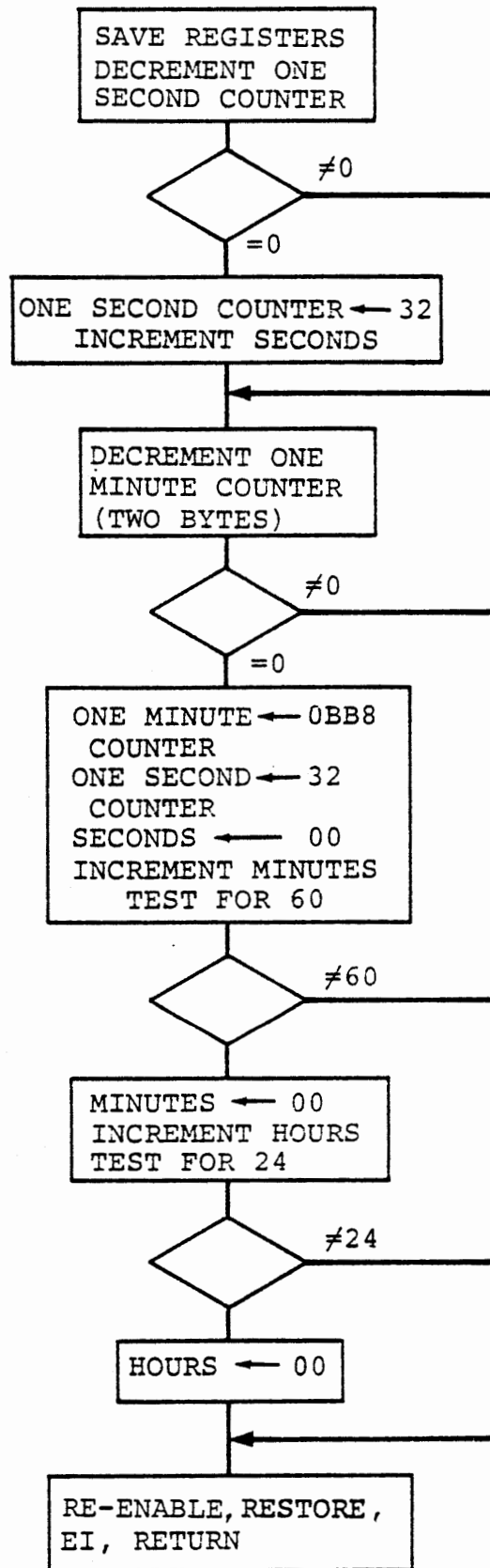
Figure 3-16

3.7.2 Real Time Clock

EXERCISE

Develop a program that will keep and display the time of day. The flow diagram of Figure 3-16 displays hours, minutes and seconds. Timer 0 generates an interrupt every 20 milliseconds and a software counter is decremented from 32 (=50 decimal). At zero a seconds counter is incremented (in decimal). At 60 seconds a minutes counter is incremented and at 60 minutes an hours counter is incremented. The display function is handled by the main program. This would permit another program to operate in conjunction with the time of day. It can use the keyboard and display, and when nothing else is going on the time can be displayed.

In this program the starting time (in hours and minutes) is loaded from the content of H and L. Use the monitor to place the time in those registers and press RUN when the second hand of your watch reaches zero. Test the timekeeping. You will probably find this clock to be quite inaccurate because the crystal of the MTS is only accurate to 0.1%. This gives an error of 86 seconds a day. The clock can be made somewhat better by using a separate software counter for one minute, with an initial value of about 0BB8 (= 3000). This can be adjusted for crystal frequency error; allowing the clock error to be less than 30 seconds a day. Figure 3-18 shows a flow diagram for this clock. A further improvement can be made with a one hour counter, with a nominal initial value of 02BF20 (180000 decimal). This permits an adjustment to less than half a second per day if the



RST 5 INTERRUPT SERVICE

Figure 3-18

crystal is sufficiently stable.

You may want to elaborate the clock program for the fine adjustment, or to load time of day by keyboard entry, or to keep date as well as time with adjustments for 28, 29, 30, or 31 days. (Remember that leap year is omitted every 100 years but included every 400 years, so 29 February 2000 will exist).

TIME OF DAY - INITIALIZE

3 - 43

	A	D	D	R	CODE					
CODING SHEET	8	2	0	0	3E	MVI	A,	82		Program 8255#1
					82					A OUT BIN COUT
					D3	OUT		CNT1		
					07					
					3E	MVI	A,	92		Program 8255#2
					92					A IN BIN COUT
					D3	OUT		CNT2		
					0F					
					3E	MVI	A,	24		Program Timer 0
					24					Load high byte
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT		TIMCT		Mode 2, Binary
	B				17					
	C				3E	MVI	A,	A0		Timer 0 ← A000
	D				A0					for 20 millisecond
	E				D3	OUT		TIMO		interrupt
	F				14					
	8	2	1	0	EB	XCHG				(DE) ← initial time
					21	LXI	H,	8300		Address software
					00					counters
					83					
INTEGRATED COMPUTER SYSTEMS					36	MVI	M,	32		(8300) ← 32 for
					32					one second counter
					23	INX	H			(8301) ← 00
					36	MVI	M,	00		for 60 second
					00					counter
					23	INX	H			(8302) ← (E)
	A				73	MOV	M,	E		for minutes
	B				23	INX	H			(8303) ← (D)
	C				72	MOV	M,	D		for hours
	D				EF	RST 5				Programmed call
E				C3	JMP		8260		to interrupt	
F				60					service to enable	
8	2	2	0	82					The interrupt	

Figure 3-17a

TIME OF DAY - PROGRAM LOOP 3 - 49

A D D R		CODE					
CODING SHEET	8	26 0	21	LXI	H,	8301	Address 60
		1	01				second counter
		2	83				
		3	CD	CALL	DMEM		Display seconds
		4	94				
		5	02				
		6	23	INX	H		Address minutes
		7	7E	MOV	A, M		
		8	CD	CALL	DBY 2		Display
		9	98				
MICROCOMPUTER TRAINING SYSTEM	A	02					
	B	23	INX	H		Address hours	
	C	7E	MOV	A, M			
	D	CD	CALL	DBY 2		Display	
	E	98					
	F	02					
	INTEGRATED COMPUTER SYSTEMS	8	27 0	C3	JMP	8260	Loop
		7 1	60				
		7 2	82				
		3					
		4					
		5					
		6					
		7					
		8					
		9					
	A						
	B						
	C						
	D						
	E						
	F						
	8	0					
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						

Figure 3-17b

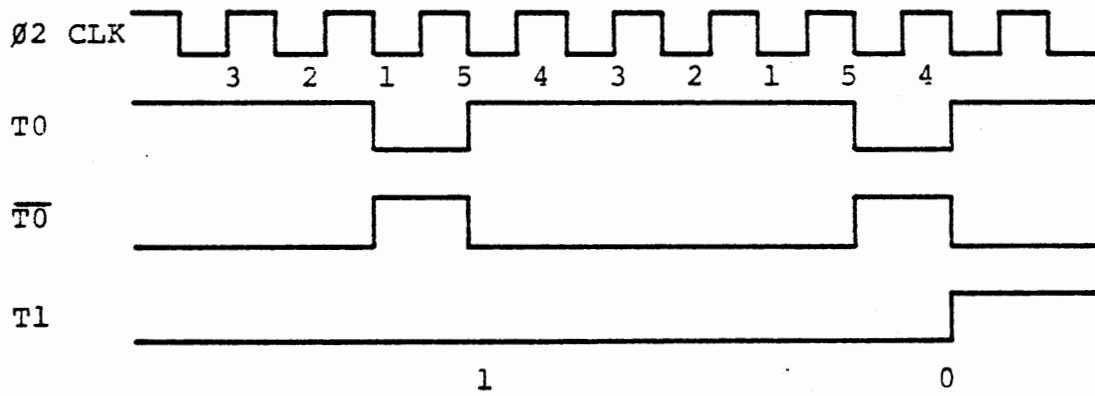
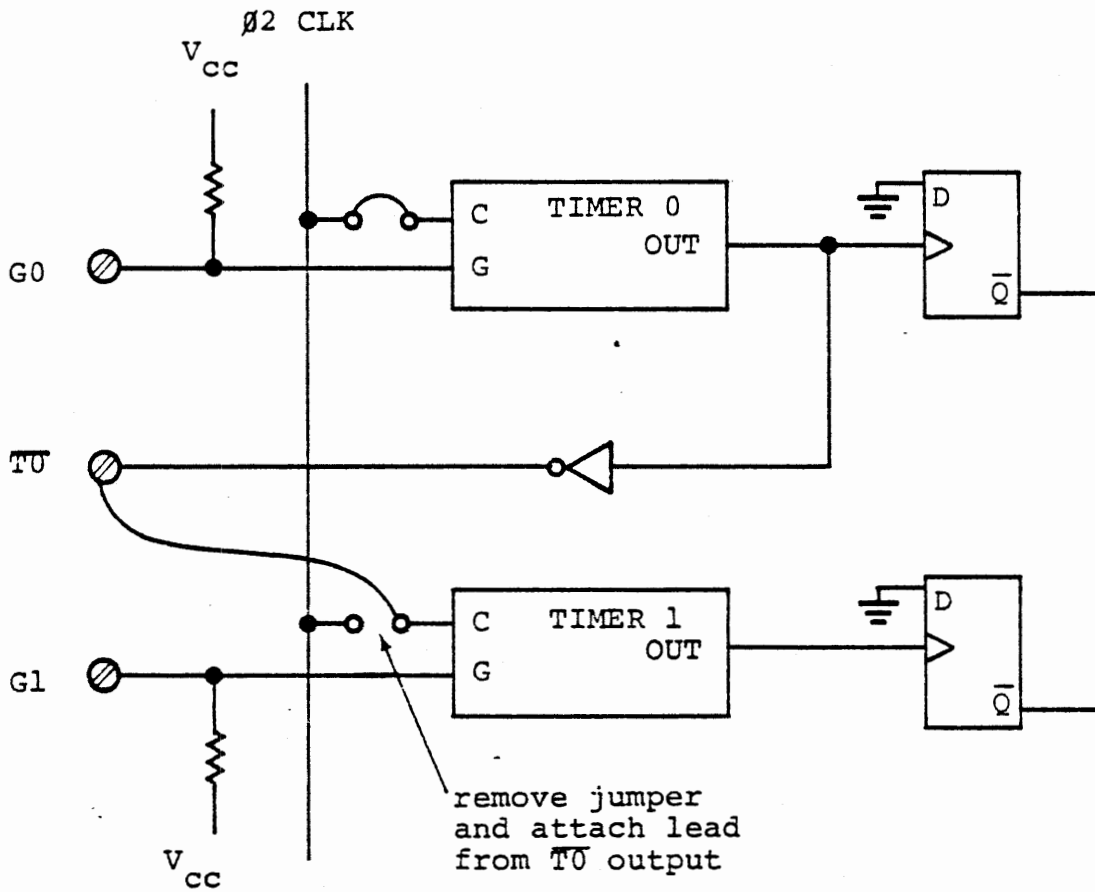
TIME OF DAY RST 5 INTERRUPT SERVICE 3 - 50

A D D R		CODE										
8	0											
	1											
	2											
	3											
	4											
	5											
	6											
	7											
CODING SHEET	822	8	FS	PUSH	PSW						Save registers	
		9	ES	PUSH	H							
		A	21	LXI	H, 8300						Address one	
		B	00								second counter	
		C	83									
		D	C3	JMP	8233							Jump past RST6
		E	33									interrupt location
MICROCOMPUTER TRAINING SYSTEM		F	82									
	823	0	E7	RST	4						No RST6 interrupt	
		1	FB	EI							should occur.	
		2	C9	RET							Leave space for JMP	
	823	3	35	DCR	M						Decrement one	
		4	C2	JNZ	8248						second counter	
		5	48									
MICROCOMPUTER TRAINING SYSTEM		6	82									
		7	36	MVI	M, 32						Restart one second	
		8	32								counter	
		9	3E	MVI	A, 59						Max value for	
		A	59								60 second counter	
		B	CD	CALL	8250						Subroutine increments	
		C	50								or clears counter	
INTEGRATED COMPUTER SYSTEMS		D	82									
		E	3E	MVI	A, 59						Max value for	
		F	59								60 minute counter	
	8	0										
		1										
		2										
		3										
	4											
	5											
	6											
	7											
	8											

Figure 3-17c

		TIME	OF DAY	RST	INT	SVC	AND	SUBR	
A	O	D	R	CODE					3 - 51
CODING SHEET	8	2	4	0	CC	CZ	8250		Call if 60 second
				1	50				counter was 59,
				2	82				to increment 60
				3	3E	MVI	A, 23		minute counter
				4	23				Max value for
				5	CC	CZ	8250		24 hour counter
				6	50				Call if 60 minute
				7	82				counter was 59
MICROCOMPUTER TRAINING SYSTEM	824			8	3E	MVI	A, 01		Reenable timer 0
				9	01				interrupt to
				A	D3	OUT	CNT2		clear flip flop
				B	0F				
				C	E1	POP	H		
				D	F1	POP	PSW		
				E	FB	EI			
				F	C9	RET			
MICROCOMPUTER TRAINING SYSTEM	825			0	23	INX	H		Subroutine -
				1	96	SUB	M		Address next
				2	CA	JZ	8259		counter and test
				3	59				for maximum value
				4	82				
				5	7E	MOV	A, M		If not maximum
				6	C6	ADI	01		increment in
				7	01				decimal
INTEGRATED COMPUTER SYSTEMS				8	27	DAA			
	825			9	77	MOV	M, A		
				A	C9	RET			
				B					
				C					
				D					
				E					
				F					
INTEGRATED COMPUTER SYSTEMS	8			0					
				1					
				2					
				3					
				4					
				5					
				6					
				7					
			8						

Figure 3-17c



CASCADED TIMERS

Figure 3-19

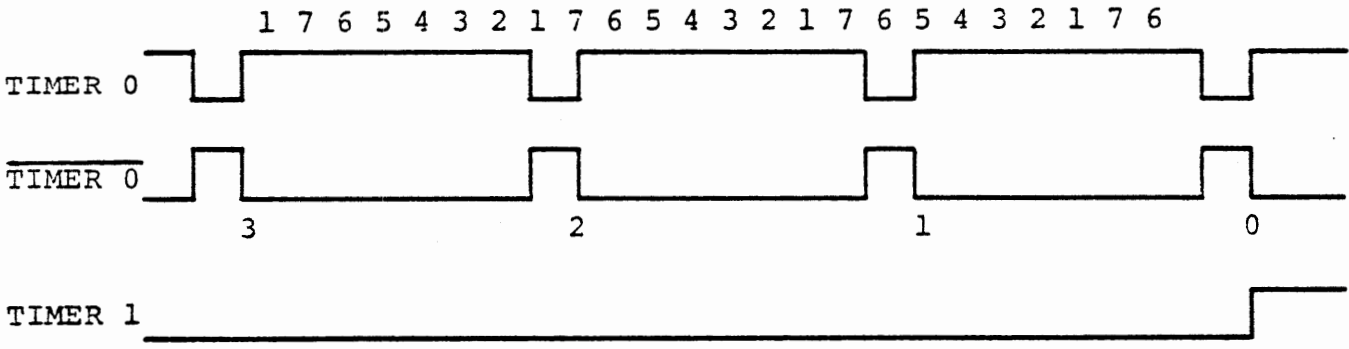
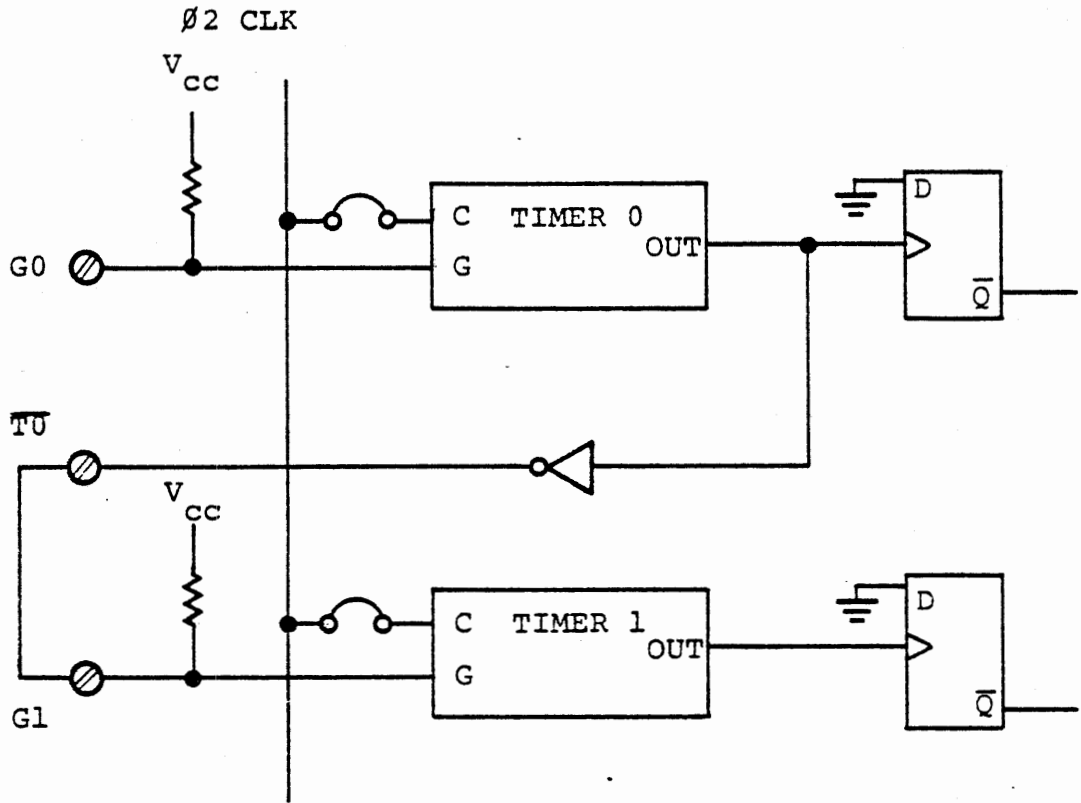
3.8 CASCADED TIMERS

It is possible to use the output of one timer to control another, in either of two ways. One output can provide a clock to another timer, but on the experiment board this requires disconnecting the system clock from the second timer as shown in Figure 3-19. With this connection two timers in mode 2 can be cascaded to generate a long time interval:

Capacity	32 bits
Maximum Count	4,294,967,295 (decimal)
Time	2,097.152 second
	= 34 minutes 57.152 seconds

A simpler connection, but with more restricted use, is to use the first timer output as a gate input to the second timer, as shown in Figure 3-20. Now if timer 0 is programmed to mode 2 its inverted output will enable the gate of timer 1 for exactly one clock pulse in each full count cycle of timer 0.

This is effective only if timer 0 is in mode 2, giving one pulse each count cycle, and timer 1 is in mode 0 or mode 4. In all other modes, the gate input rising edge restarts the counter by reloading it with the initial value from its storage register, so cascading can only be done with the clock input.



TIMER 0 GATING TIMER 1
 TIMER 0 IN MODE 2
 TIMER 1 IN MODE 0

CASCADING TIMERS WITH GATE INPUT

Figure 3-20

EXERCISE

We will use the simpler connection from \overline{TO} OUT (at left of ITS board) to G1 IN to gate the second timer. In the program of Figure 3-21, we accept keyboard data for a time delay to be loaded to timer 1, which is in mode 0 and gated by timer 0. At the interrupt from timer 1 we shift a bit in the LED display as a visual indication.

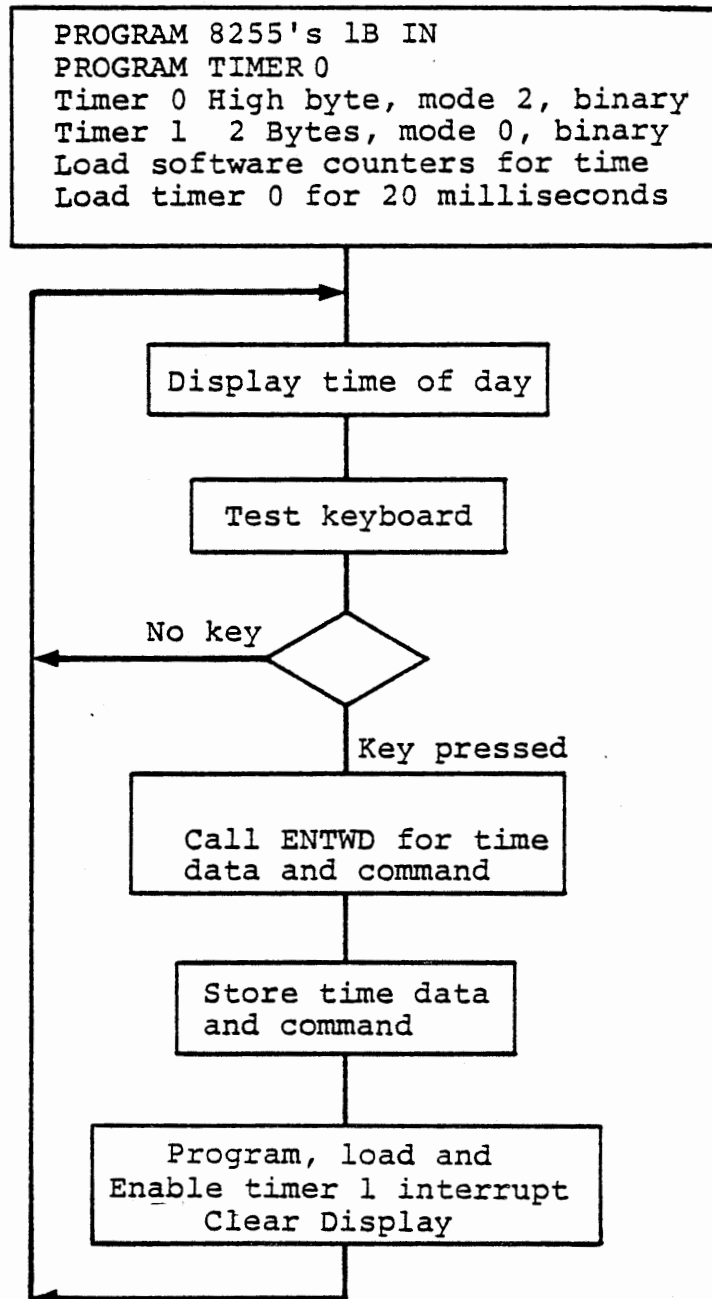
If the STEP key is pressed following the numeric data, the interrupt service routine disables the timer 1 interrupt, which is not restarted until a new keyboard entry is given. If the RUN key is pressed following the numeric data, then interrupt service reloads timer 1 and reenables the interrupt.

This program is designed to work concurrently with the time of day display of Figure 3-16 or 3-18. When no keyboard entry is made, the time of day display is shown. While ENTWD is accepting keyboard data, it controls the display.

The effect of STEP and RUN commands here is analagous to mode 0 and mode 2 in the timers. With STEP timer 1 is decremented to zero and interrupts only once, like mode 0. With RUN it is reloaded and restarted each time it reaches zero.

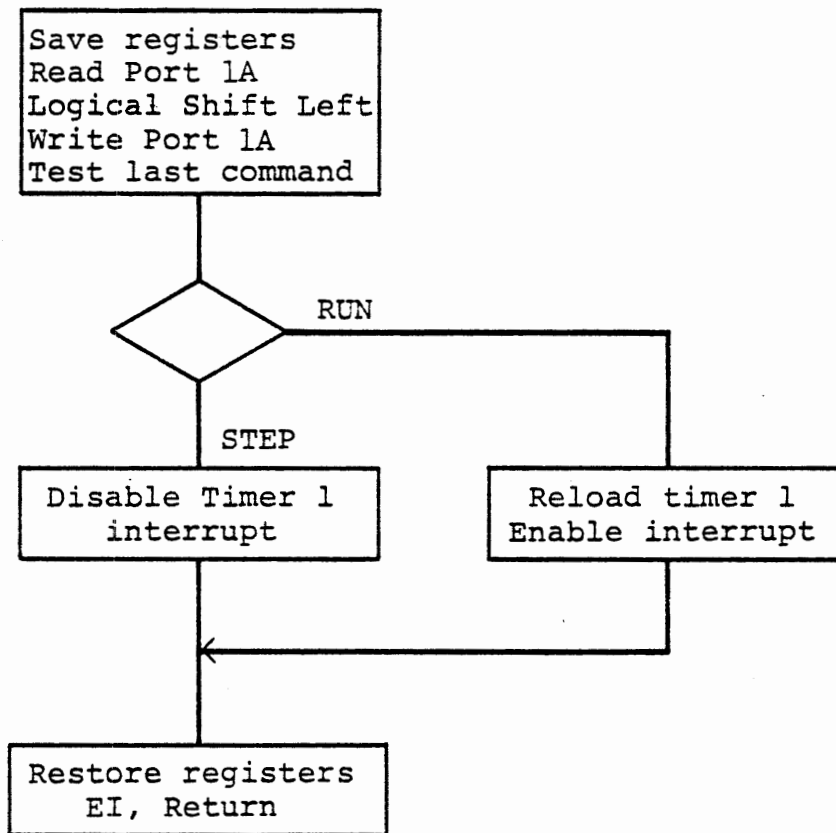
This program can be instructive in other ways. Note that in the solution given we load timer 1 and then enable (or disable) its interrupt. If the time delay loaded is 0002, the RST6 interrupts will occur frequently. If the time loaded is 0000, the interrupts will occur very infrequently (once ever 1310 seconds). If a value of 0001

is entered, no interrupts will occur at all. With this initial value the interval timer will not function correctly!



TIME DELAY PROGRAM - MAIN

Figure 3-21a



INTERRUPT SERVICE FOR TIMER 1

Figure 3-21b

REVISED CLOCK CONTINUED

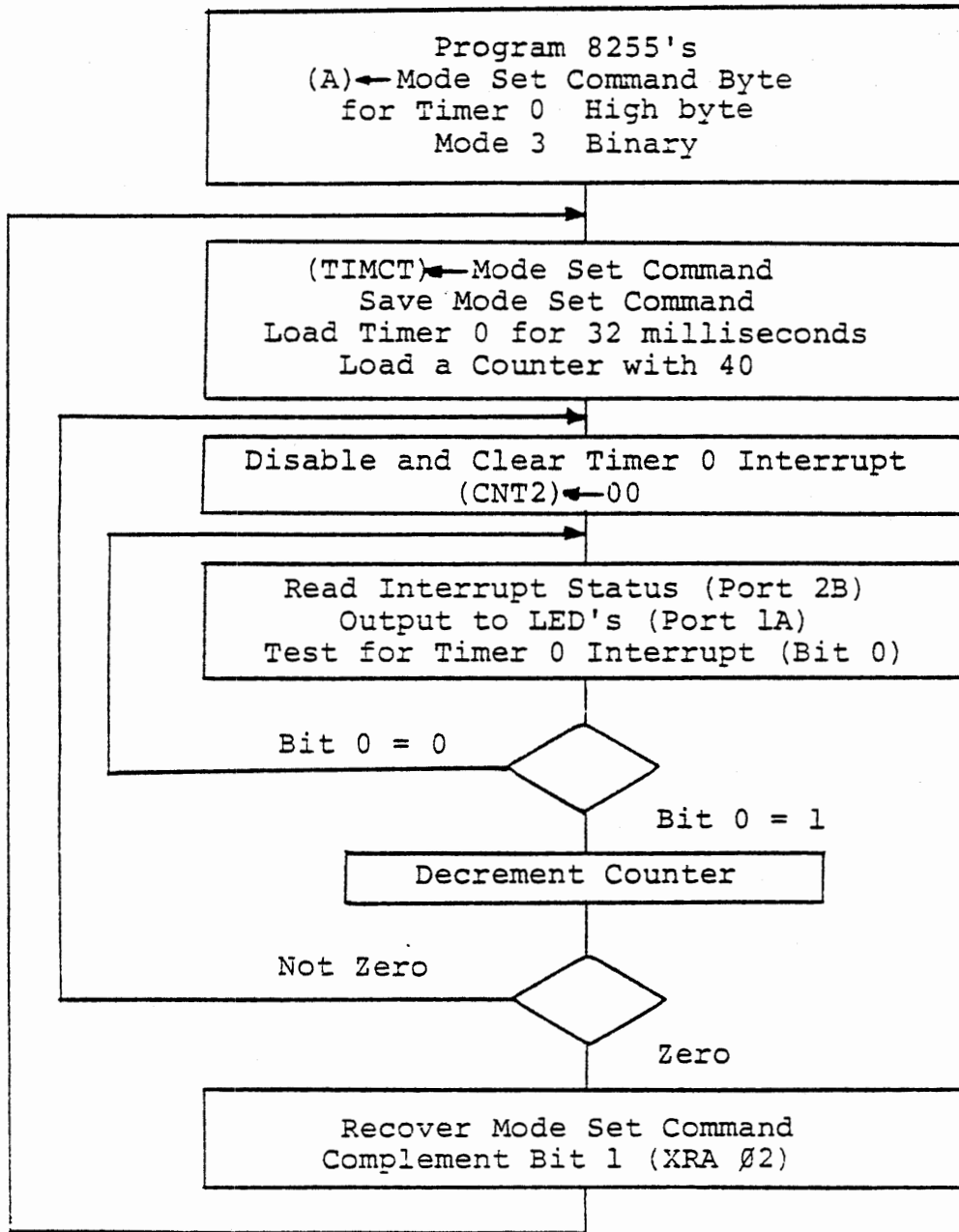
		A	D	D	R	CODE					
CODING SHEET	8	2	8	0		3E	MVI	A, 01			Set right LED on
						01					All others off
						D3	OUT	PORTIA			
						04					
						37	STC				Flag to enable
						CD	CALL	8290			Subr programs,
						90					loads, and enables
						82					Timer 1
						CD	CALL	CLEAR			Monitor subroutine
						87					clears display
MICROCOMPUTER TRAINING SYSTEM	A					02					
	B					C3	JMP	8260			Back to time display
	C					60					
	D					82					
	E										
	F										
	8	2	9	0		3E	MVI	A, 70			Subroutine to
						70					program, load
						D3	OUT	TIMCT			and enable timer 1
						17					Both bytes
					7D	MOV	A, L			Mode 0	
					D3	OUT	TIMI			Binary	
					15						
					7C	MOV	A, H				
					D3	OUT	TIMI				
					15						
INTEGRATED COMPUTER SYSTEMS	A					3E	MVI	A, 01			(A) ← 03 to enable if CY set
	B					01					
	C					17	RAL				(A) ← 02 to disable if CY clear
	D					D3	OUT	CNTZ			
	E					0F					
	F					C9	RET				
	8					0					
						1					
						2					
						3					
					4						
					5						
					6						
					7						
					8						

FIGURE 3-21d

RST6 INTERRUPT SERVICE

	A	D	D	R	CODE						
CODING SHEET	8	2	3	0	C3	JMP	82A0				Patch to
				1	A0						Figure 3-17b
				2	82						
				3							
				4							
				5							
				6							
				7							
				8							
				9							
MICROCOMPUTER TRAINING SYSTEM	A										
	B										
	C										
	D										
	E										
	F										
	8	2	A	0	F5	PUSH	PSW				
				1	E5	PUSH	H				
				2	DB	IN	PORTA				
				3	04						
				4	07	RLC					
				5	D3	OUT	PORTA				
				6	04						
				7	2A	LHLD	8305				(HL) ← time delay
				8	05						
				9	83						
	A			3A	LDA	8304				(A) ← last command	
	B			04							
INTEGRATED COMPUTER SYSTEMS	C				83						
	D				C6	ADI	EC				Set carry if RUN
	E				EC						Clear carry if STEP
	F				CD	CALL	8290				To enable timer 1
	8	2	B	0	90						if RUN, disable if
				1	82						STEP.
				2	E1	POP	H				
				3	F1	POP	PSW				
				4	FB	EI					
				5	C9	RET					
				6							
				7							
				8							

FIGURE 3-21e



Square Waver Generator - Mode 3

Figure 3-22

3.9 MODE 3 SQUARE WAVE GENERATOR

When programmed in mode 3, a timer repeatedly counts down from its initial value, starting with its output high. Halfway through the count, the output goes low. At zero, the output goes high and the initial value is reloaded from the count register. Thus a square wave is generated. If the initial value is an odd number, the first half of the count will be one bit time longer than the second half.

The gate input disables counting when it is low. At a rising edge of the gate input, the initial value is reloaded from the count register into the counter. The output becomes high and a new complete cycle starts.

If the count register is reloaded while the timer is running in this mode, the current half period of counting will be completed with the old value. The new value will become effective when the output changes in either direction, or at a rising edge of the gate input.

3.9.1 Observing the Output

EXERCISE

Write a program that will change the mode of timer 0 every few seconds, alternating between mode 2 and mode 3 (figure 3-22 shows a flow diagram). Observe the inverted output TO OUT in one of these ways:

- a) With an oscilloscope
- b) With a voltmeter

- c) By connecting $\overline{\text{TO}} \text{ OUT}$ to EXT4 IN, reading Port 2B and displaying its data in the LEDs.

An oscilloscope permits direct observation of the inverted square wave at $\overline{\text{TO}} \text{ OUT}$ and additional experiments. The voltmeter across $\overline{\text{TO}} \text{ OUT}$ and GND will show a low output (0.4 volts) when the timer is running in mode 2, but about 2 volts in mode 3. With the jumper connected (as in 'c' above), LED DS6 will be visibly illuminated in mode 3 but not in mode 2.

3.9.2 Observing the Counting

The square wave generator conceivably could operate in any one of three ways

- (a) Divide initial value by 2 before loading the counter from the count register. Toggle the output and reload at zero.

- (b) Load the counter with the initial value, and decrement by 2 at each clock. Toggle the output and reload at zero.

- (c) Compare the counter content with half of the initial value, and set the output low at equal. Set the output high at zero.

The following exercise permits you to determine which of these is actually used.

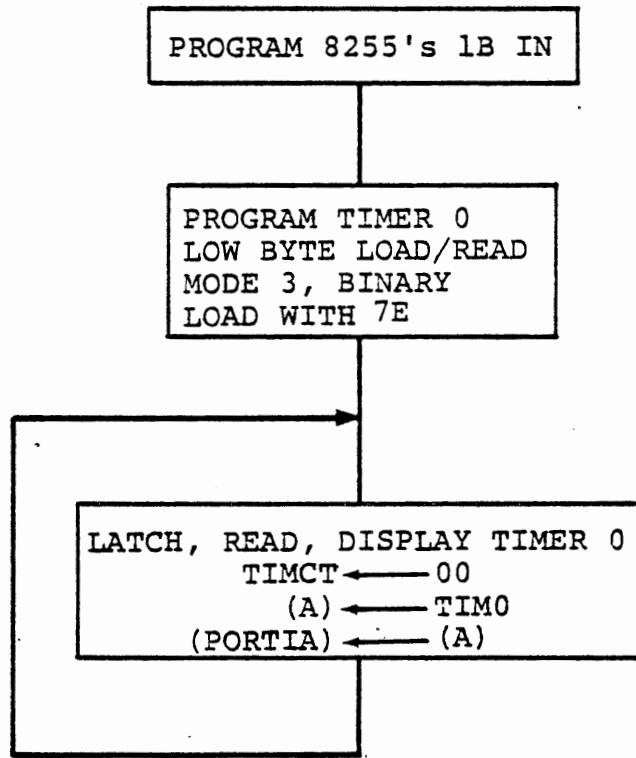
EXERCISE

Program a timer for mode 3 operation, low byte only. Load it with 7E. In a loop, repeatedly latch and read the timer while it is counting. Display the byte in the LEDs of port 1A. Determine from this how the timer really operates in mode 3.

If (a) is true, the LEDs will never show a value greater than half of the initial value.

If (b) is true, the least significant bit will never change.

If (c) is true, the full value will be shown and the least significant bit will count.



READING THE TIMER CONTENTS

Figure 3-23

3.10 TIMER MODE DESCRIPTIONS

This section defines all six modes of the timer, including modes 0,2 and 3 which have previously been discussed as well as the three modes that have been neglected.

The modes differ principally in the effect of the gate input and the behavior of the output.

<u>Modes</u>	<u>Gate Input</u>		
	<u>Low</u>	<u>Rising Edge</u>	<u>High</u>
0,4	Disables Counting		Enables Counting
2,3	Disables Counting	Reloads counter with initial value and initiates counting.	Enables Counting
1,5		Reloads counter with initial value and initiates counting	

<u>Modes</u>	<u>Output Signal</u>
0,1	Low while counting High at count = 0
2,4,5	High while counting Low for one clock period
3	High during first half cycle Low during second half cycle

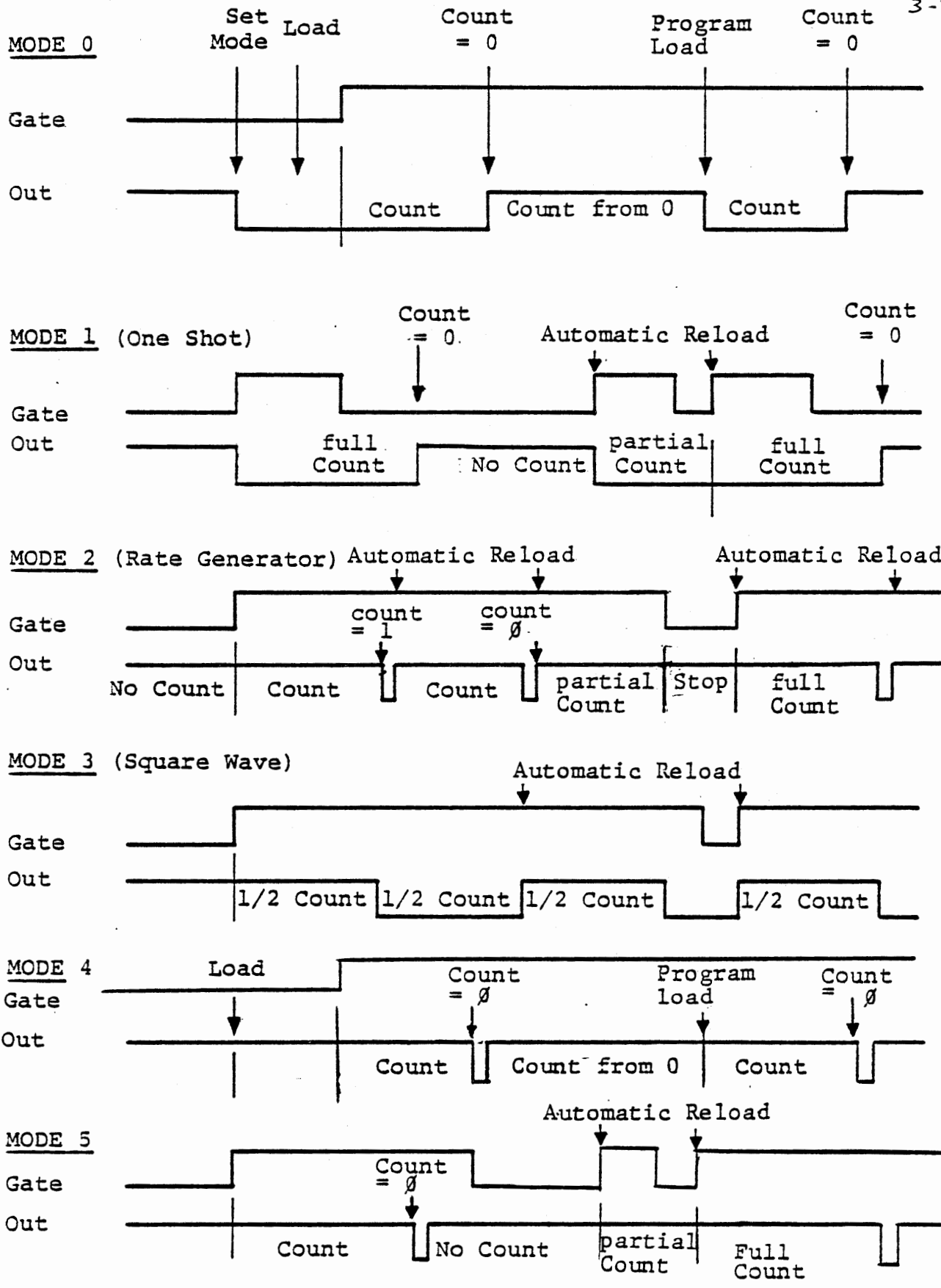
<u>Modes</u>	<u>After Terminal Count</u>
0,4	Counting continues but output remains high
1,5	Counting stops until a new gate rising edge occurs
2,3	Counting starts again from the initial value and output pattern repeats for each full count cycle.

Figure 3-24 shows more detail of the gate effect and output timing, and the following sections define each mode in detail. Figure 3-25 indicates the timing relationships. Note that mode 0 and mode 4 are similar except for the output state during counting, but for a given count loaded to the timer, mode 4 will generate an interrupt one clock time later than mode 0. The same relationship is true of modes 1 and 5.

Mode	Output after mode set	Starts counting	Output goes low	Output goes high	Count restarted	Comments
0 Interrupt	Low	When final byte loaded	At mode set	At zero	By reloading	Output is set low by setting mode or by reloading.
1 One Shot	High	After gate rising edge	After gate rising edge	At zero	By gate rising edge	Can be preloaded during counting. Present period not affected. New value effective for next period.
2 Rate Generator	High	When final byte loaded	At count=1	At zero	At zero or by gate rising edge	
3 Square Wave	High	When final byte loaded	At $n/2$ or $(n + 1)/2$	At zero	At zero or by gate rising edge	If loaded while counting new period is effective for next half of total period.
4 Software Strobe	High	When final byte loaded	At zero	At next clock after zero	By reloading	
5 Hardware Strobe	High	After gate rising edge	At zero	At next clock after zero	By gate rising edge	If loaded while counting new period is effective after next gate rising edge.

8253 TIMER MODES

Figure 3-24



TIMING RELATIONSHIPS

Figure 3-25

Mode 0 Interrupt on Terminal Count

The timer counts down from the initial value, continues from zero. The output goes low when mode 0 is set, high when the count reaches zero. Counting starts when the final byte of the initial value is loaded. If a new value is loaded during counting, loading the first byte stops the count and sets the output low. Mode 0 is useful for generating a single time delay function or for measuring time from a programmed or external event, providing that the time is less than the 32 millisecond capability of the 16 bit counter. It can be used to measure the duration of an external signal, since counting is enabled only when the gate input is high.

Mode 1 Programmable One Shot

Starts counting down from the initial value after a rising edge of the gate input. The output goes low at the first count after the gate rising edge, high at zero. Counting starts again from the initial value each time a rising edge occurs at the gate input. Mode 1 is useful for generating a time delay or measuring time from an external event, especially if the external event is a narrow pulse.

Mode 2 Rate Generator

The output goes high when the mode is set. After the count has been loaded, the timer will repetitively count down from the initial value to zero. The output goes low when the count reaches one and high when it reaches zero, so a 0.5 microsecond pulse is generated. Mode 2 is especially useful for timing functions where software counters are to be used for times greater than the 32 millisecond capacity of the timers. Counting restarts from the initial value immediately after zero is reached, so a delay before the program services the counter does not introduce any uncertainty in the timing. If the counter register is reloaded during counting, the present period is not affected, but the new value is effective for subsequent periods. The gate input inhibits counting when it is low. A rising edge restarts the counter from the initial value.

Mode 3 Square Wave Rate Generator

The output goes high when the mode is set. After the count is loaded, the timer will repetitively count down from the initial value to zero. The output will go low when the count reaches half the initial value and high when the count reaches zero, so a square wave is generated. If the initial value is odd, the output will be high for $(n+1)/2$ counts and low for $(n-1)/2$ counts. If the counter register is reloaded during counting, the present half cycle is not affected, but the new value is effective for the next half cycle and subsequent periods. The gate input inhibits counting when it is low. A rising edge restarts the counter from the initial value.

Mode 4 Software Triggered Strobe

The timer counts down from the initial value and continues from zero. The output goes high when the mode is set, low the first time the count reaches zero, then high at the next clock pulse after zero. If the counter register is reloaded during counting, the present period is not affected, but the new count starts immediately after zero is reached. Therefore, mode 4 is especially useful when successive delays of different durations are required. The gate input inhibits counting when it is low.

Mode 5 Hardware Triggered Strobe

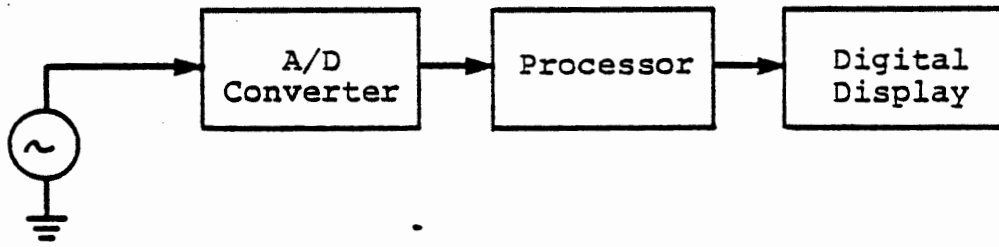
Starts counting down from the initial value after a rising edge of the gate input. The output goes high when the mode is set, low when the count reaches zero, then high at the next clock pulse after zero. Counting starts again from the initial value each time a rising edge occurs at the gate input. If the count register is reloaded during counting, the present period is not affected. The new count is effective when the next gate rising edge occurs.

MICROCOMPUTER INTERFACING WORKBOOK

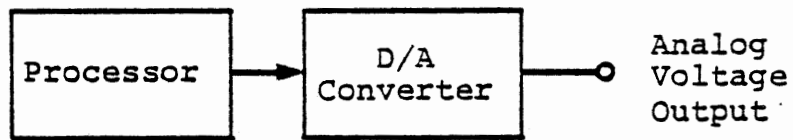
CHAPTER 4

DIGITAL TO ANALOG OUTPUT

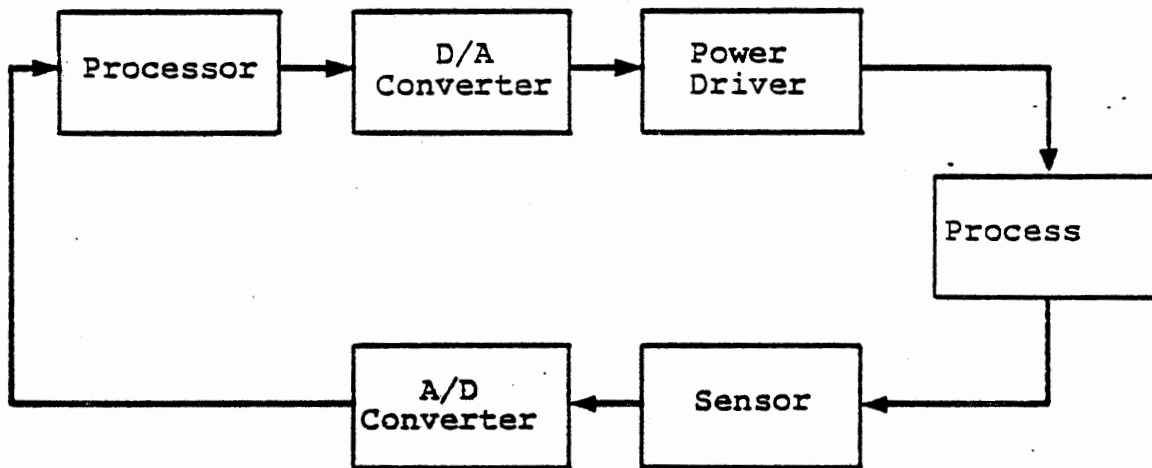
DIGITAL VOLTMETER



Function Generator



Closed Loop Process Control



A/D AND D/A CONVERSION

Figure 4-1

4. DIGITAL TO ANALOG OUTPUT

Very commonly a computer or microprocessor in a control system must generate an analog output - a signal which represents some value in a continuous range of values, rather than a binary 0 or 1. An analog signal may be a variable voltage which is the output of a measuring instrument or the input to a control driver: the voltage is like the rate of flow, the temperature, the position, that is being measured or controlled; it is an analog of the real variable.

In this chapter we will experiment with several means by which the computer can generate an analog signal. In chapter 5, we will investigate the opposite task, of converting an analog signal into a digital form that the computer can process. Instruments usually have a one-way conversion, but control systems very often need both, as suggested in Figure 4-1.

4.1 Methods of D/A Output

Although a variable voltage is one of the most common analogs, there are many others, each having particular advantages in appropriate circumstances. We will discuss some of these in the next chapter, which deals with analog input; here we introduce the few that are especially suited for analog output from the computer.

It is not always clear where analog conversion ends. A computer might output a set of binary data which is converted to a voltage input to an op-amp, which drives a power transistor, whose output current controls a hysteresis motor that generates a torque to precess a gyro. The ultimate conversion was from digital data to a new position for the guidance gyro: en route, we have had a voltage, a current, magnetic flux, magnetic force, mechanical force and a precessing torque; each of these was an analog of the desired motion.

There are two basic approaches that can be used for output from the digital processor to give a variable value: the output may be several binary signals representing a number that is converted to a voltage or current; or the output may be a single bit with time as the continuous variable, so that either frequency or average power output is the analog. In the latter case some external device, other than the load which is being driven, must integrate the signal. For example, the binary signal may turn a heater on and off to maintain a desired average temperature; a bimetal thermostat controlling a household furnace does exactly that, and the building integrates the binary (on and off) condition of the furnace.

In this chapter, we will consider four digital to analog conversions:

Pulse width modulation

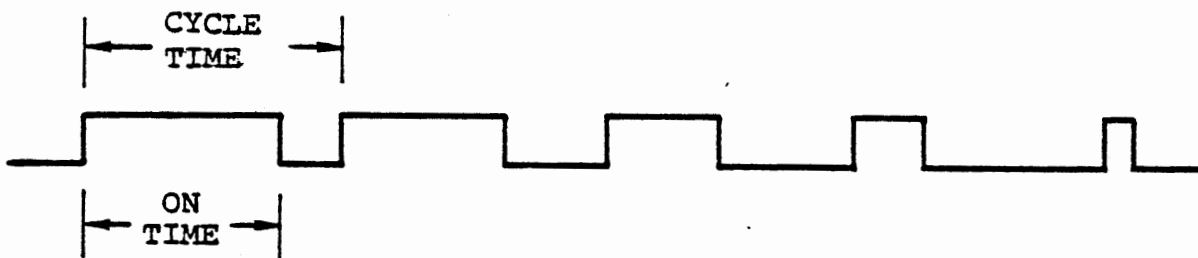
Frequency modulation

Direct multi-bit output

Ladder network digital to voltage conversion

4.2 Pulse Width Modulation

Pulse width modulation (pulse duration modulation or pulse-time modulation) is a technique to vary average power output over time by varying the ratio of power source on-time versus cycle-time (one on-time plus off-time cycle). The implementation we shall discuss is the switching of a binary output on and off with varying duration to generate an average power output.



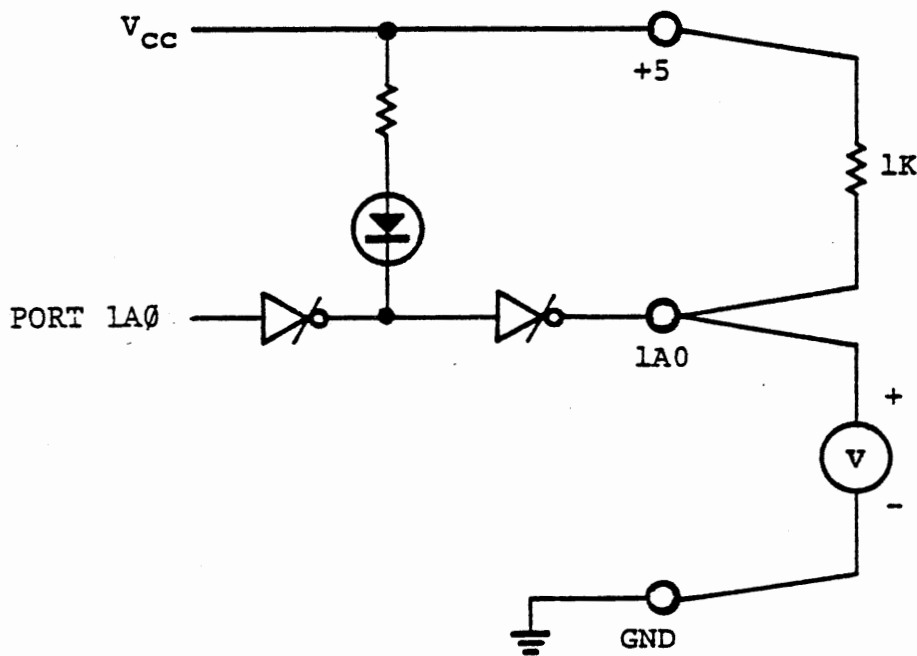
The figure shows a signal whose average power is decreasing over time; there is a constant cycle-time, but a varying on-time whose duration is decreasing. Although the constant cycle-time is not necessary, it simplifies the computation. Some loads that might be driven may limit the minimum or maximum on-time, in which case the cycle-time must be varied in order to vary the on-time/cycle-time ratio (duty cycle).

Pulse width modulation has three great advantages for analog output: it requires only a single bit from the processor to switch from the on state to the off state; it allows the load power to be controlled by a switching device such as a relay or SCR rather than a power amplifier; and it minimizes power dissipation (heat loss) in the control device.

4.2.1 PWM Output Program

EXERCISE

We will develop a program to generate a pulse width modulated output signal, with keyboard entries to set the cycle time and the duty cycle. (Duty cycle = on-time/cycle-time). We will drive one of the port 1A outputs with this signal and observe the result with a voltmeter. It will also be visible to some degree in the brightness of the LED. Figure 4-2 shows the connections required.



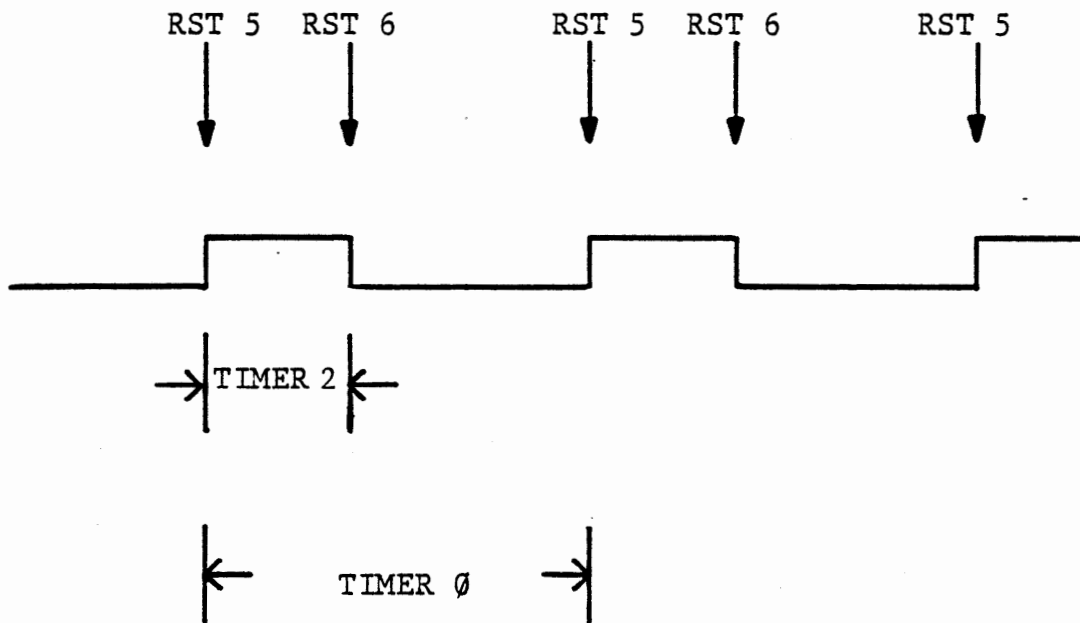
OUTPUT CONNECTIONS FOR PWM

Figure 4-2

4.2.1.1 PWM Program Operation

We will use two timers and two interrupt service routines to control the PWM output signal. Timer 0, operating in mode 2, will control the uniform cycle time. It will repetitively count down from its initial value, generating an RST 5 interrupt when it reaches zero. The RST 5 service routine will turn the output signal on, load Timer 2 with the on-time and enable the Timer 2 interrupt. It will also reenale the Timer 0 interrupt to clear the latch.

When Timer 2 counts down to zero, an RST 6 interrupt will occur. The RST 6 interrupt service routine will turn the output signal off, and disable Timer 2 interrupt.



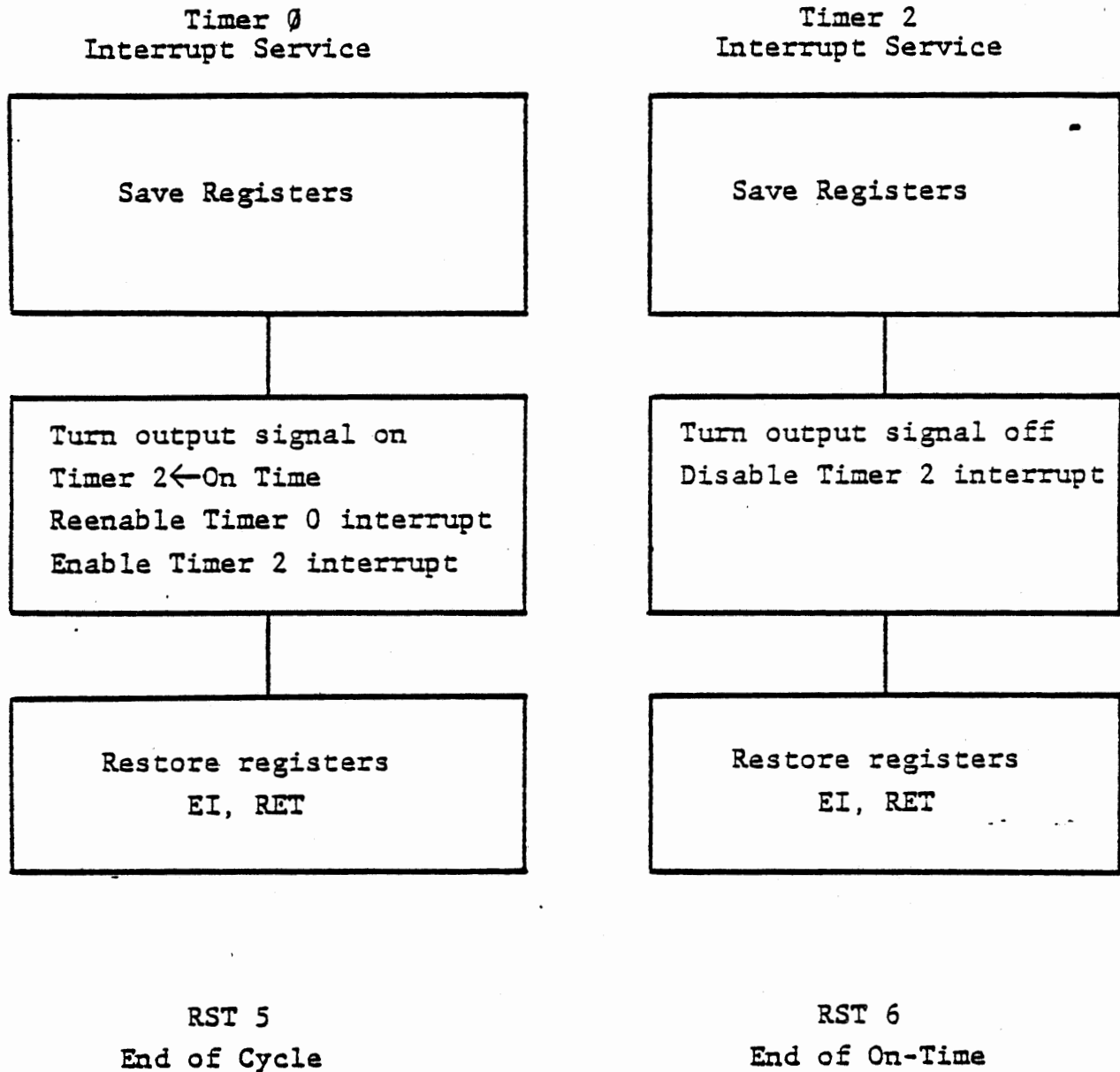
Since timer 0 generates RST 5, then only timer 2 will be enabled for RST 6; the program will distinguish the two interrupts by their RST instructions. For the moment, let us ignore the main program and examine the interrupt service routines in Figure 4-3.

4.2.1.2 PWM Interrupt Service

At the end of a cycle, timer 0 generates an RST 5 interrupt. After saving the registers, we turn the output signal on. The 8255 does not have individual bit control for its port A, but we can achieve it by:

```
IN    PORT1A
ORI   01
OUT   PORT1A
```

Even when a port is programmed for output, its content can be read by the program. This allows restoration of all bits in port 1A except the one we want to change. Since we are not using the other bits, this procedure is not really needed, but in some other program it is a useful technique.

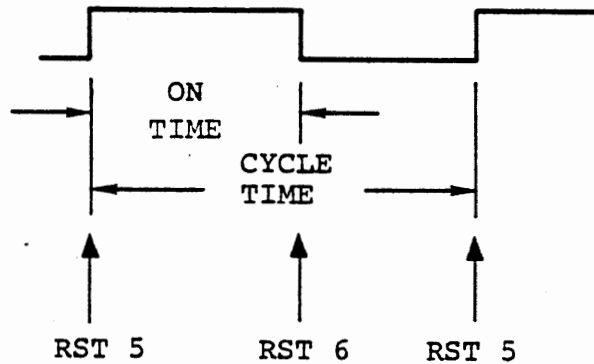


PULSE WIDTH MODULATION INTERRUPT SERVICE

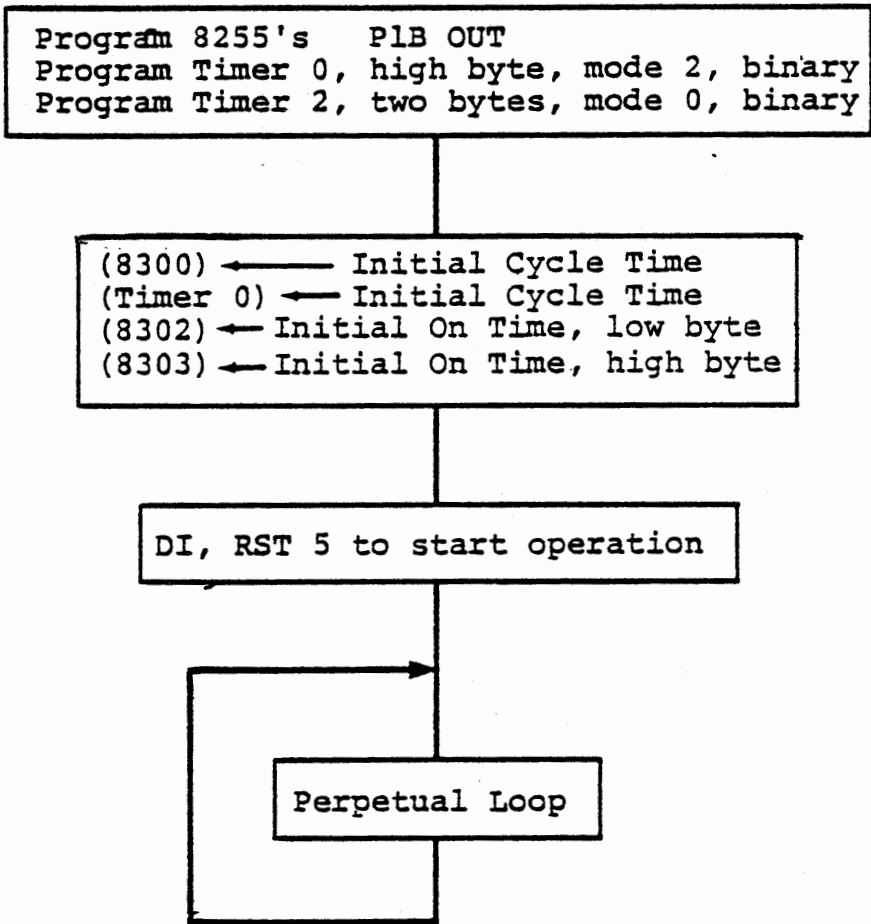
Figure 4-3

Now the RST 5 routine loads timer 2, which is operating in mode 0; enables both interrupts, and exits. The output signal has been turned on and will stay on until a timer 2 interrupt occurs.

The RST 6 interrupt service routine is invoked by timer 2. It turns the output signal off; disables its own interrupt, and exits. Now the output will stay off until the timer 0 interrupt service turns it on again.



The time loaded to timer 0 sets the cycle time; the time loaded to timer 2 sets the on-time.

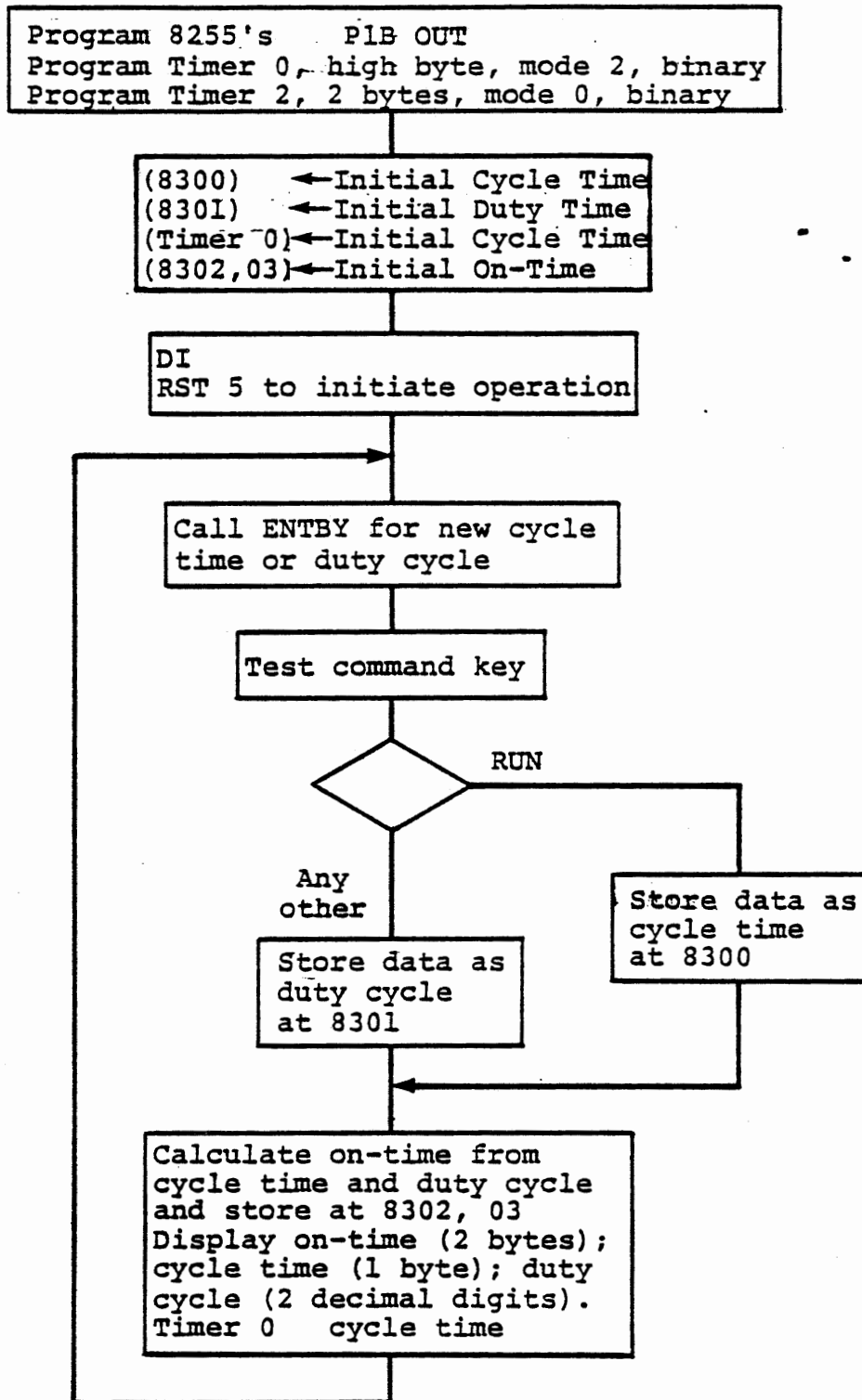


PWM TEST PROGRAM

Figure 4-4

4.2.1.3 PWM Test Program

Write the interrupt service routines according to Figure 4-3, and a trivial main program to program the ports and timers as shown in Figure 4-4. This will prove the hardware interface and the interrupt service routines. The average power can be changed by entering different initial values for cycle time and on-time.



PWM MAIN PROGRAM

Figure 4-5

4.2.1.4 PWM Main Program

After testing the hardware and interrupt service routines, we will develop a more interesting program to allow keyboard control of cycle time and duty cycle. (Duty Cycle = on-time/cycle time.) The program of Figure 4-5 calls the monitor subroutine ENTBY to obtain a one byte value and a command key:

```
CD  CALL ENTBY
    36
    03
```

ENTBY accepts numeric keys, always returning the last two keys entered as a byte in register L, and returns when a command key has been pressed and released, with the command key value in registers A and B. All registers except E are used. (See Course 525, Section 6.10.3) During operation most of the time will be spent scanning the keyboard, waiting for keyboard entries. Although the monitor program as a whole cannot be interrupted (since it disables the interrupt), any of its subroutines can be, so the PWM interrupt service routines will control the output.

This page intentionally left blank

When keyboard data are returned by ENTBY, we will test the command key (register A): if it is RUN (=14), the data byte (register L) will be stored as a new cycle time (and subsequently written to timer 0); otherwise, the data byte will be taken as a decimal duty cycle.

We use a mixed number system in this program. Cycle time and on-time are kept as binary numbers, but duty cycle is accepted, stored and displayed as a decimal fraction. A subroutine (at 8290) multiplies the binary cycle time by the decimal fraction duty cycle to obtain a two byte binary on-time. This unorthodox procedure is used because it is easy to choose a cycle time from the table given in Appendix A, Figure A-7, but binary fractions expressed in hexadecimal are awkward to handle mentally.

To multiply a binary value by a decimal fraction, we must convert the decimal to a binary fraction. The subject of decimal to binary fraction conversion is discussed in Appendix D, where the BVXDF subroutine is developed. Curious readers are directed to appendix D, sections D.1 to D.2. We will assume the existence of a subroutine, BVXDF, which multiplies a binary number by a decimal fraction, and use this subroutine in our program.

4.2.1.5 Use of the PWM program

Write your complete program in accordance with Figures 4-3 and 4-5. You can test and debug the data entry, display and multiplication sections without enabling the interrupts by omitting the DI and RST 5 instructions. The solution given in Figure 4-6 is subdivided as follows:

4-6a	8200-8223	Initialization
4-6b	8228-823F	RST 5 entry and RST 6 processing
4-6c	8240-8257	RST 5 Processing
4-6d	8260-8288	Main Program Loop
4-6e	8290-82AA	Subroutine BVXDF

To run the given program, depress RST, then RUN. The voltmeter should show about 2-1/2 volts, due to an initial cycle-time of 50H(10ms) and an initial duty cycle of 50%. Keying in a decimal duty cycle (1-99), followed by the NEXT key (any command key except RUN) will change the duty cycle accordingly and display it in two right hand digits. The four left digits will contain the binary count on-time (in hexadecimal clock pulses, see Figure 4-7). Keying in a hexadecimal value followed by the RUN key, will change the cycle timer to the new value and display it in the remaining two display digits.

When the entire program is operating the voltmeter (connected as shown in Figure 4-2) will display a value proportional to the duty cycle. This depends on the mechanical inertia of the voltmeter to integrate the signal; a digital voltmeter will be confused by the PWM signal

unless it has an averaging or true RMS capability.

The average output voltage is proportional to the duty cycle, and independent of the cycle time. With a maximum cycle time, you may be able to see some slight motion of the voltmeter needle; or if you make the cycle time = 16.625 milliseconds, you may see it with the stroboscopic effect of fluorescent lighting.

PULSE WIDTH MODULATION OUTPUT 4 - 19

A D D R		CODE					
CODING SHEET	8	200	3E	MVI	A, 80	Program 8255's	
		1	80			PIB Out	
		2	D3	OUT	CNT1		
		3	07				
		4	3E	MVI	A, 92		
		5	92				
		6	D3	OUT	CNT2		
		7	0F				
		8	3E	MVI	A, 24	Program timer 0	
		9	24			high byte	
MICROCOMPUTER TRAINING SYSTEM	A	D3	OUT	TIMCT	mode 2, binary		
	B	17					
	C	3E	MVI	A, B0	Program timer 2		
	D	B0			2 bytes		
	E	D3	OUT	TIMCT	mode 0, binary		
	F	17					
	8	210	21	LXI	H, 5050		
		1	50			(L) ← Cycle time 10 ms	
		2	50			(H) ← Duty cycle 50%	
	MICROCOMPUTER TRAINING SYSTEM		3	22	SHLD	8300	
		4	00			(8300) ← Cycle time	
		5	83			(8301) ← Duty Cycle	
		6	7D	MOV	A, L	(A) ← Cycle Time	
		7	D3	OUT	TIM0	Timer 0 ← Cycle Time	
		8	14				
		9	21	LXI	H, 2800	(HL) ← Initial OnTime	
		A	00			= 5 milliseconds.	
		B	28				
INTEGRATED COMPUTER SYSTEMS			C	22	SHLD	8302	
		D	02			(8302) ← On time (low)	
		E	83			(8303) ← On time (high)	
		F	F3	DI		Start operation	
	8	220	EF	RST	5	Jump to program	
		1	C3	JMP	8260	loop, beyond	
		2	80			interrupt service	
		3	82				
		4	00				
		5	00				
	6	00					
	7	00					
	8						

Figure 4-6a

INTERRUPT SERVICE FOR PWM

	A	D	D	R	CODE																	
CODING SHEET	8	0																				
		1																				
		2																				
		3																				
		4																				
		5																				
		6																				
		7																				
MICROCOMPUTER TRAINING SYSTEM	822	8	F5		PUSH	PSW														RST5-Timer 0		
		9	E5		PUSH	H																
		A	C3		JMP	8240																
		B	40																			
		C	82																			
		D	60		NOP																	
		E	00																			
		F	00																			
	INTEGRATED COMPUTER SYSTEMS	823	0	F5		PUSH	PSW															RST6-Timer 2
			1	E5		PUSH	H															
		2	DB		IN	PORT1A															Turn output	
		3	04																			signal at IAO
		4	E6		ANI	FE																off. Do not
		5	FE																			change other bits
		6	D3		OUT	PORT1A																
		7	04																			
		8	3E		MVI	A, 04																Disable timer 2
		9	04																			interrupt
	A	D3		OUT	CNT2																	
	B	0F																				
	C	E1		POP	H																	
	D	F1		POP	PSW																	
	E	FB		EI																		
	F	C9		RET																		
INTEGRATED COMPUTER SYSTEMS	8	0																				
		1																				
		2																				
		3																				
		4																				
		5																				
		6																				
		7																				
	8																					

Figure 4-6b

INTERRUPT SERVICE FOR TIMER 0 contd 4 - 21

A D D R		CODE									
CODING SHEET	8	24	0	DB		IN		PORT1A		Read Port1A	
			1	04						existing data	
			2	FG		ORI		01		Set bit 1 high	
			3	01							
			4	D3		OUT		PORT1A		Output data	
			5	04							
			6	2A		LHLD		8302		(HL) ← On Time	
			7	02							
			8	83							
			9	7D		MOV		A, L		Timer 2 ← On Time	
MICROCOMPUTER TRAINING SYSTEM	A			D3		OUT		TIM 2			
	B			16							
	C			7C		MOV		A, H			
	D			D3		OUT		TIM 2			
	E			16							
	F			3E		MVI		A, 01		Renenable timer 0	
	8	25	0	01						interrupt	
			1	D3		OUT		CNT 2			
			2	0F							
			3	3E		MVI		A, 05		To enable timer 2	
		4	05						interrupt		
		5	C3		JMP		823A		Jump to enable		
		6	3A						and return		
		7	82								
INTEGRATED COMPUTER SYSTEMS											

Figure 4-6c

PWM OUTPUT - PROGRAM LOOP 4 - 22

A	D	D	R	CODE					
8	2	6	0	CD	CALL	ENTBY			
	1			36					
	2			03					
	3			FE	CPI	RUN			Test for command
	4			14					key = RUN
	5			7D	MOV	A, L			(A) ← keyed byte
	6			21	LXI	H, 8300			Address for
	7			00					cycle time
	8			83					
	9			CA	JZ	826D			Jump if RUN key
A				6D					
B				82					
C				23	INX	H			Address duty cycle
826	D			77	MOV	M, A			Store cycle time or
	E			2A	LHLD	8300			duty cycle
	F			00					(L) ← cycle time
827	0			83					(H) ← duty cycle
	1			E5	PUSH	H			Save for display
	2			CD	CALL	BVXDF			Multiply
	3			90					Binary Value
	4			82					X Decimal Fraction
	5			22	SHLD	8302			Store on time
	6			02					at 8302, 03
	7			83					
	8			CD	CALL	DWORD			Display on-time
	9			D1					at left (2 bytes)
A				02					
B				E1	POP	H			(L) ← cycle time
C				7C	MOV	A, H			(H) ← duty cycle
D				CD	CALL	DBYTE			Display duty
	E			95					cycle at right
	F			02					
828	0			7D	MOV	A, L			(A) ← cycle time
	1			D3	OUT	TIMO			TIMO ← cycle time
	2			14					
	3			CD	CALL	DBY2			Display cycle time
	4			98					between on-time
	5			02					and duty cycle
	6			C3	JMP	8260			Loop
	7			60					
	8			82					

Figure 4-6d

SUBR - MULTIPLY BINARY VALUE X DECIMAL FRACTION

A	D	D	R	CODE			
8	29	0	4C	MOV	C, H		(C) ← decimal fraction
		1	55	MOV	D, L		(D) ← binary value
		2	21	LXI	H, 0000		Clear product in (HL) and
		3	00				low byte of
		4	00				binary value in (E)
		5	5D	MOV	E, L		
829		6	7A	MOV	A, D		LOOP - Test binary
		7	B3	ORA	E		value and return
		8	C8	RZ			when zero
		9	7A	MOV	A, D		Shift two byte
A			1F	RAR			binary value
B			57	MOV	D, A		right one bit
C			7B	MOV	A, E		for half of
D			1F	RAR			preceding value
E			5F	MOV	E, A		
F			79	MOV	A, C		(A) ← decimal fraction
82A		0	87	ADD	A		Double decimal
		1	27	DAA			fraction
		2	4F	MOV	C, A		(C) ← remainder
		3	D2	JNC	8296		Do not add binary
		4	96				value to product
		5	82				if no carry
		6	19	DAD	D		Add binary value to
		7	C2	JNZ	8296		product and loop
		8	96				unless decimal
		9	82				remainder zero
A			C9	RET			Return if decimal
B							remainder zero
C							
D				ENTER WITH			
E				(L) = BINARY VALUE			
F				(H) = DECIMAL FRACTION			
8		0					
		1		RETURN WITH			
		2		(HL) = TWO BYTE PRODUCT			
		3					
		4		USES ALL REGISTERS			
		5		EXCEPT B			
		6					
		7					
		8					

Figure 4-6e

Binary Count	Decimal Count	Time (msecs)
0100	256	0.125
0200	512	0.250
0400	1024	0.500
0800	2048	1.000
1000	4096	2
1800	6144	3
2000	8192	4
2800	10240	5
3000	12288	6
3800	14336	7
4000	16384	8
4800	18432	9
5000	20480	10
A000	40960	20
F000	61440	30
0000	65536	32

Conversion of Binary Count to Time

Figure 4-7

4.2.7 Variable Cycle Time

OPTIONAL EXERCISE

Postulate an open loop control system for the heating of a chemical reactor, in which the heater duty cycle is set according to the volume of material being cooked. As in the preceding exercise, the required duty cycle is an input to the computer. Because a gas fired heater is used, and ignition is not instantaneous, the minimum useful on-time is 20 seconds; fuel efficiency is enhanced by longer on-times. When the batch is small, however, a heater off-time exceeding 180 seconds may result in excessive cooling between heat cycles, and shorter off-time is preferable. The chemical engineer has provided this table of on time vs. duty cycle; interpolation is to be used between these values.

Duty Cycle	On Time	Off Time	Cycle Time
.10	20	180	200
.20	20	80	100
.30	30	70	100
.40	40	60	100
.50	60	60	120
.60	90	60	150
.70	140	60	200
.80	160	40	200
.90	180	20	200
1.00	200	0	200

Design a program that will vary the on-time and off-time according to this table, with any reasonable interpolation scheme. To make a convenient display during program debugging, divide the times by 10.

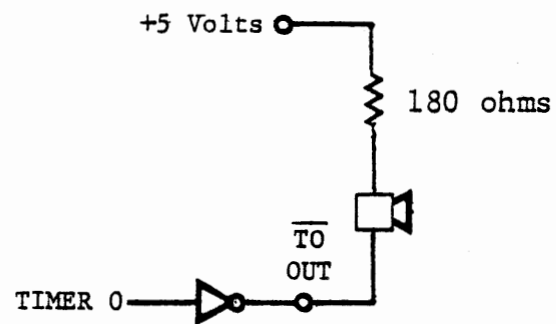
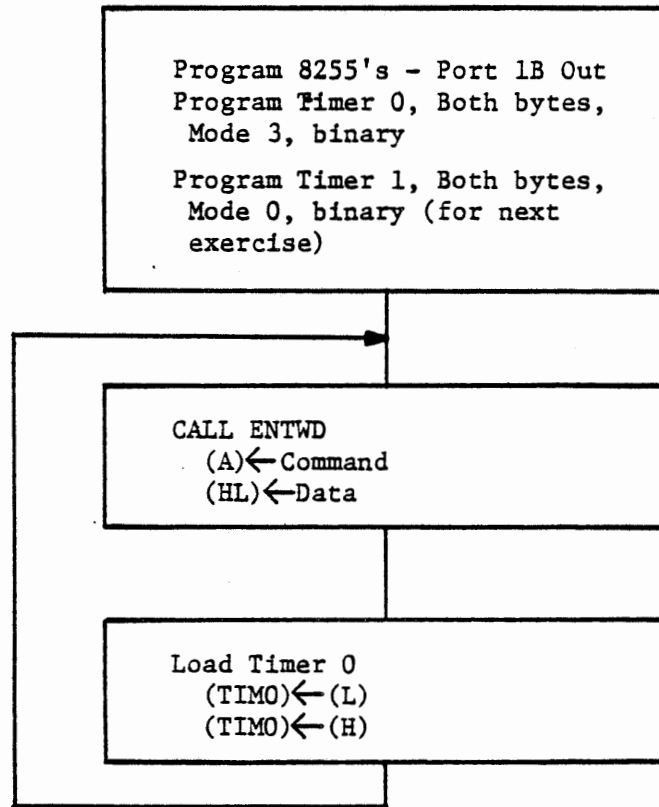
This page intentionally left blank

4.3 Frequency Control

Frequency is another analog that requires only a single output bit with time as the continuous variable. Varying the frequency of an output signal can control an induction or synchronous AC motor, can control the delivery or flow of a fluid, or can test frequency dependent hardware.

Generation of a fixed frequency signal is automatically accomplished by the 8253 in mode 3, the square wave generator, as we demonstrated in Section 3.9.1. The frequency can be varied by loading different time intervals to the timer. Some applications of variable frequency control cannot tolerate the high harmonic content of a square wave, and some form of multi-bit output is required to create a more nearly sinusoidal signal. We will experiment with this in Section 4.7.

The following exercise creates audio tones with the square wave generator thus demonstrating a technique for frequency generation. Although tones are sometimes used as alarms, the most common use of tone generation is for frequency modulated data communication and recording, which is used in the cassette interface included on the experiment board. In the next exercise we will modify the previous program to vary the frequency and to demonstrate frequency modulation. The final exercise in this section will create music from the square wave generator. This is more of a toy than part of a useful system, but it uses important programming techniques such as bit manipulation and table look-up, as well as frequency modulation.



AUDIO OUTPUT PROGRAM AND CIRCUIT

FIGURE 4-8

4.3.1 Audio Tone Generator

Write a program to load timer 0 with data entered through the keyboard. Connect the loudspeaker as shown in Figure 4-8 to output the tone generated by timer 0. The timer is to operate in square wave mode.

Monitor subroutine ENTWD (0346) will accept two data bytes and a command, returning the data in register pair HL and the command in register A. Load the data (less significant byte first) into timer 0. If you enter data from the list in Figure 4-9 the tone will be a defined musical note. If you have perfect pitch or a standard for comparison such as a tuning fork or a high quality audio oscillator you may detect that the tone is imperfect. This stems from error in the computer's crystal clock and from rounding error in the calculation of $\text{period} = 1/\text{frequency}$. (The latter is only significant at the very high frequencies). Try keying in the data for various notes and listen to the tone.

Musical Note	Frequency (Hertz)	Timer Period (Hex Count)
C (below Middle C)	130.81	3D28
C#	138.59	39B9
D	146.83	367C
D#	155.56	336D
E	164.81	308A
F	174.61	2DD1
F#	185.00	2B3E
G	196.00	28D1
G#	207.65	2687
A	220.00	245D
A#	233.08	2253
B	246.94	2065
C (Middle C)	261.63	1E94
C#	277.18	1CDD
D	293.66	1B3E
D#	311.13	19B7
E	329.63	1845
F	349.23	16E8
F#	369.99	159F
G	392.00	1468
G#	415.30	1343
A A440	440.00	122F
A#	466.16	1129
B	493.88	1033
C (above Middle C)	523.25	0F4A
C#	554.37	0E6E
D	587.33	0D9F
D#	622.25	0CDB
E	659.26	0C23
F	698.46	0B74
F#	739.99	0AD0
G	783.99	0A34
G#	830.61	09A2
A	880.00	0917
A#	932.33	0895
B	987.77	0819
C C''	1046.50	07A5
C#	1108.73	0737
D	1174.66	06CF
D#	1244.51	066E
E	1318.51	0611
F	1396.91	05BA
F#	1479.98	0568
G	1567.98	051A
G#	1661.22	04D1
A	1760.00	048C
A#	1864.66	044A
B	1975.53	040D

List of Concert Pitch

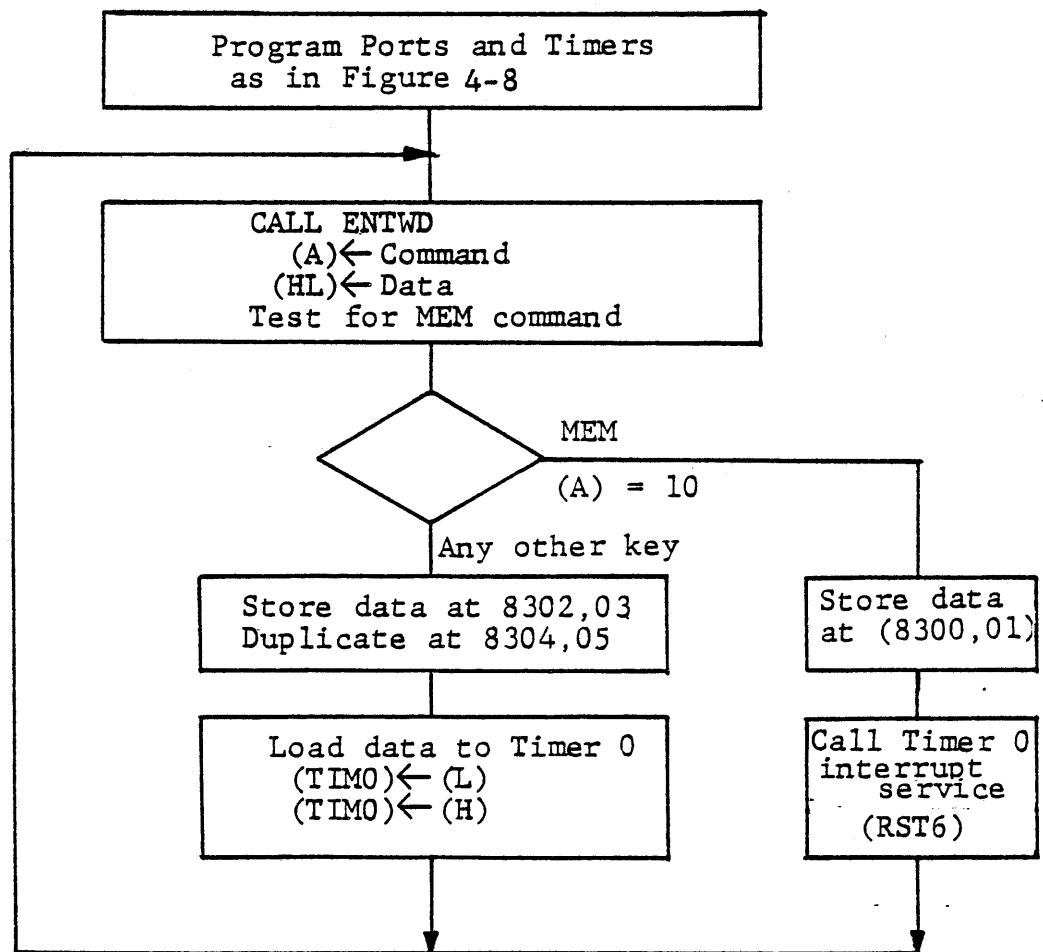
Musical Tones

FIGURE 4-9

4.3.2 Frequency Modulation Program

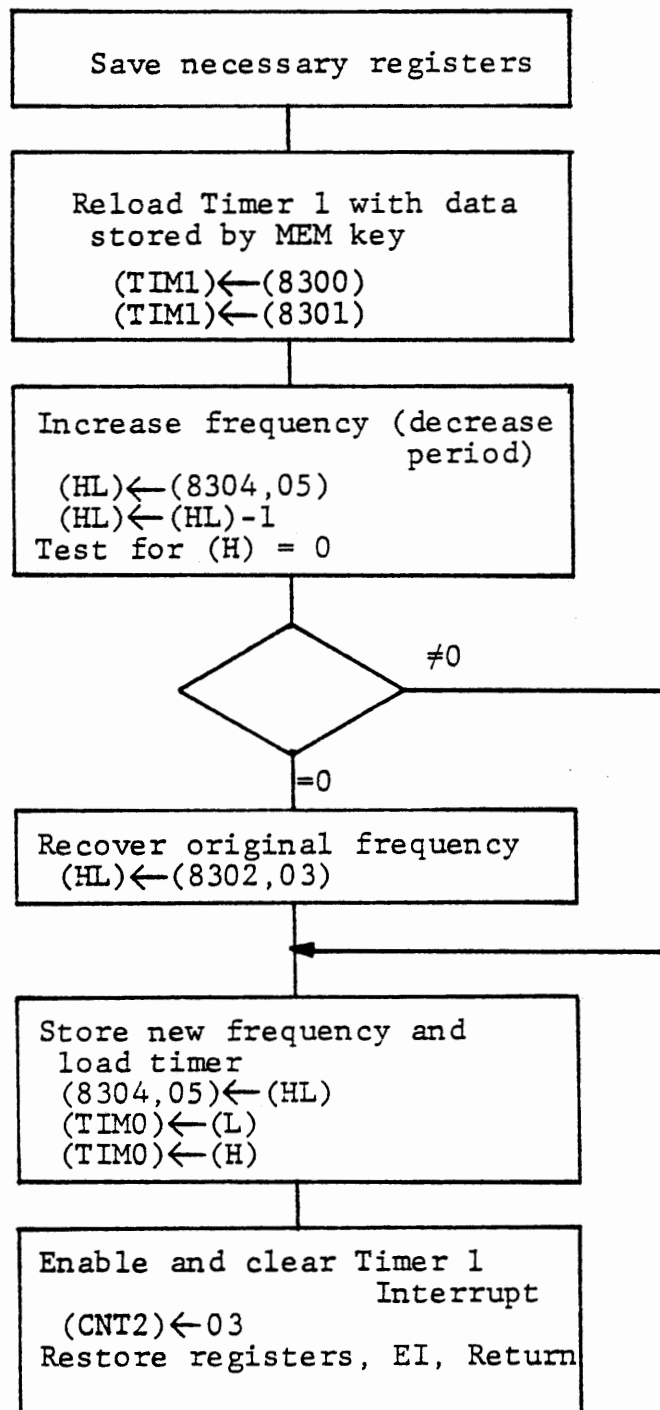
Modify the tone generator program to generate a tone whose frequency increases with time. Data loaded with the MEM command will control the rate of change by setting an interrupt period in timer 1. Data loaded with any other command key will provide the initial frequency. At each interrupt the period loaded to timer 0 will be reduced by one count to increase its frequency. After the frequency of 8000 HZ (period = 0100 hex) has been generated the original frequency will be reloaded. Figure 4-10 shows the main program and Figure 4-11 shows timer 1 interrupt service.

To use the program enter a period from the tone table of Figure 4-9 using any command except MEM. The tone will be steady at first, since timer 1 is not running. Now enter an interval to timer 1, using the MEM key. The tone will slide from the initial frequency up to 8000 HZ, and then repeat.







TONE GENERATOR - MAIN PROGRAM

FIGURE 4-10



TONE GENERATOR INTERRUPT SERVICE

FIGURE 4-11

				
C (Below Middle C)	00	80	40	20
C# (D♭)	01	81	41	21
D	02	82	42	22
D# (E♭)	02	83	43	23
E	04	84	44	24
F	05	85	45	25
F# (G♭)	06	86	46	26
G	07	87	47	27
G# (A♭)	08	88	48	28
A	09	89	49	29
A# (B♭)	0A	8A	4A	2A
B	0B	8B	4B	2B
C (Middle C)	0C	8C	4C	2C
C# (D♭)	0D	8D	4D	2D
D	0E	8E	4E	2E
D# (E♭)	0F	8F	4F	2F
E	10	90	50	30
F	11	91	51	31
F# (G♭)	12	92	52	32
G	13	93	53	33
G# (A♭)	14	94	54	34
A	15	95	55	35
A# (B♭)	16	96	56	36
B	17	97	57	37
C (Above Middle C)	18	98	58	38
C# (D♭)	19	99	59	39
D	1A	9A	5A	3A
D# (E♭)	1B	9B	5B	3B
E	1C	9C	5C	3C
F	1D	9D	5D	3D
F# (G♭)	1E	9E	5E	3E
Rest	1F	9F	5F	3F

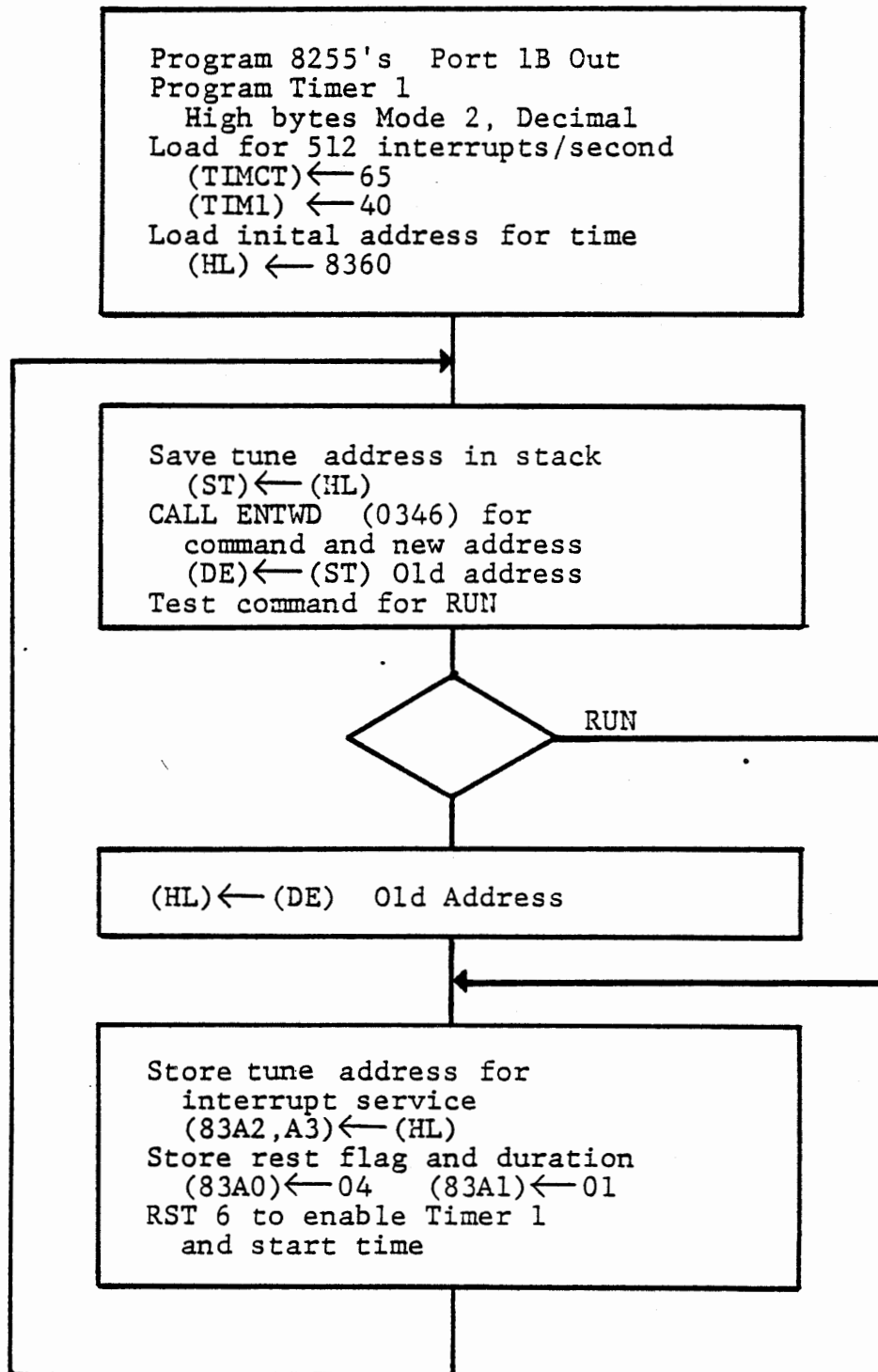
CODES FOR MUSICAL NOTES

FIGURE 4-12

4.3.3 Recorded Music Player

This program module reads music - in the form of notes in a list. Each note in the tune includes three bits to indicate duration of the note (eighth, quarter, half and whole notes) and five bits to select one of 30 tones starting at C below middle C and covering two and one-half octaves. (See Figure 4-12). For each note it performs a table lookup on the five low order bits to find a time interval corresponding to the tone frequency, and outputs that time to timer 0 operating as a square wave generator. The inverted output of timer 0 drives a loudspeaker, as shown in Figure 4-8.

Timer 0 runs in mode 3; its interrupt is disabled, and its only function is to generate the square wave. Timer 1 runs in mode 2 to give a repetitive timing interrupt which is used to count down a software counter, loaded from the high three bits of the note, to set the note duration.



TUNE - MAIN PROGRAM

FIGURE 4-13

The main program (figure 4-13) initializes the ports and timer 1, and loads a fixed address (8360) where a tune is stored. It calls ENTWD to permit entry of different addresses where other tunes can be loaded. The RUN key repeats the last tune addressed, while any other key requires a new tune address. This address and a flag and counter are stored for use by the interrupt service routine.

Timer 1 is programmed during initialization to mode 2, decimal, and loaded with 80 in its high byte to interrupt 256 times per second. This value makes a whole note of one second. Faster tempo can be obtained by loading smaller values to timer 1. (You may want to elaborate the program to accept a value from the keyboard.)

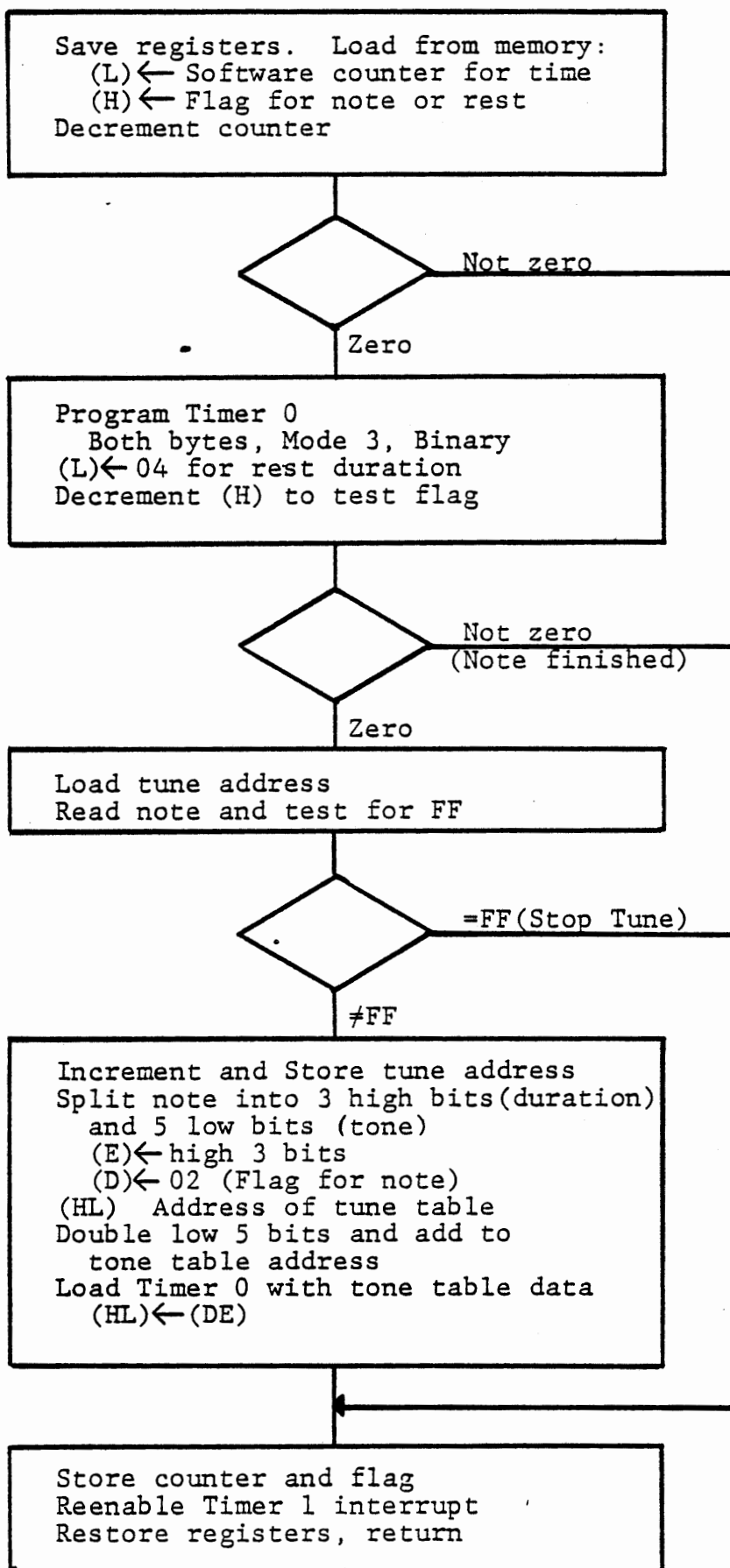


FIGURE 4-14

Interrupt service for timer 1 (figure 4-14) decrements a software counter (83A0) and exits if not zero.

At zero it reprograms timer 0 (to mode 3, binary) which stops counting until the timer is reloaded. A flag at 83A1 is decremented, and if it goes from 02 to 01 a rest of 1/64 note (4 counts) is generated to separate one musical note from the next. The next time that the software counter (83A0) reaches zero the flag will count down from 01 to 00, and a new note is played. The tune address is loaded from memory (83A2, A3) and the next note is read. If this note is FF the tune is finished and an exit is made without loading timer 0. For any other note the tune address is incremented and stored, and the note is split into three high bits for duration and five low bits for tone. The high bits are entered to the software counter, and the flag is set to 02 to indicate a note is being played. Now the five low bits are tested for code 1F which indicates a rest in the music. For any other code, 00 to 1E, the tone table is addressed to find the time interval for the corresponding note. The tone table data are copied to timer 0, which now generates a square wave of the required frequency.

This page intentionally left blank

Figure 4-15 is a complete program with a tone look-up table (Figure 4-16) and a few tunes (Figure 4-17). The tone table covers four octaves, more than can be addressed by the five bits allotted for selecting a note. You can change the table address to obtain a different set of notes. This also permits transposing a tune from one major key to another, simply by entering a different address for the table. You may want to provide keyboard entry for this, also.

TUNE - MAIN PROGRAM

	A	D	D	R	CODE								
CODING SHEET	8	2	0	0	3E	MVI	A,	80				Program Ports	
					80								
					D3	OUT	CNT1						
					07								
					3E	MVI	A,	92					
					92								
					D3	OUT	CNT2						
					0F								
					3E	MVI	A,	65					Program Timer 1
					65								High byte
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT	TIMCT					Mode 2	
	B				17							Decimal	
	C				3E	MVI	A,	80				Load for	
	D				80							interrupts/second	
	E				D3	OUT	TIMI					(change for	
	F				15							different tempo)	
	8	2	1	0	11	LXI	D,	8360					Initial address
					60								for tune
					83								
					D5	PUSH	D						Save address
INTEGRATED COMPUTER SYSTEMS					CD	CALL	ENTWD					(A) ← command	
					46							(HL) ← new tune	
					03							address	
					D1	POP	D					(DE) ← old address	
					FE	CPI	RUN					Test command	
					14								
	A				CA	JZ	821E						IF RUN use
	B				1E								new address
	C				82								
					EB	XCHG							(HL) ← old address
	8	2	1	E	22	SHLD	83A2					Store tune address	
				F	A2							for interrupt svc	
INTEGRATED COMPUTER SYSTEMS	8	2	2	0	83								

FIGURE 4-15a

TUNE - MAIN CONTINUED

	A	D	D	R	CODE								
CODING SHEET	8												
	8	2	2	1	EB	XCHG						(DE) ← tune address	
				2	21	LXI	H	0	1	0	4		
				3	04							(L) ← 1/64 note	
				4	01							(H) ← flag for rest	
				5	22	SHLD		8	3	A	0	Store counter	
				6	A0							and flag for	
				7	83							interrupt service	
				8	F7	RST6						Initiate interrupts	
				9	C3	JMP		8	2	1	3		
MICROCOMPUTER TRAINING SYSTEM	A				13								
	B				82								
	C												
	D												
	E												
	F												
	8			0									
				1									
				2									
				3									
				4									
				5									
				6									
				7									
				8									

INTEGRATED COMPUTER SYSTEMS

FIGURE 4-15b

TUNE - INTERRUPT SERVICE

A D D R		CODE						
CODING SHEET	8 2 3 0	F5	PUSH	PSW				
		1	ES	PUSH	H			
		2	D5	PUSH	D			
		3	C5	PUSH	B			
		4	2A	LHLD	83A0			
		5	A0					(L) ← counter
		6	83					(H) ← flag
		7	2D	DCR	L			Decrement counter
		8	C2	JNZ	EXIT			Exit if not zero
		9	70					
MICROCOMPUTER TRAINING SYSTEM	A	82						
	B	3E	MVI	A, 36				Program Timer 0
	C	36						Both bytes
	D	D3	OUT	TIMCT				Mode 3
	E	17						Binary
	F	2E	MVI	L, 04				(L) ← 1/64 note
	8 2 4 0	04						to counter for rest
		1	25	DCR	H			If flag was 02
		2	C2	JNZ	EXIT			go to exit for rest
		3	70					
MICROCOMPUTER SYSTEMS		4	82					
		5	2A	LHLD	83A2			(HL) ← tune address
		6	A2					
		7	83					
		8	7E	MOV	A, M			(A) ← note
		9	FE	CPI	FF			Test for end of
		A	FF					tune
		B	CA	JZ	EXIT			Jump to rest if end
		C	70					
		D	82					
INTEGRATED COMPUTER SYSTEMS		E	E6	ANI	E0			(A) ← 3 high bits
		F	E0					
	8	0						
		1						
		2						
		3						
		4						
		5						
		6						
		7						
	8							

FIGURE 4-15c

TUNE - INTERRUPT SERVICE continued 4 - 46

		A	D	D	R	CODE													
CODING SHEET	8	2	5	0		5F		MOV	E, A									(E) ← high 3 bits	
				1		AE		XRA	M									(A) ← low 5 bits	
				2		23		INX	H									Next tune address	
				3		22		SHLD		83A2									Store address
				4		A2													
				5		83													
				6		16		MVI	D, 02										Set flag for note
				7		02													
				8		FE		CPI	IF										Test for rest
				9		1F													in tune
MICROCOMPUTER TRAINING SYSTEM	A					CA		JZ		826A								Do not load timer	
	B					6A												if rest code	
	C					82													
	D					21		LXI	H, 8300									Address tone table	
	E					00													
	F					83													
	8	2	6	0		87		ADD	A									Double low 5 bits	
				1		85		ADD	L										Add to table
				2		6F		MOV	L, A										address
				3		7E		MOV	A, M										} (Timer 0) ← low byte of interval
			4		D3		OUT	TIM0											
			5		14														
			6		23		INX	H										} (Timer 0) ← high byte of interval	
			7		7E		MOV	A, M											
			8		D3		OUT	TIM0											
			9		14														
INTEGRATED COMPUTER SYSTEMS	A					EB		XCHG										(H) ← Flag	
	B					00		NOP										(L) ← Count	
	C					00													
	D					00													
	E					00													
	F					00													
	8			0															
				1															
				2															
				3															
			4																
			5																
			6																
			7																
			8																

FIGURE 4-15d

TUNE - INTERRUPT SERVICE EXIT

A D D R		CODE						
8	27	0	22		SHLD	83A0		Store counter (L)
	1		A0					and flag (H)
	2		83					
	3		3E		MVI	A, 03		
	4		03					
	5		D3		OUT	CNT2		
	6		0F					
	7		C1		POP	B		
	8		D1		POP	D		
	9		E1		POP	H		
	A		F1		POP	PSW		
	B		FB		EI			
	C		C9		RET			
	D							
	E							
	F							
8	0							
	1							
	2							
	3							
	4							
	5							
	6							
	7							
	8							
	9							
	A							
	B							
	C							
	D							
	E							
	F							
8	0							
	1							
	2							
	3							
	4							
	5							
	6							
	7							
	8							

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

FIGURE 4-15e

CLOCKS

FREQUENCY

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE						
8	3	0	0	28	C	3	=	130.81	C	BELOW MIDDLE C
			1	3D						
			2	B9	C#	3	=	138.59		
			3	39						
			4	7C	D	3	=	146.83		
			5	36						
			6	6D	D#	3	=	155.56		
			7	33						
			8	8A	E	3	=	164.81		
			9	30						
	A			D1	F	3	=	174.61		
	B			2D						
	C			3E	F#	3	=	185.00		
	D			2B						
	E			D1	G	3	=	196.00		
	F			28						
8	3	1	0	87	G#	3	=	207.65		
			1	26						
			2	5D	A	3	=	220.00		
			3	24						
			4	53	A#	3	=	233.08		
			5	22						
			6	65	B	3	=	246.94		
			7	20						
			8	94	C	4	=	261.63	MIDDLE C	
			9	1E						
	A			DD	C#	4	=	277.18		
	B			1C						
	C			3E	D	4	=	293.66		
	D			1B						
	E			B7	D#	4	=	311.13		
	F			19						
8			0		(CONTINUED)					
			1							
			2		TONE TABLE FOR					
			3		CHROMATIC SCALE					
			4							
			5							
			6							
			7							
			8							

FIGURE 4-16a

CLOCKS

FREQUENCY

A	D	D	R	CODE		FREQUENCY									
CODING SHEET	8	3	2	0	45	E	4	=	3	2	9	.	6	3	
				1	18										
				2	E8	F	4	=	3	4	9	.	2	3	
				3	16										
				4	9F	F#4	=	3	6	9	.	9	9		
				5	15										
				6	68	G	4	=	3	9	2	.	0	0	
				7	14										
				8	43	G#4	=	4	1	5	.	3	0		
				9	13										
MICROCOMPUTER TRAINING SYSTEM	A				2F	A	4	=	4	4	0	.	0	0	
	B				12										
	C				29	A#4	=	4	6	6	.	1	6		
	D				11										
	E				33	B	4	=	4	9	3	.	8	8	
	F				10										
	8	3	3	0	4A	C	5	=	5	2	3	-	2	5	C ABOVE MIDDLE C
				1	0F										
				2	6E	C#5	=	5	5	4	.	3	7		
				3	0E										
			4	9F	D	5	=	5	8	7	.	3	3		
			5	0D											
			6	DB	D#5	=	6	2	2	.	2	5			
			7	0C											
			8	23	E	5	=	6	5	9	.	2	6		
			9	0C											
INTEGRATED COMPUTER SYSTEMS	A				74	F	5	=	6	9	8	.	4	6	
	B				0B										
	C				DO	F#5	=	7	3	9	.	9	9		
	D				0A										
	E				34	G	5	=	7	8	3	.	9	9	
	F				0A										
	8			0		(CONTINUED)									
				1											
				2											
				3											
			4												
			5												
			6												
			7												
			8												

FIGURE 4-15b

CLOCKS

FREQUENCY

		A	D	D	R	CODE						
CODING SHEET	8	3	4	0		A 2	G # 5 = 830.61					
				1		0 9						
				2		1 7	A 5 = 880.00					
				3		0 9						
				4		9 5	A # 5 = 932.33					
				5		0 8						
				6		1 9	B 5 = 987.77					
				7		0 8						
				8		A 5	C 6 = 1046.50					
				9		0 7						
MICROCOMPUTER TRAINING SYSTEM		A				3 7	C # 6 = 1108.73					
		B				0 7						
		C				C F	D 6 = 1174.66					
		D				0 6						
		E				6 E	D # 6 = 1244.51					
		F				0 6						
		8	3	5	0		1 1	E 6 = 1318.51				
				1			0 6					
				2			B A	F 6 = 1396.91				
				3			0 5					
INTEGRATED COMPUTER SYSTEMS				4		6 8	F # 6 = 1479.98					
				5		0 5						
				6		1 A	G 6 = 1567.98					
				7		0 5						
				8		D 1	G # 6 = 1661.22					
				9		0 4						
			A				8 C	A 7 = 1760.00				
			B				0 4					
			C				4 A	A # 7 = 1864.66				
			D				0 4					
		E				0 D	B 7 = 1975.53					
		F				0 4						
	8			0								
				1								
				2								
				3								
				4								
				5								
				6								
				7								
				8								

FIGURE 4-16c

HOME ON THE RANGE

A D D R		CODE											Tone	Duration			
CODING SHEET	8	360	4C												C	1/4	0
		1	4C												C	1/4	GIVE
		2	51												F	1/4	ME
		3	53												G	1/4	A
		4	9✓												A	1/2	HOME
		5	31												F	1/8	WHERE
		6	30												E	1/8	THE
		7	6E												D	3/8	BUFF
		8	36												Bb	1/8	A
		9	56												Bb	1/4	LO
MICROCOMPUTER TRAINING SYSTEM	A	96												Bb	1/2	ROAM	
	B	35												A	1/8	WHERE	
	C	36												Bb	1/8	THE	
	D	98												C	1/2	DEER	
	E	31												F	1/8	AND	
	F	31												F	1/8	THE	
	8	0	51											F	1/4	ANT	
		1	50												E	1/4	E
		2	51												F	1/4	LOPE
		3	13												G	whole	PLAY
INTEGRATED COMPUTER SYSTEMS	4	4C												C	1/4	WHERE	
	5	4C												C	1/4	SEL	
	6	51												F	1/4	DDM	
	7	53												G	1/4	IS	
	8	95												A	1/2	HEARD	
	9	31												F	1/8	A	
	A	30												E	1/8	DIS	
	B	6E												D	3/8	COUR	
	C	36												Bb	1/8	AG	
	D	56												Bb	1/4	ING	
E	96												Bb	1/2	WORD		
F	36												Bb	1/8	AND		
8	0	36											Bb	1/8	THE		
	1	75												A	3/8	SKIES	
	2	33												G	1/8	ARE	
	3	51												F	1/4	NOT	
	4	70												E	3/8	CLOUD	
	5	31												F	1/8	Y	
	6	53												G	1/4	ALL	
	7	11												F	whole	DAY	
	8																

FIGURE 4-17a

4.3.4 Music Recording Program

OPTIONAL EXERCISE

Develop a program that will play notes entered from the keyboard, recording the tune as it is played. Define the hexadecimal keys to represent sixteen notes in the key of C.

C	D	E	F
F	G	A	B
B	C	D	E
E	F	G	A

Define the command keys for playing and editing the music:

ADDR Load a starting address optionally followed by a four digit address. Otherwise use the last address entered.

BRK Set a musical key (to be followed by a note, or by sharp or flat and a note).

MEM Sharp (to precede a note)

REG Flat (to precede a note)

CIR Delete (from the recorded tune) the last note played, replacing it with the stop code (FF).

RUN Play the tune from the beginning to the end, and wait for a new note to be added to the tune.

STEP Play the next note recorded (if any), and wait for a new note to be added.

NEXT Enter a rest in the tune.

To record a tune the musician will enter:

ADDR XXXX to locate the tune

REG D (for example) to set the key of
 D flat.

CIR to delete any note already recorded at
 that location, and prepare to replace it.

Now enter the successive notes. The program must transpose the note of the selected musical key into a note in the chromatic scale, play that note while the hexadecimal key is held down, and measure the duration of the note. When the key is released, generate and store the code for the tune and duration.

To end the recording enter any note or a rest (NEXT) and press CIR to replace it with the stop code.

To play the tune back, press ADDR, RUN.

To edit the tune, press ADDR, STEP, STEP, STEP etc. to play one note at a time. Replace any desired note by CIR and the desired note.

Development of this program is left as an exercise for the student. The relationship between the musical keys (C, D flat, etc.) and the meaning of the hex keys is shown in figure 4-18. The hex code given for each note is the whole note code of Figure 4-12.

Hex Key	C	C#	D	D#	E	F
0	E 04	D# 03	E 04	D# 03	E 04	E 04
1	F 05	F 05	F# 06	F 05	F# 06	F 05
2	G 07	F# 06	G 07	G 07	G# 08	G 07
3	A 09	G# 08	A 09	G# 08	A 09	A 09
4	B 0B	A# 0A	B 0B	A# 0A	B 0B	A# 0A
5	C 0C	C 0C	C 0C	C 0C	C# 0D	C 0C
6	D 0E	C# 0D	D 0E	D 0E	D# 0F	D 0E
7	E 10	D# 0F	E 10	D# 0F	E 10	E 10
8	F 11	F 11	F# 12	F 11	F# 12	F 11
9	G 13	F# 12	G 13	G 13	G# 14	G 13
A	A 15	G# 14	A 15	G# 14	A 15	A 15
B	B 17	A# 16	B 17	A# 16	B 17	A# 16
C	C 18	C 18	C 18	C 18	C# 19	C 18
D	D 1A	C# 19	D 1A	D 1A	D# 1B	D 1A
E	E 1C	D# 1B	E 1C	D# 1B	E 1C	C 1C
F	F 1D	F 1D	F# 1E	F 1D	F# 1E	F 1D

Hex Key	F#	G	G#	A	A#	B
0	F 05	E 04	E 04	E 04	D# 03	F 04
1	F# 06	F# 06	F 05	F# 06	F 05	F# 06
2	G# 08	G 07	G 07	G# 08	G 07	G# 08
3	A# 0A	A 09	G# 08	A 09	A 09	A# 0A
4	B 0B	B 0B	A# 0A	B 0B	A# 0A	B 0B
5	C# 0D	C 0C	C 0C	C# 0D	C 0C	C# 0D
6	D# 0F	D 0E	C# 0D	D 0E	D 0E	D# 0F
7	F 11	E 10	D# 0F	E 10	D# 0F	E 10
8	F# 12	F# 12	F 11	F# 12	F 11	F# 12
9	G# 14	G 13	G 13	G# 14	G 13	G# 14
A	A# 16	A 15	G# 14	A 15	A 15	A# 16
B	B 17	B 17	A# 16	B 17	A# 16	B 17
C	C# 19	C 18	C 18	C# 19	C 18	C# 19
D	D# 1B	D 1A	C# 19	D 1A	D 1A	D# 1B
E	F 1D	E 1C	D# 1B	E 1C	D# 1B	E 1C
F	F# 1E	F# 1E	F 1D	F# 1E	F 1D	F 1D

FIGURE 4-18

4.4 Multi-bit Output

A multi-bit output can represent a continuous variable to any desired precision. The output is usually in the form of a binary number with each bit having a weighted value (e.g. 1,2,4,8,16---); this must be converted to a voltage or current by external hardware. Section 4.5 deals with this procedure. Another possibility, occasionally used as a display device, is to illuminate an LED as a pointer. A prototype automobile speedometer has been shown with an LED at each mile per hour position; here all of the lower values are illuminated, up to and including the actual speed. We will modify the tune program of Section 4.3.3 to display the tone in this fashion, using the LED's of port 1A.

In the program of figure 4-15 the interrupt service obtains a note to be played and makes a conditional jump if the note is a rest. This is a good place to insert a patch to display the tone. At 825A replace JZ 826A by JMP 8280, where we will place the patch.

We will display the notes as follows:

NOTES	CODES	DISPLAY
C,C#,D,D#	00,01,02,03	00000001
E,F,F#,G	04,05,06,07	00000011
G#,A,A#,B	08,09,0A,0B	00000111

Octave of middle C

C,C#,D,D#	0C,0D,0E,0F	00001111
E,F,F#,G	10,11,12,13	00011111
G#,A,A#,B	14,15,16,17	00111111

Octave above middle C

C,C#,D,D#	18,19,1A,1B	01111111
E,F,F#	1C,1D,1E	11111111
Rest		00000000

The patch must save the note and the flag, and it must include the JZ 826A instruction that was replaced. We can obtain the desired display by masking unwanted bits and shifting, so that the codes 00,01,02,03 are transformed to 00, and 04,05,06,07 are transformed to 01, etc. Then increment the result so that the values range from 01, to 08. Now this procedure will shift from one to eight 1's into register H:

```

                LXI        H,00FF
LOOP           DAD        H
                DCR        A
                JNZ        LOOP

```

Register H can be displayed in port 1A. Figure 4-19 shows the patch. For many tunes it may be more interesting to mask for the three low bits and omit the shifting, so each note within a small range will be displayed differently.

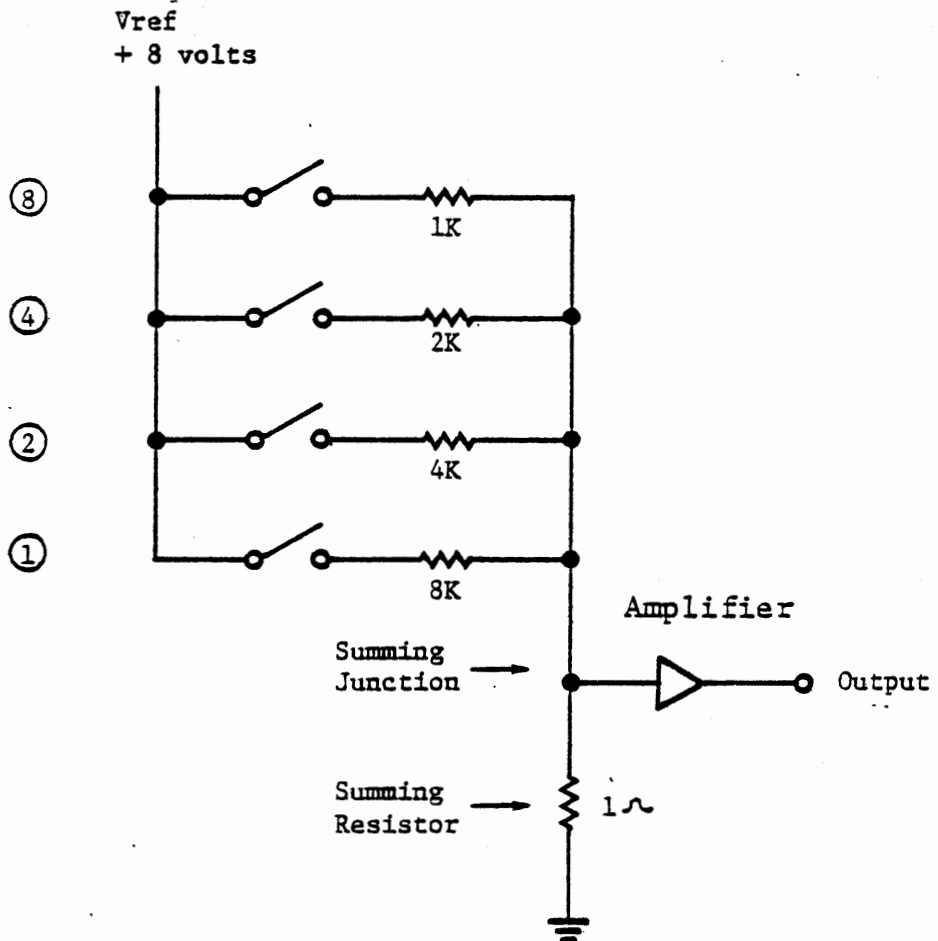
4.5 Analog Voltage Generation

Probably the most common analog signal is a variable voltage. The remainder of this chapter and most of Chapter 5 are concerned with variable voltage signals and their interface with the computer.

Clearly a variable voltage can be generated by a pulse width modulated signal integrated by resistors and capacitors; we will use such a generator in Chapter 5. Other schemes involve multi-bit output.

4.5.1 Binary Summing Circuit

Consider the network shown in Figure 4-20.



Note: Resistance and reference voltage shown are selected for convenience of discussion; they are not typical values.

BINARY SUMMING CIRCUIT

FIGURE 4-20

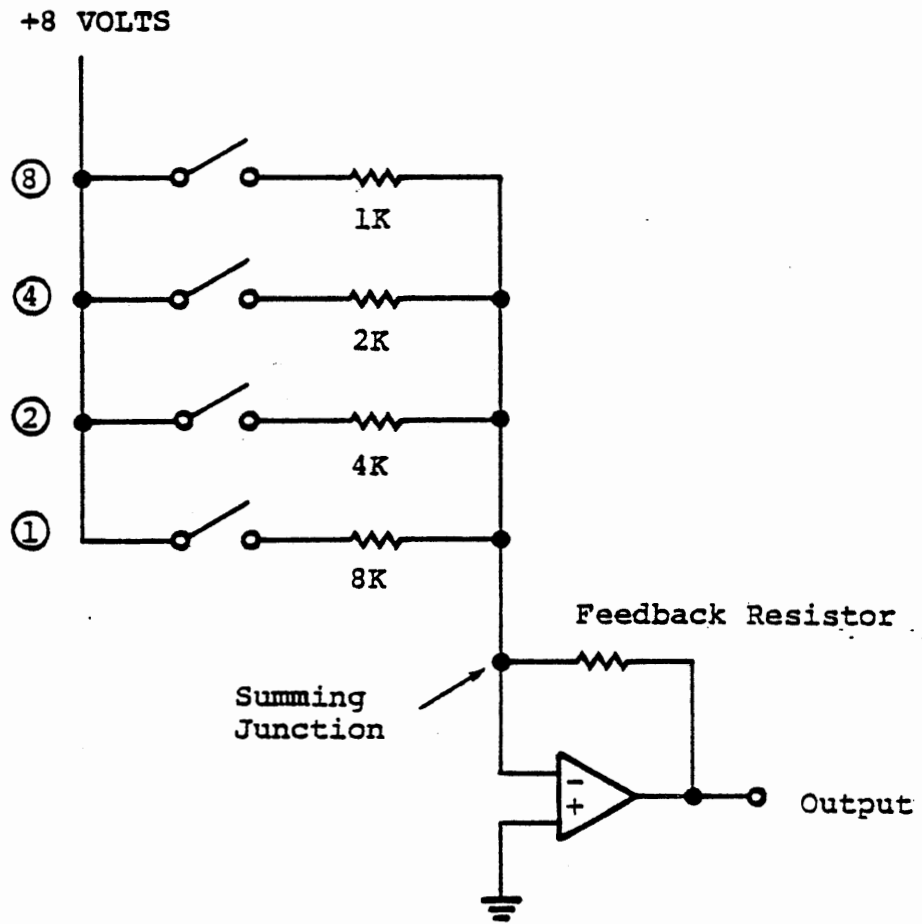
Each of the switches, labelled 1,2,4, and 8, represents a contact closure or a transistor switch operated by one bit of the computer output. If switch 8 is closed, a current of 8 milliamperes flows through the 1K resistor and generates an 8 millivolt signal across the 1 ohm summing resistor. If switch 4 is also closed, a current of 4 milliamperes flows through the 2K resistor; the two currents are summed to generate 12 millivolts at the summing junction. Thus any combination of the four switches generates an analog voltage proportional to the binary output as shown in Figure 4-21. The output amplifier generates a more useful signal level.

Bit Value	Resistor	Current	
1	8K	1 ma	
2	4K	2 ma	
4	2K	4 ma	
8	1K	8 ma	

Binary Value	Parallel Resistance	Current	Voltage
0000		0	0.0
0001	8000	1	0.001
0010	4000	2	0.002
0011	2667	3	0.003
0100	2000	4	0.004
0101	1600	5	0.005
0110	1333	6	0.006
0111	1143	7	0.007
1000	1000	8	0.008
1001	889	9	0.009
1010	800	10	0.010
1011	727	11	0.011
1100	667	12	0.012
1101	585	13	0.013
1110	571	14	0.014
1111	533	15	0.015

NUMERICAL VALUES FOR CIRCUIT OF 4-12

FIGURE 4-21

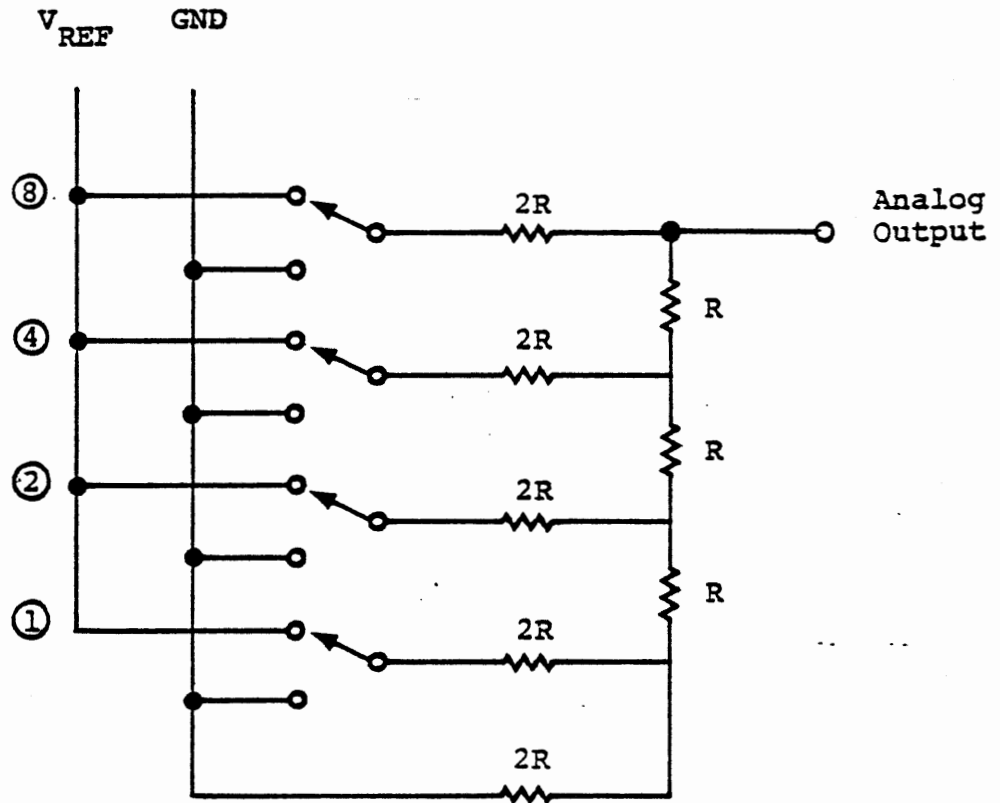


BINARY SUMMING CIRCUIT WITH OP AMP

FIGURE 4-22

The accuracy of this device is limited by the influence of the voltage at the summing resistor; as more bits are used in the conversion, this becomes more significant, but is largely overcome by the use of an operational amplifier, as shown in Figure 4-22. With this connection, the op-amp output is inverted from the input signal; the current from the resistor network actually flows to the op-amp output through the feedback resistor, and the voltage at the summing junction is held very close to ground, so that the crosstalk between bits (i.e. the influence of one bit on the signal generated by another) is very small. For a detailed discussion of operational amplifiers, the student is referred to Wait, Huelsman and Vorn, "Introduction to Operational Amplifier Theory and Applications", McGraw-Hill, 1975. This particular subject is discussed in Chapter 1, page 11.

Even with the op-amp summing circuit, the binary weighted network suffers from the wide range of precision resistor values required. For a modest number of bits (up to 8 or even 12) these can be obtained with discrete resistors, but a range from 1k to 128K (for an eight bit converter) is impractical for monolithic construction. The R-2R ladder network overcomes this problem.



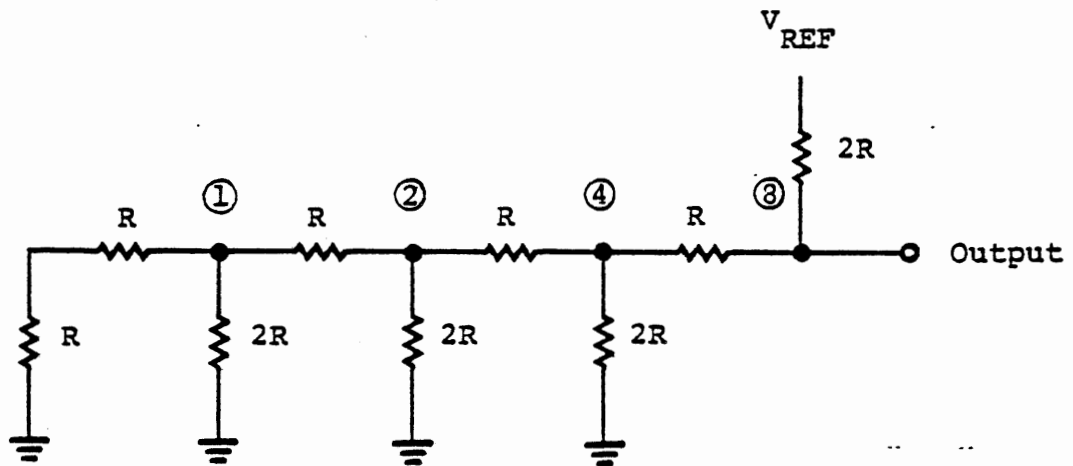
R-2R LADDER NETWORK

FIGURE 4-23

4.5.2 R-2R Ladder Network

Figure 4-23 shows an R-2R Ladder Network for digital to analog conversion. In this circuit bipolar (i.e. double throw) switches are required, so that for each bit a resistor is connected either to the reference voltage or to ground. If all bits are 0, then all connections go to ground and the output is 0 volts; if any bit is 1, its resistor is connected to the reference voltage and injects current into the network to develop a positive output voltage.

The R-2R network has two major advantages: only two resistor values are used, and they differ only by a factor of 2, so it can readily be constructed as a monolithic circuit; and it has a constant impedance independent of the binary input. The figure below shows an equivalent circuit for the case where the most significant bit is a 1 and the remaining bits are 0.



Looking to the left along the circuit from any node, with all of the less significant bits 0, one always sees an impedance of $2R$ to ground as depicted in Figures 4-24a through 4-24c. Now, if the $2R$ resistor for the most significant bit is connected to the positive reference voltage, a simple voltage divider is formed, giving an output signal equal to half the reference voltage as shown in Figure 4-24d.

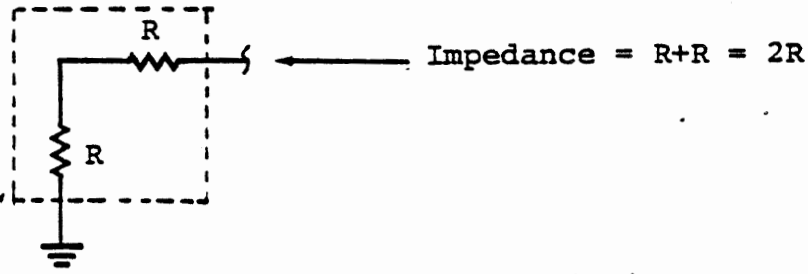


FIGURE 4-24a

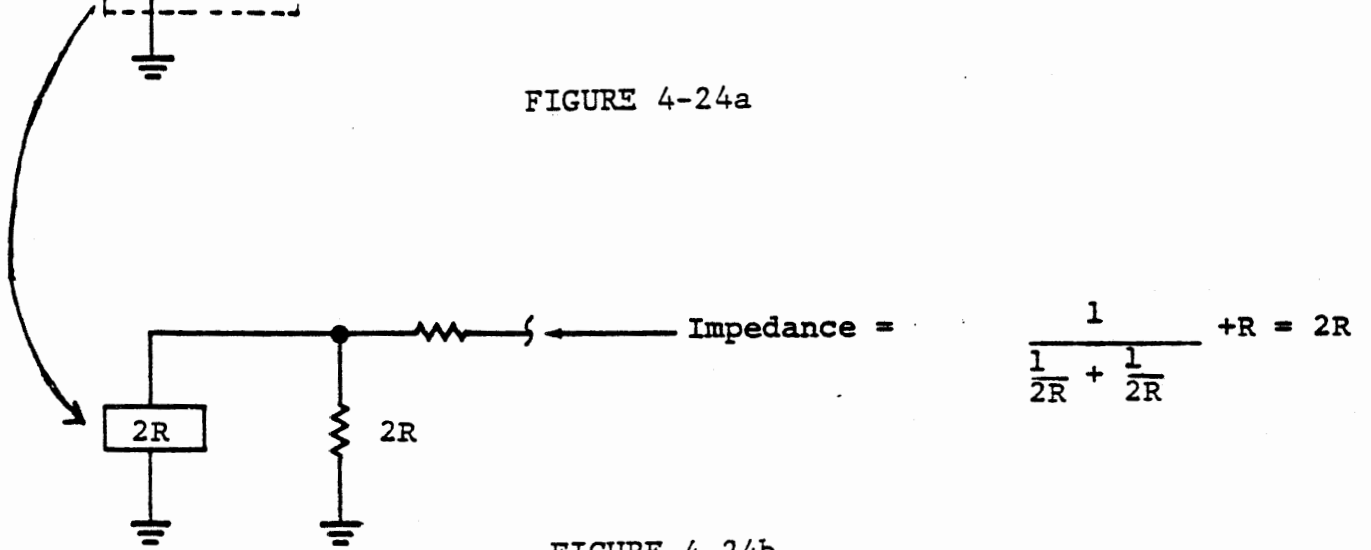


FIGURE 4-24b

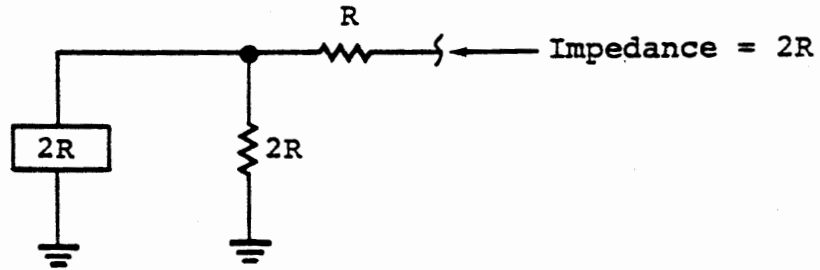


FIGURE 4-24c

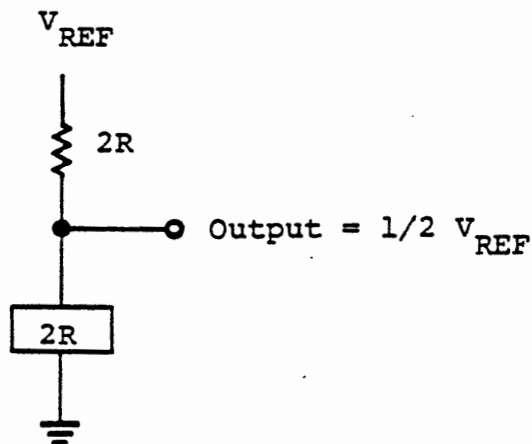
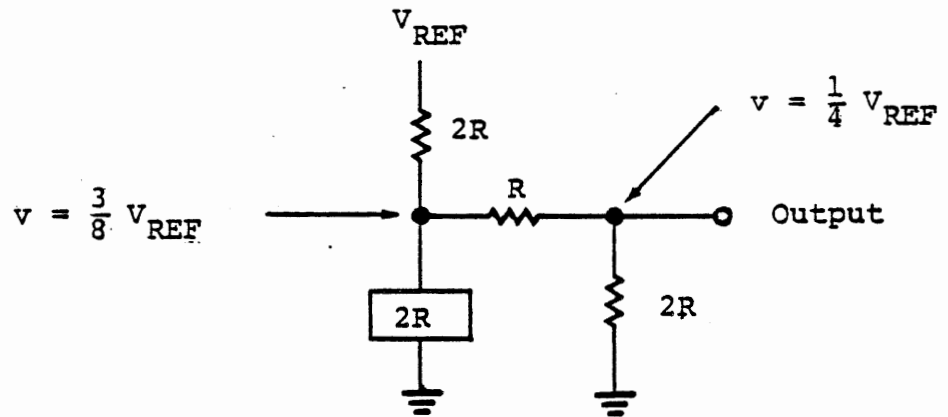
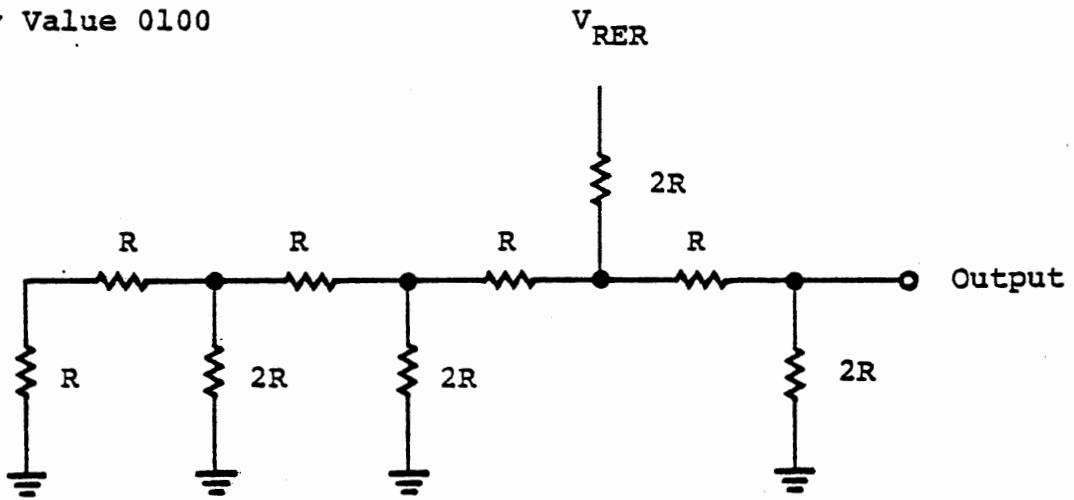
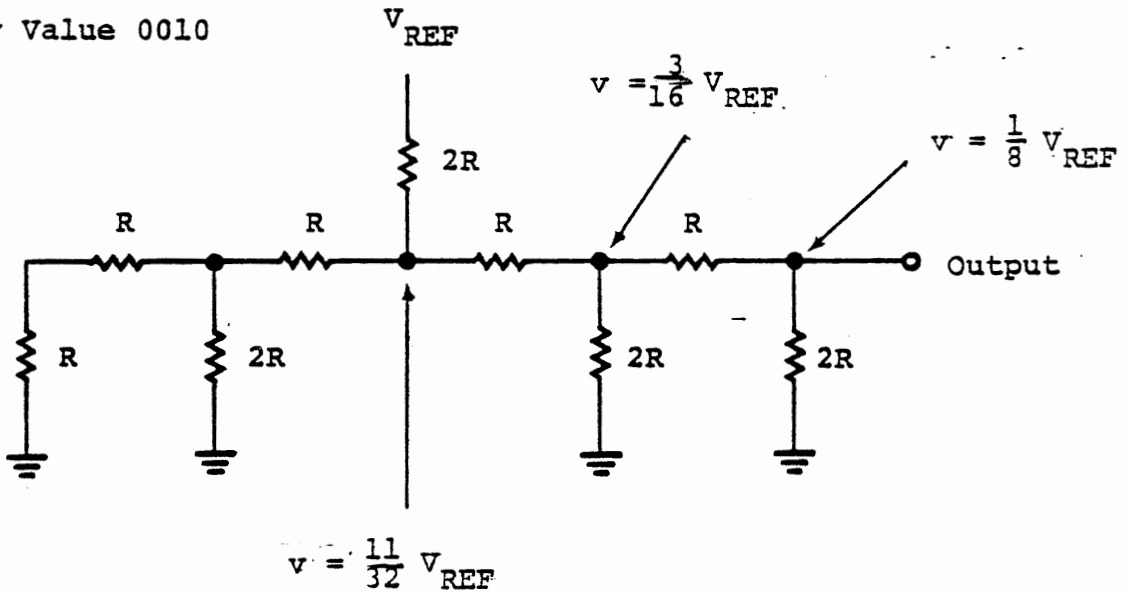


FIGURE 4-24d

Binary Value 0100



Binary Value 0010



EQUIVALENT CIRCUITS FOR SINGLE BIT = 1

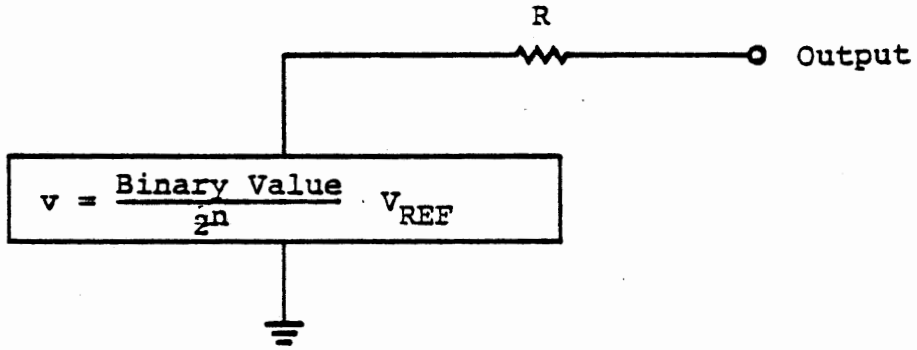
FIGURE 4-25

Figure 4-25 shows the voltages for two other cases where a single bit is 1. When multiple bits are 1's, their voltages add, to give an output proportional to the binary input and the reference voltage.

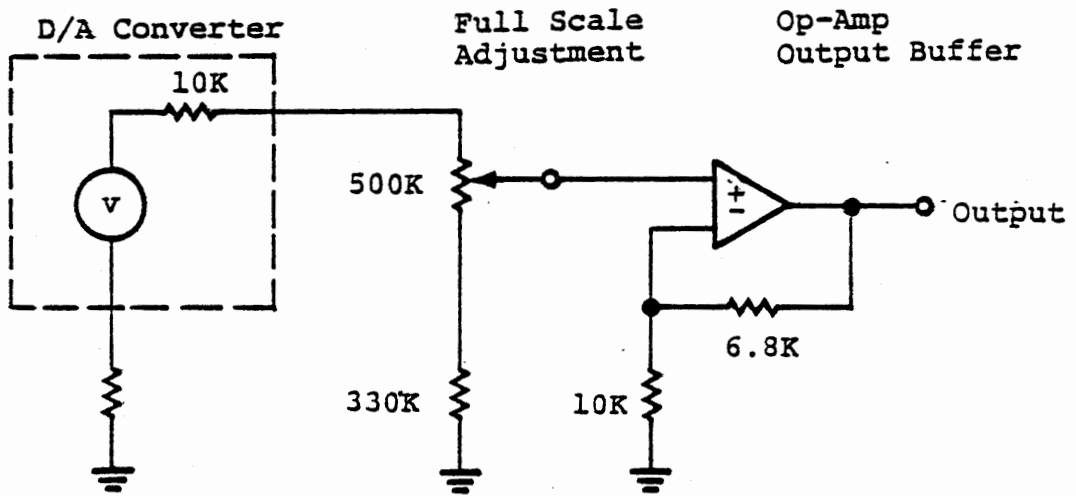
$$V_{\text{out}} = \frac{\text{Binary Value}}{2^n} V_{\text{ref}}$$

The output circuit sees a source impedance equal to R , from the parallel combination of the $2R$ resistor for the high bit and the ladder network to the left. Thus, the Thevenin equivalent circuit for the ladder network is the voltage given above with a series resistance equal to R , as shown in Figure 4-26.

The R - $2R$ Ladder has been discussed in detail because it is used in the Ferranti D/A Converter included on the experiment board. The operation of this device is discussed in the next section. It drives an operational amplifier, also shown in Figure 4-26, to isolate the load from the converter. A pot provides for adjustment of the full scale output, to compensate for error in the reference voltage and the value of R . In a system designed for a single purpose much less adjustment range would generally be provided, but in the experiment board it was considered desirable to have a wide range to allow for various experiments.



R - 2R LADDER EQUIVALENT CIRCUIT



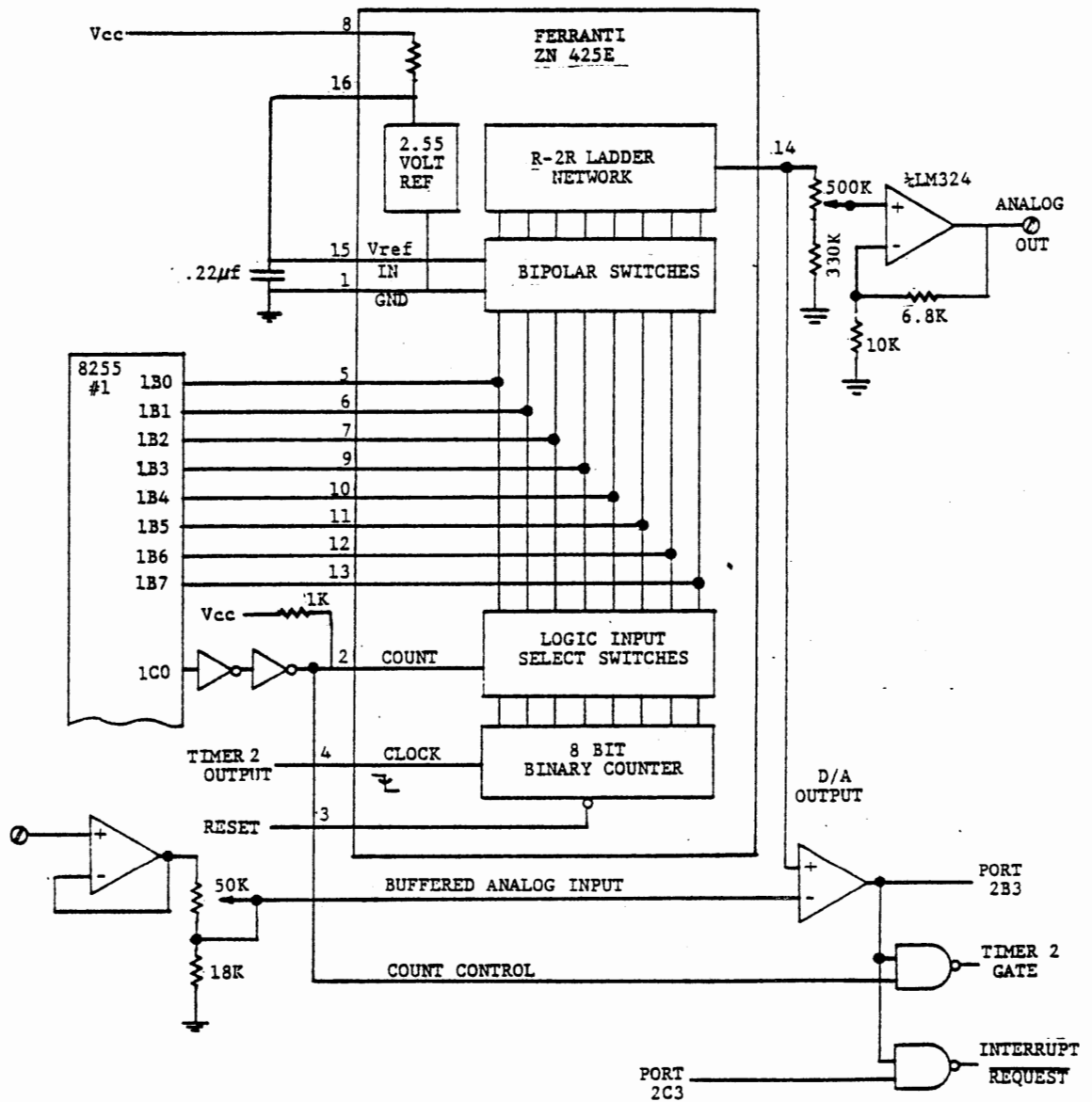
D/A CONVERTER OUTPUT CIRCUIT

FIGURE 4-26

4.6 Ferranti D/A Converter

In this section we will describe the Ferranti ZN 425E Digital to Analog/ Analog to Digital Converter, and experiment with its D/A mode. Chapter 5 deals with analog to digital input using the 425.

The device is a monolithic 8 bit D/A converter using an R-2R ladder network. It contains an internal voltage reference source and a binary counter used for A/D conversion. The manufacturer's data sheet is appended at the end of this chapter. Figure 4-27 is a block diagram of the device, with its inputs and outputs.



FERRANTI D/A CONVERTER

FIGURE 4-27

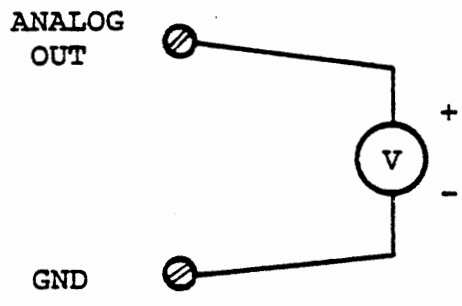
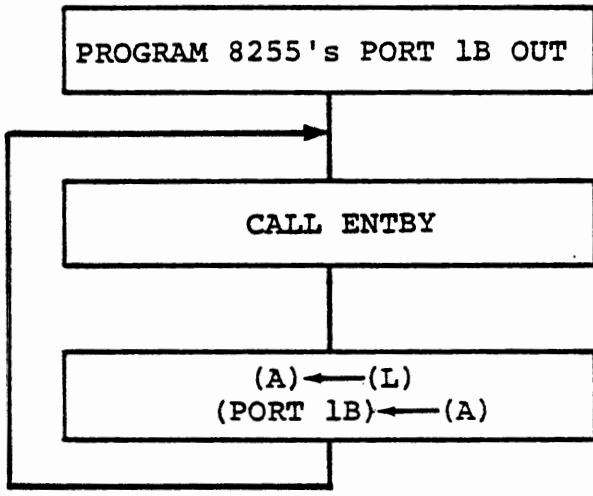
4.6.1 D/A Circuit Input and Output

The 425 has an eight bit port for digital data, connected to port 1B of 8255 #1. These 8 bits control the bipolar switches for the R-2R ladder network. An internal circuit generates a 2.55 volt reference voltage for the ladder, although an external source can be connected. The analog output voltage appears at pin 14 and is connected to two op-amps. One of these (at the upper right of Figure 4-23) has a pot for full scale output adjustment as discussed in Section 4.5, and the op-amp generates a buffered analog output signal available at a tie block. This is the output signal to be used in the experiments of the following sections of Chapter 4.

4.6.2 D/A Circuit Control Signals

A count control signal at pin 2 of the 425 determines whether the eight bit digital data port is to be input to the 425 or output from the 425. When this signal from port 1C0 is low, the 425 accepts digital data from port 1B and converts the binary data to an analog voltage. This is the mode we will use in the remainder of Chapter 4. This signal also forces a NAND gate output high to give an enabling signal to Timer 2 gate input; in this mode, Timer 2 is independent of the D/A circuit and can be used for other purposes. The A/D interrupt control (port 2C3) should be low to inhibit any interrupt from the A/D comparator.

This Page Intentionally Left Blank



KEYBOARD TO VOLTAGE PROGRAM FLOW
AND CIRCUIT CONNECTION
FIGURE 4-28

We will discuss the remaining signals shown in Figure 4-27 in the next chapter, since they are concerned only with A/D input. To operate the 425 in D/A output mode, the following procedure should be used:

```

MVI  A,80      Program 8255 #1
OUT  CNT 1     A out B out C out
MVI  A,92      Program 8255 #2
OUT  CNT 2     A in B in C out

```

This sets count control (1C0) and interrupt control (2C3) low, as well as programming port 1B for output to allow writing data to the D/A converter.

4.6.3 Generating an Analog Voltage

EXERCISE

Write a program to accept data from the keyboard and write the data to the D/A converter. Observe this voltage at the ANALOG OUT tie block with a voltmeter. Adjust the full scale output to make the least significant bit of the data byte correspond to 10 millivolts. Figure 4-28 shows the program flow and the voltmeter connection.

For a 10 millivolt least count, full scale output should be 2.55 volts when the digital value is FF = 255 (10). It is easier to read 2.50 volts on the meter, so key in FA = 250 (10) (remember, you press a command key following the hex value) and adjust the output pot. Now key in various hexadecimal values and see that the output correctly follows the keyed value.

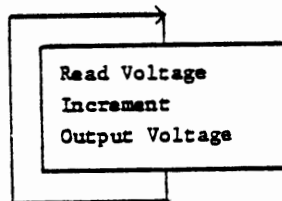
This Page Intentionally Left Blank

4.7 FUNCTION GENERATOR

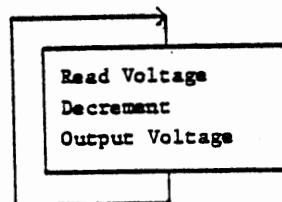
The microprocessor, with the D/A converter, can be used to generate an analog signal that varies over time. If the variation of the signal repeats itself in a predictable manner, the result is a wave. The procedure of repeating a sequence of analog signals over time is called waveform synthesis or function generation. In this section we will experiment with several such functions, including sawtooth and triangular.

Unfortunately, the microprocessor is too slow to generate signals at useful frequencies for most purposes, typically being limited to less than one Hertz. However, some control applications do want very low frequency signals. Another possible use for waveform synthesis is examination of complex waveforms resulting from harmonics or the combination of non-harmonic frequencies, when real-time operation is not required.

Positive Sawtooth



Negative Sawtooth



Triangular Function

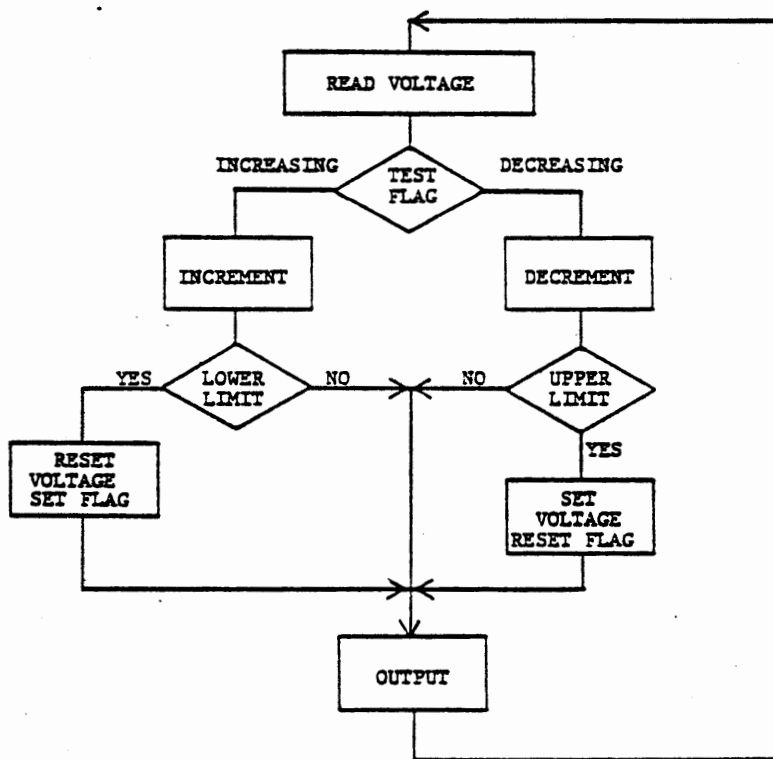
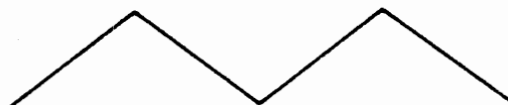


FIGURE 4-29

4.7.1 Voltage Ramps

One of the commonly needed control functions is a voltage ramp - an output voltage that increases or decreases linearly with time. Figure 4-29 shows positive and negative sawtooth functions and a triangular wave generator. The flow diagrams shown could be simple loops in a main program with some delay built in, but more probably each would be in an interrupt service routine invoked by a timer. The rate of increase or decrease is set by the time interval loaded to the timer.

If a full scale sawtooth is to be generated the service routine can read the present output voltage from the output port, increment the value, and output the new voltage. (Remember that a port programmed for output can be read). The service routine can be as simple as this:

PUSH	PSW	Save A and F
MVI	A,01	Re-enable timer 0
OUT	CNT2	Interrupt
IN	PORT1B	Read voltage
INR	A	Increment
OUT	PORT1B	Output voltage
POP	PSW	Restore A,F
EI		
RET		

Change INR A to DCR A for a negative sawtooth.

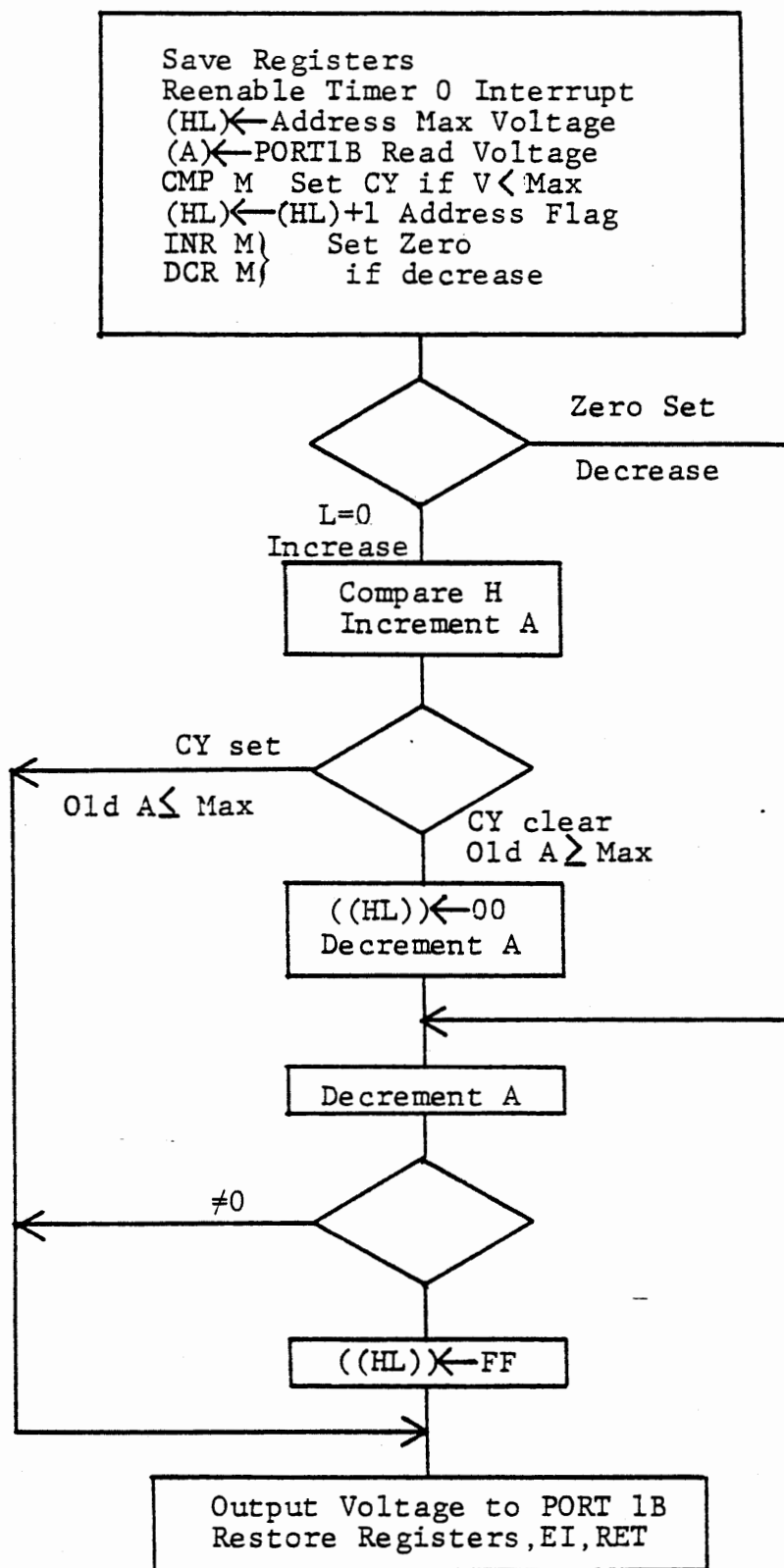
This Page Intentionally Left Blank

4.7.1.1 Voltage Ramp Program

EXERCISE:

Write a program to generate a positive sawtooth waveform using the above interrupt service routine. The main program must initialize the ports and timers. Then it has no further function, so it can end with an instruction that jumps to itself. The signal will appear at the Analog Out tie block. Connect your voltmeter across Analog Out to GND and observe the voltage increase gradually from zero to full scale (about 2.55 volts) and drop back to zero, in cycles of about 8 seconds.

You can generate a negative sawtooth function by changing the INR A instruction in the Interrupt Service Routine to DCR A.



TRIANGULAR FUNCTION GENERATOR

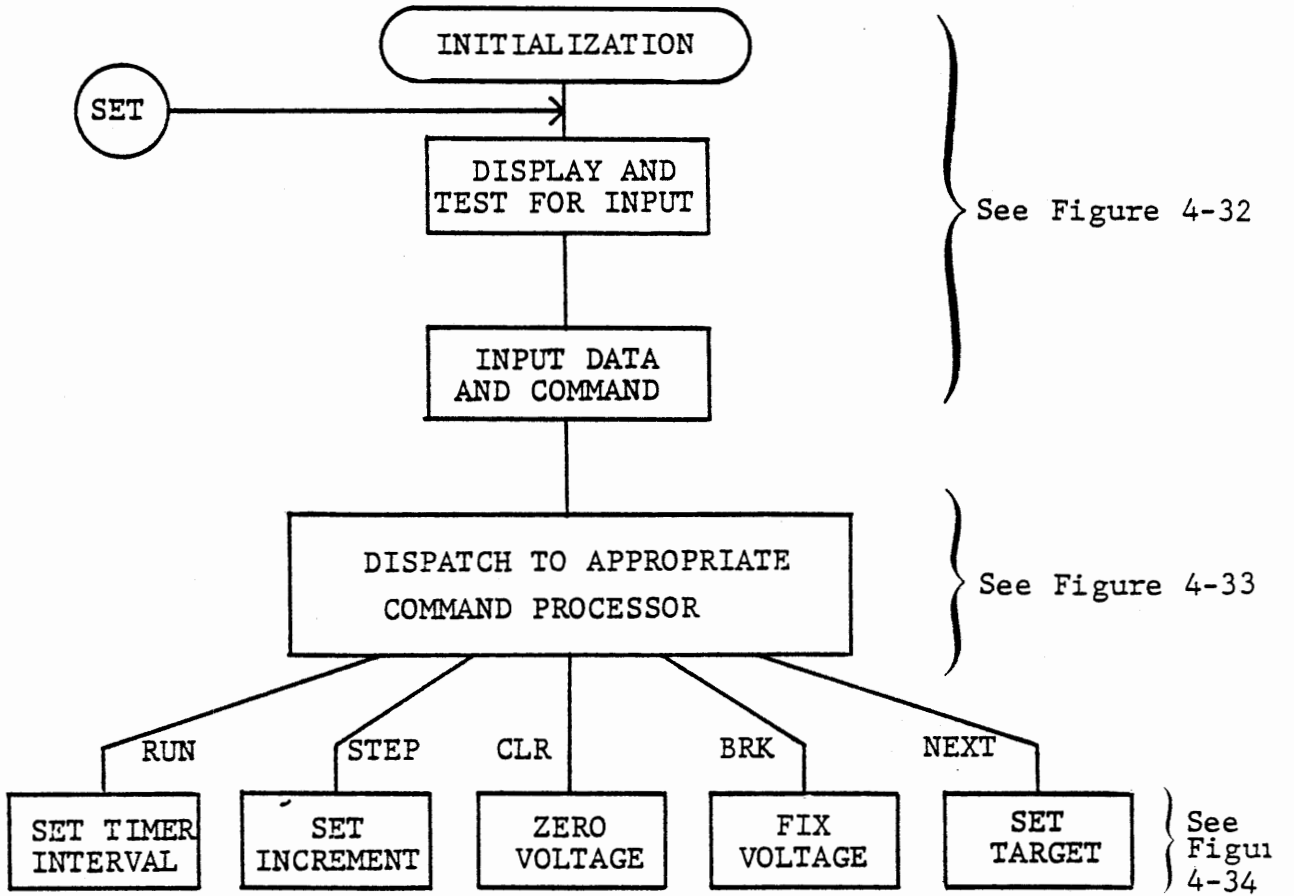
FIGURE 4-30

4.7.1.2 Triangular Wave Program

EXERCISE:

Write a program to generate a triangular waveform. The program must store a flag to indicate whether the voltage is increasing or decreasing. It will also need maximum amplitude data if the output is to be less than full scale. Figure 4-30 is a flow diagram of a service routine to generate a triangular function.

Rewrite the service routine of the previous program to compare the voltage with a maximum amplitude (stored at 8391) and to test an increase (FF) or decrease (00) flag stored at 8392. At the maximum voltage or at zero, reverse the flag. Use fixed values for the maximum amplitude and timer interval, or call for keyboard input of these if you wish. The next major exercise involves keyboard entry of data for a function generator.



KEYBOARD CONTROLLED FUNCTION GENERATOR

FIGURE 4-31

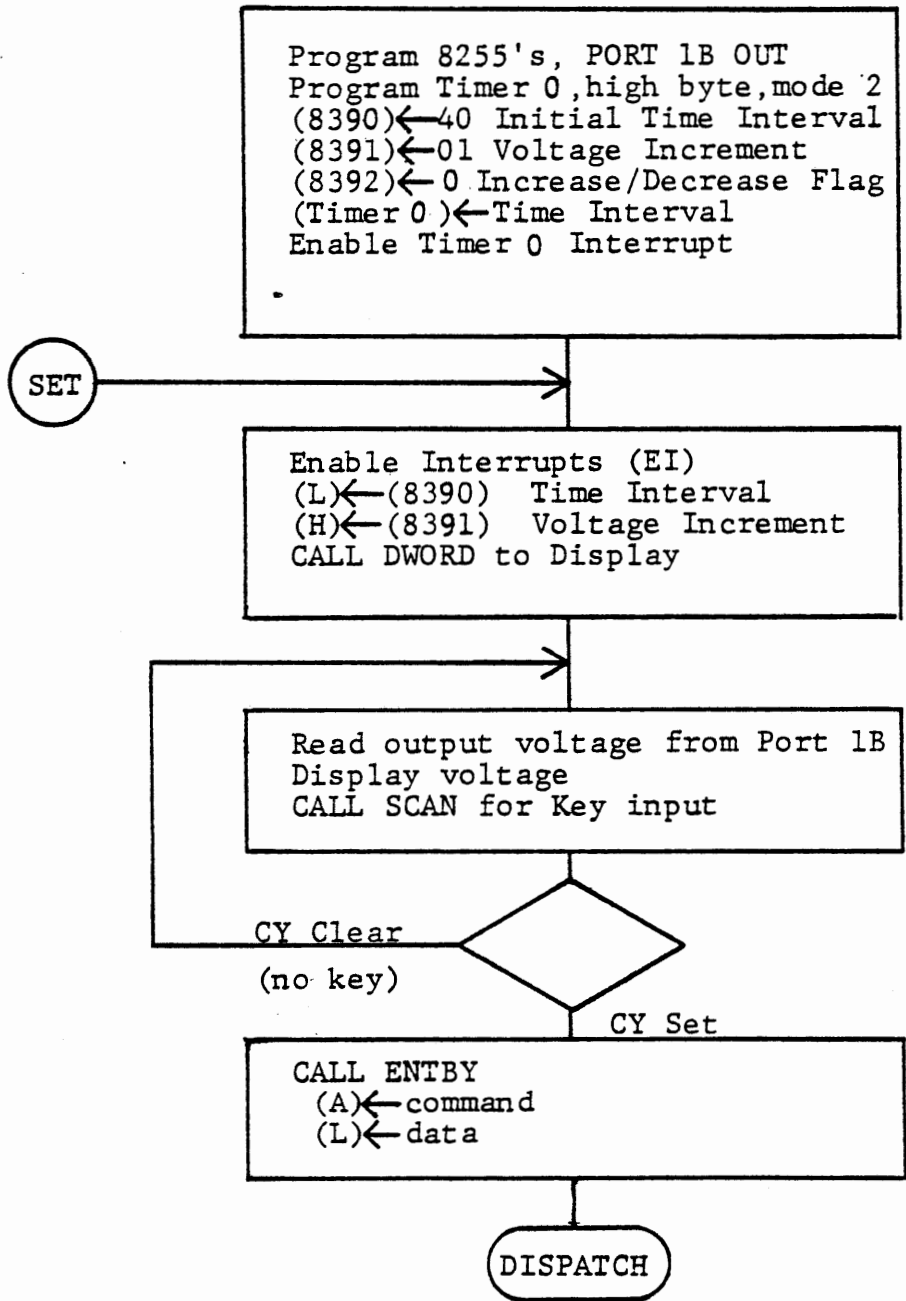
4.7.2 Keyboard Controlled Function Generators

In this and following exercises we shall develop a program to generate several different waveforms selected by keyboard commands. The first exercise will again generate the triangular wave; later an exponential will be generated by numerical integration and a sine will be calculated from a power series. This exercise reviews some important programming techniques: interrupt service, keyboard input, dispatch tables, and using the stack for addresses. It also introduces methods of passing arguments to subroutines, and a variable subroutine call.

EXERCISE:

Write a program that repetitively increases the output voltage toward a target voltage and then gradually decreases it toward the complement of the original target. Accept keyboard data to set the rate of increase or decrease, and the target voltage.

The rate can be adjusted either by adding a variable value to the output data at fixed time intervals (or subtracting the value for the decreasing ramp), or by incrementing (or decrementing) the output at a variable time interval. The latter approach gives a smoother ramp. The program shown in Figure 4-31 and following figures provides both approaches and also permits increasing or decreasing the rate by command key input. Timer 0 provides interval timing; it is used as a rate generator (mode 2) and interrupts the main program to add or subtract the voltage increment to the existing output data. When the output voltage is increased beyond the upper limit by adding the

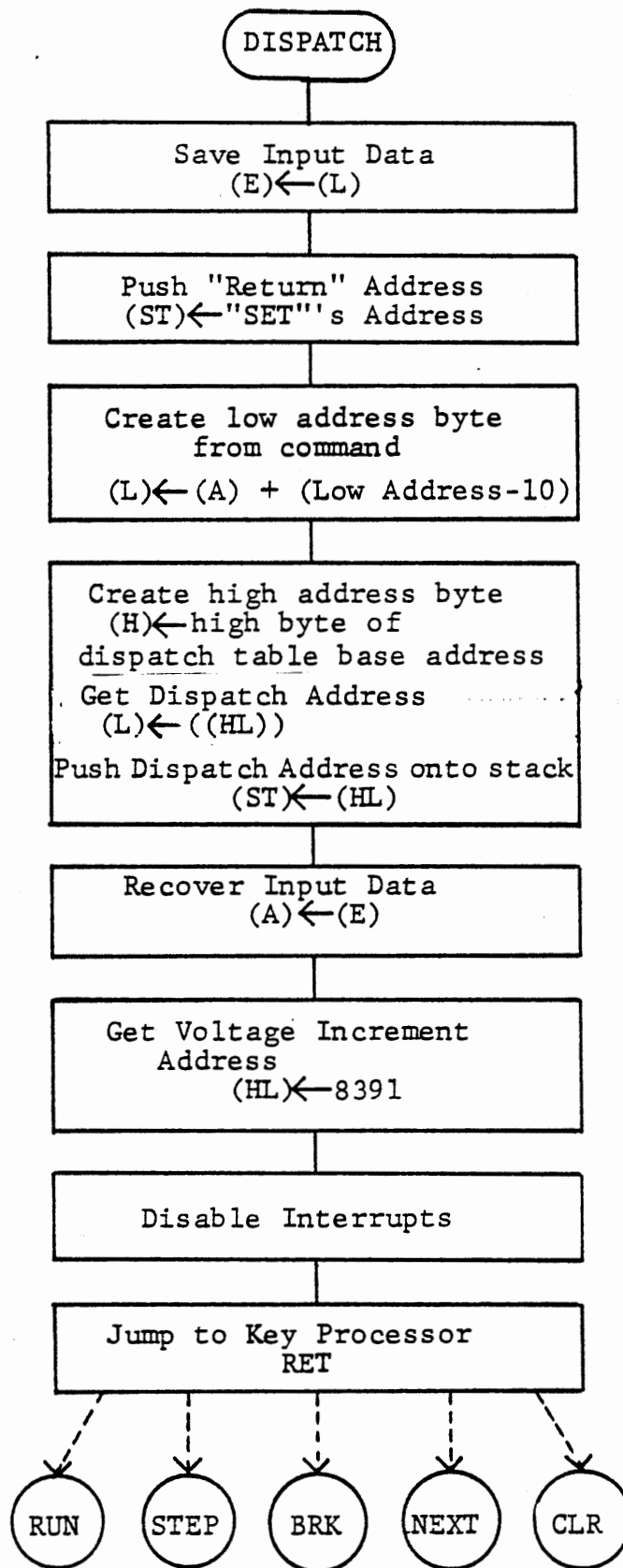


(to Figure 4-33)

KEYBOARD CONTROLLED FUNCTION GENERATOR

FIGURE 4-32

voltage increment, the voltage is set equal to the target and the mode is changed to decrease, and similarly when the output is decreased below the lower limit. This leads to a triangular output wave centered on half scale output. Section 4.7.2.5 describes the process in detail.



(To Figure 4-34)

RAMP - DISPATCH

FIGURE 4-33

4.7.2.1 Main Loop

In the main program (Figure 4-32) the display is controlled to show:

Time interval being used for interrupt

Voltage increment

Present output voltage

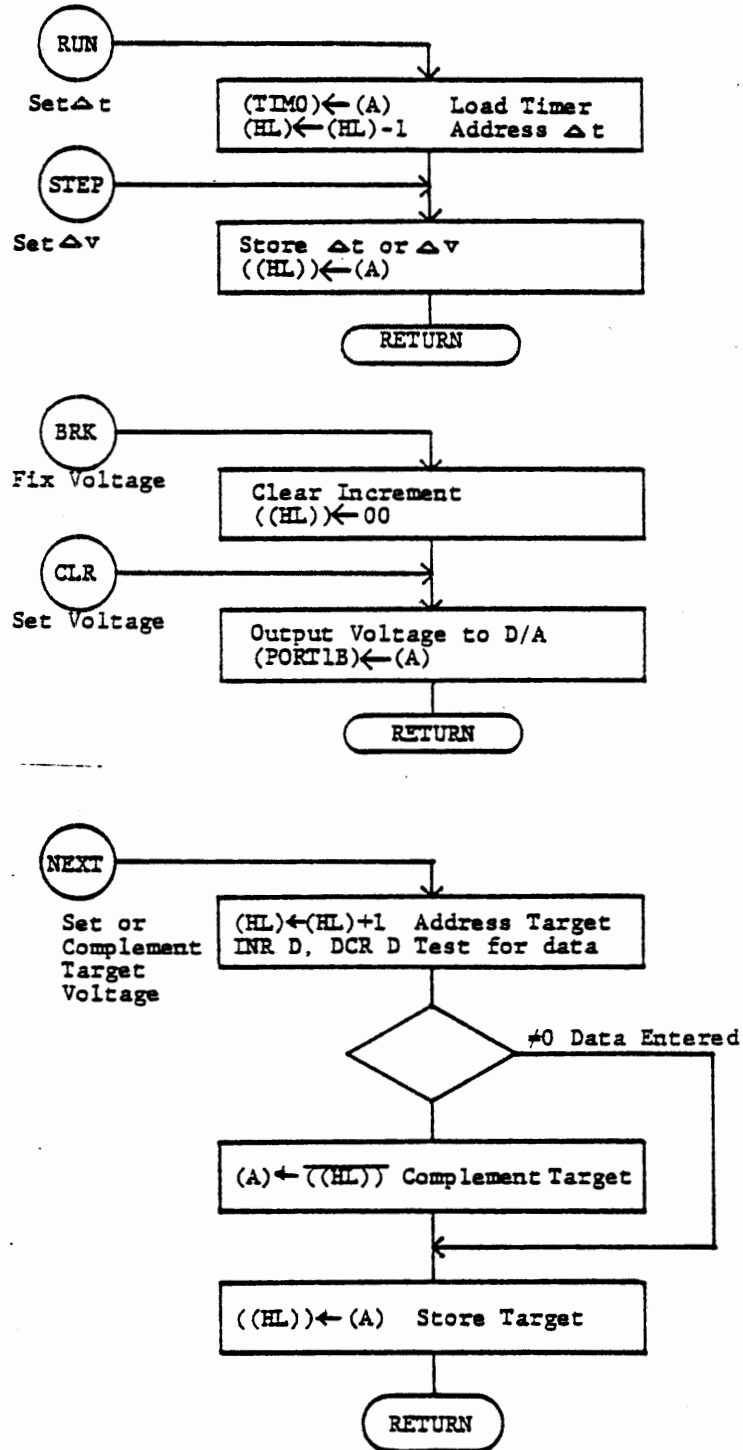
Keyboard inputs are accepted to alter the time interval, the voltage increment, or the target voltage. Numeric entry is optional; a command key is required, and processed as shown in the table below:

RUN	Set time interval
STEP	Set voltage increment
CLR	Start ramp at zero
BRK	Set voltage and clear increment
NEXT	Store or complement target

Figure 4-32 shows the main program. After initialization and display of the initial values it repeatedly reads and displays the output voltage and tests the keyboard. When a key is pressed it calls ENTEY for new data and a command. The command is used to address an appropriate processing module.

Enter key processing module from Dispatch with:

(A) = numeric data keyed (00 if none)
 (HL) = 8391 (address for Δv)



FUNCTION - KEY INPUT PROCESSING
 FIGURE 4-34

4.7.2.2 Dispatch for Keyboard Input

Dispatch is shown in Figure 4-33. We use the technique of dispatch tables and pushing addresses onto the stack: if you have not been using these techniques a review of Course 525, Section 7.4.8 is advisable.

Since all of the processes must return to the display of time interval and voltage increment, we will push that address onto the stack. The various processing modules can end with the return instruction instead of a three byte jump.

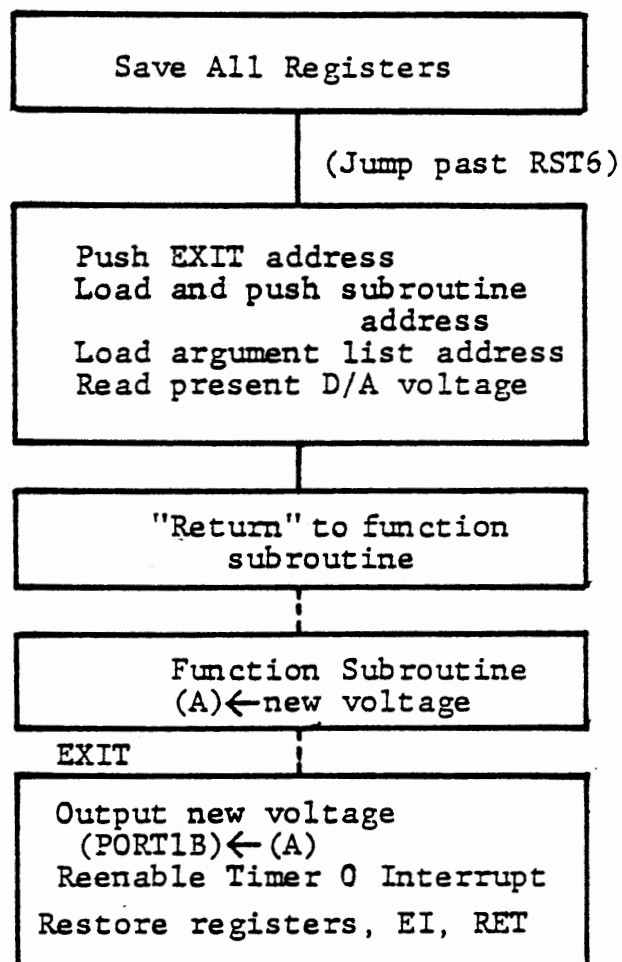
Then we add to the command key value (10H to 17H) the low byte of the dispatch table address minus 10H, so that MEM (10H) will direct us to the first location in the dispatch table. The dispatch low address byte is entered to L. Register H has already been loaded with 82. This dispatch address is pushed onto the stack. To reduce processing in the individual modules we will move the input data into register A and load HL with the address of the voltage increment (8391). Now a return will go to the appropriate module and the return address to the display function will be back on the top of the stack.

This Page Intentionally Left Blank

Some of the key input processing manipulates data that can also be changed by interrupt service. To prevent confusion the interrupts should be disabled while such processing is done. It is convenient to disable the interrupts by DI just before the "return" to the processing module, and enable just after the return from the processing module. Therefore, we have an EI instruction at the start of the main loop.

Since the monitor requires interrupts to operate in debug mode (STEP and BRK) it is desirable to modify the instructions that affect interrupts as you debug various parts of the program. Initially omit the instruction that enables the Timer 0 interrupt. Replace the DI instruction (before the "return" in dispatch) with RST4, so that the monitor will be called immediately before dispatching to the key processing module. Now you can either STEP or RUN through the main loop, using breakpoints as needed, but will always enter the monitor before dispatching. Since the timer interrupt is not enabled you can debug this part of the program even before writing the interrupt service routine.

After the main loop has been checked out replace the EI instruction at the beginning of the loop with a DI. Now the main loop, DWORD and ENTBY will operate without interruptions by the monitor, and you can try the various command entries very easily, always entering the monitor just before dispatch.



TIMER 0 INTERRUPT SERVICE

FIGURE 4-35

4.7.2.3 Key Processing

Figure 4-34 shows the key processing modules. We enter the appropriate module with a return address (to the display function) in the stack, the key input in register A, and the address (8391) of the voltage increment in HL.

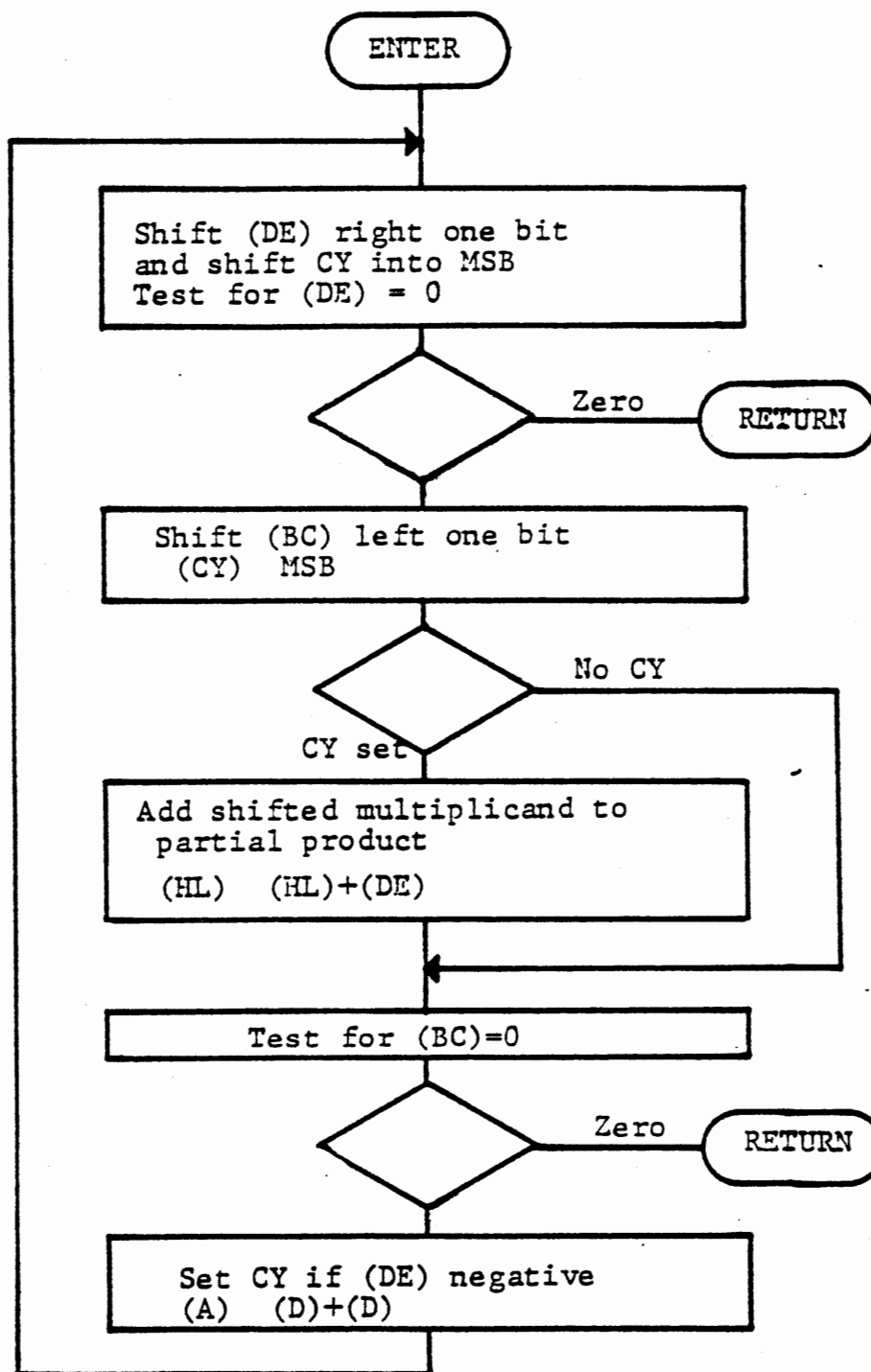
RUN sets the time interval. The input data byte is loaded to Timer 0 and stored at 8390 for display.

STEP stores the input byte at 8391 to set a new voltage increment. Since processing for RUN ends with MOV M,A; RET; we can simply dispatch to the MOV M,A instruction rather than duplicating it.

BRK stops the ramp and sets a fixed voltage. The ramp is stopped by setting the voltage increment (at 8391) to zero. Then the input data byte is written to PORT1B for the D/A output.

CLR writes the input data byte to the D/A output (00 if no data entered).

NEXT either sets a new target voltage (if a data byte was keyed in) or complements the old target voltage. Address the target voltage (8392) by INX H. ENBY returns in register D a count of the number of hex keys entered. If this is zero, recover and complement the old target voltage; otherwise store the new data.



Returns (HL) (HL) + (DE) (BC)

Subroutine BMULT

Figure 4-42

4.7.2.4 Interrupt Service Routine

Interrupt service for timer 0 is shown in Figure 4-35. It performs the normal housekeeping duties of any interrupt service routine. At entry it clears and re-enables the timer 0 interrupt flip-flop, restores the registers, enables interrupts and returns.

To provide for later exercises where other functions will be generated, the interrupt service routine enters a separate subroutine to perform the ramp calculation. Different subroutine calls are allowed (although at present only one will be used) by loading the subroutine's entry address from data memory. The procedure is:

LXI H,EXIT	Push an address for return
PUSH H	from the subroutine
LHLD FUNC	Push the stored entry
PUSH H	address for the subroutine
LXI H,8381	Load a data address
RET	Dispatch to the subroutine

The first function subroutine to be developed (TRI WV) is located at 8250. This address will be stored in memory locations 8398,99 to be loaded by interrupt service. Later we will make this a variable.

TRI WV needs as input data the present voltage, voltage increment, and increase/decrease flag. It returns the new voltage in register A.

We could require the function subroutine to read the voltage from PORT1B, load the other data from the defined addresses in memory, and

output the result to PORT1B. There are three advantages to the method chosen here:

- * The subroutine is "global". Another program could call the subroutine to operate on different data.
- * A different subroutine that needs the same data can be substituted for this one.
- * The function subroutine can be debugged by a temporary calling program.

The data needed by the function subroutine (arguments) can be passed in any of three ways:

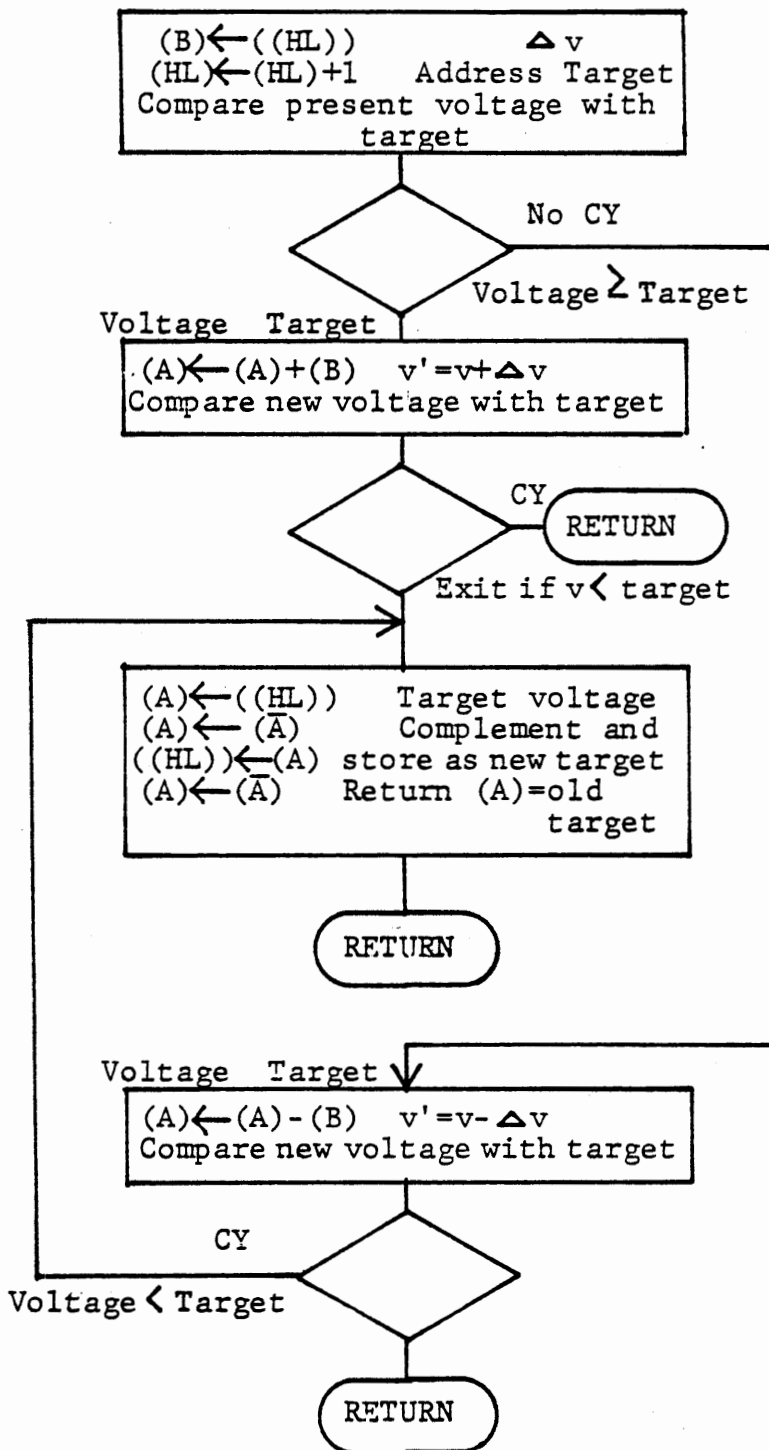
- * Loaded into registers
- * Stored in memory locations reserved for this subroutine and its calling program
- * Stored in memory locations assigned to these particular data, with the address or addresses loaded into registers or into reserved memory locations.

Results can be returned in the same ways. In this program we will combine the first and third methods. The present voltage will be read for PORT1B by interrupt service and passed to the function subroutine in register A. The voltage increment and increase/decrease flag are in their assigned memory locations 8391 and 8392. Address 8391 is loaded into register pair HL and passed to the function. TRIWV

obtains the increment from ((HL)) and the flag from ((HL)+1). The function subroutine would work equally well if some other calling program passed it a different memory address, with different data stored there.

The subroutine returns the new voltage in register A. This is output to the D/A converter by the interrupt service routine as the first step of its exit procedure.

Enter with (A) = present voltage
 ((HL)) = increment
 ((HL)+1) = target voltage



TRIWV FUNCTION SUBROUTINE
 FIGURE 4-36

4.7.2.5 Function Subroutine TRIWV

TRIWV calculates a new voltage by either adding or subtracting the voltage increment to the present voltage. It receives the following "arguments" (input data) from the calling program:

Location	Assignment
(A)	Present voltage
((HL))	Voltage increment
((HL)+1)	Target Voltage

The subroutine is shown in Figure 4-36. The increment is copied to register B and the present voltage is compared to the target to decide whether to add or subtract the increment. The new voltage is calculated and again compared with the target. If it was and still is less than the target, or if it was and still is greater, return with the new voltage in register A. If the voltage has passed the target, however, we will now complement the target voltage to be ready for the next entry, and return with the previous target voltage in register A to be output.

The result is a triangular waveform increasing until it reaches the target voltage and then decreasing to the complement of the voltage. The waveform is centered on 1.275 volts, half of full scale.

Note that TRIWV looks like any normal subroutine. It is not affected by the variable calling procedure, which could equally well be CALL TRIWV.

4.7.2.6 Instructions

Write the complete program in accordance with the flow charts presented. The solution shown in Figure 4-37 (a-f) follows the flow charts precisely and can be referred to if help is needed.

For display of the voltage in the main loop you can CALL DBYTE. For the sake of amusement, the program given in Figure 4-37 calls a subroutine to show the voltage in the LED'S as well as in the seven segment display. This subroutine has not been documented here; you can copy it, or figure it out, or just use DEYTE.

Memory assignments in the solution given are:

8200-8227	Initialize
8228-824F	Interrupt service
8250-827F	TRIWV function subroutine
8280-82A4	Main loop
82A5-82AC	Dispatch table
82AD-82DF	Key processing
82EB-82FF	Display subroutine
8390	Time interval
8391	Voltage increment
8392	Target Voltage
8398,99	Entry address for subroutine

4.7.2.7 Debugging

Debugging techniques for the main program were suggested in Section 4.7.2.2. These are repeated here, referring to addresses in the given solution.

- a) Omit OUT CNT2 at 8223 to avoid enabling timer interrupt.
- b) To debug dispatch loop omit EI at 8280 and DI at 82A3.
- c) To debug key processing modules enter DI at 8280 and RST4 at 82A3.
- d) To debug interrupt service enter RST5 at 8280 and EI at 82A3. Replace EI at end of interrupt service with DI at 824E. Now interrupt service will be called, with monitor interrupts enabled, after each key entry has been processed. The monitor will be disabled during key input and dispatch but enabled during key processing and interrupt service.
- e) To run the debugged program restore all of the modified instructions:

8223	OUT	CNT2
824E	EI	
8280	EI	
82A3	DI	

As you develop your own program keep debugging in mind and provide for similar techniques. Keep a separate list of modified instructions to

be sure you restore them all.

4.7.2.8 Program Operation

With the initial value of 40 for the time interval and 01 for the voltage increment, the output voltage will increase and decrease slowly enough to be observed in the display and on a voltmeter. Using NEXT will reverse the direction part way up or down. With an oscilloscope you can observe the triangular output at higher frequencies. Try entering smaller values of both time interval and voltage increment, keeping a constant ratio so that the total period is the same (see list below). Now observe the effect the on the waveform.

Voltage Increment	Time Interval	Total Period
(STEP)	(RUN)	(Seconds)
01	40	4.096
02	80	4.096
03	C0	4.096
04	00	4.096
01	01	0.064
02	02	0.064
04	04	0.064
08	08	0.064
10	10	0.064
20	20	0.064
40	40	0.064

The CLR key starts the triangle from zero or from any desired value keyed in. This will be of more use in later exercises. BRK stops the function and sets the voltage to any value keyed in. This is convenient for calibrating the A/D output. Request a voltage and adjust the analog out pot to make the output agree with the voltmeter. Operate at a slower rate (press 0, RUN)

Enter 1.55 volts for the target by 9B,NEXT. Press CLR to start a ramp at zero. Observe the voltage climb to 1.55 and then drop to 1.00, (64 hex) and climb again. Press NEXT while it is rising and see it fall.

While the voltage is falling, press CLR to start at zero. Now the voltage will rise toward the 1.0 volt target, complement the target to give 1.55 volts, and continue to climb. Then it will resume the steady rise and fall between 1.0 and 1.55.

This Page Intentionally Left Blank

FUNCTION GENERATOR - INITIALIZE

A	D	D	R	CODE					
8	20	0	3E	MVI	A, 80				Program Ports
		1	80						Port 1B Out
		2	D3	OUT	CNT1				
		3	07						
		4	3E	MVI	A, 92				
		5	92						
		6	D3	OUT	CNT2				
		7	0F						
		8	3E	MVI	A, 24				Program Timer-0
		9	24						High byte
	A	D3	OUT	TIMCT					Mode 2
	B	17							Binary
	C	21	LXI	H, 0140					Initial Values
	D	40							
	E	01							
	F	22	SHLD	8390					
8	21	0	90						(L) ← Δt
		1	83						(8390) ← Δt
		2	AF	XRA	A				(8391) ← ΔN
		3	32	STA	8392				(8392) ← target
		4	92						(will be changed
		5	83						by interrupt)
		6	D3	OUT	PORT1B				(D/A) ← 00
		7	05						
		8	7D	MOV	A, L				
		9	D3	OUT	TIM0				(TIM1) ← Δt
	A	14							
	B	21	LXI	H, 8250					Store address
	C	50							
	D	82							of TRIWV
	E	22	SHLD	8398					subroutine for
	F	98							
									interrupt service
8	22	0	83						
		1	3E	MVI	A, 01				Enable Timer 0
		2	01						interrupt
		3	D3	OUT	CNT2				
		4	0F						
		5	C3	JMP	8280				Jump to main loop
		6	80						
		7	82						
		8							

FIGURE 4-37a

FUNCTION GENERATOR- EXIT FROM INTERRUPT

4-111

A D D R		CODE																		
CODING SHEET	8	0																		
		1																		
		2																		
		3																		
	824	4	D3																	
		5	05																	
		6	3E																	
		7	01																	
		8	D3																	
		9	0F																	
MICROCOMPUTER TRAINING SYSTEM	A	CI																		
	B	DI																		
	C	EI																		
	D	FI																		
	E	FB																		
	F	C9																		
INTEGRATED COMPUTER SYSTEMS	8	0																		
		1																		
		2																		
		3																		
		4																		
		5																		
		6																		
		7																		
		8																		
		1																		
		2																		
		3																		
		4																		
		5																		
		6																		
		7																		
	8																			

FIGURE 4-37c

FUNCTION TRIWV

A	D	D	R	CODE	FUNCTION	TRIWV	
8	2	5	0	46	MOV	B, M	(B) ← ΔN
			1	23	INX	H	Address target
			2	BE	CMP	M	Compare voltage
			3	D2	JNC	8260	If voltage ≥ target
			4	60			go to subtract
			5	82			
			6	80	ADD	B	(A) ← N' = N + ΔN
			7	DA	JC	8266	If beyond FF go
			8	66			to reverse
			9	82			
	A			BE	CMP	M	Compare new
		B		D2	JNC	8266	voltage to target
		C		66			If past target
		D		82			go to reverse
		E		C9	RET		Exit if not past
		F		00	NOP		
8	2	6	0	90	SUB	B	(A) ← N' = N - ΔN
			1	DA	JC	8266	If below 00 go
			2	66			to reverse
			3	82			Compare new
			4	BE	CMP	M	voltage to target
			5	D0	RNC		Exit if not past
8	2	6	6	7E	MOV	A, M	Reverse
			7	2F	CMA		Complement target
			8	77	MOV	M, A	voltage
			9	2F	CMA		Return with
		A		C9	RET		N = old target
		B					
		C					
		D			ENTER WITH		
		E			(A) = PRESENT VOLTAGE		
		F			((HL)) = INCREMENT, ΔN		
8			0		((HL) + 1) = TARGET VOLTAGE		
			1				
			2		RETURN		
			3		(A) = NEW VOLTAGE		
			4		(HL) ADDRESSING TARGET		
			5		AT OVERFLOW RETURN		
			6		(A) = OLD TARGET		
			7		TARGET COMPLEMENTED		
			8				FIGURE 4-37d

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

FUNCTION - KEY PROCESSING

4-114

		A	D	D	R	CODE					
CODING SHEET	8	0									
		1									
		2									
		3									
		4									
MICROCOMPUTER TRAINING SYSTEM	82A	5	A4			MEM					Undefined
		6	A4			REG					↓
		7	A4			ADDR					
		8	B0			STEP					Set ΔN
		9	AD			RUN					Set Δt
		A	C0			NEXT					Set or reverse target
		B	B2			BRK					Set fixed voltage
		C	B4			CLR					Set voltage
	82A	D	D3			OUT	TIM0				RUN Set Δt
		E	14								Load Timer 0
INTEGRATED COMPUTER SYSTEMS		F	2B			DCX	H				Address Δt
	82B	0	77			MOV	M, A				STEP Set ΔN
		1	C9			RET					
	82B	2	36			MVI	M, 00				BRK set fixed N
		3	00								Clear ΔN
	82B	4	D3			OUT	PORT1B				CLR set N
		5	05								
		6	C9			RET					
		7									
		8									
	9										
	A										
	B										
	C										
	D										
	E										
	F										
	8	0									
	1										
	2										
	3										
	4										
	5										
	6										
	7										
	8										

FIGURE 4-37 f

FUNCTION - KEY PROCESSING cont'd 4-115

A D D R		CODE							
CODING SHEET	8 2 C 0	2 3		I N X	H				NEXT Set Target
		1 4		I N R	D				Test for data entered
		2 5		D C R	D				
		3 2		J N Z	8 2 C 8				If data entered
		4 8							jump to store as
		5 2							target voltage
		6 E		M O V	A 2 M				Else complement
		7 F		C M A					old target
		8 2 C 8	7 7		M O V	M 2 A			Store target
MICROCOMPUTER TRAINING SYSTEM		9 9		R E T					
		A							
		B							
		C							
		D							
		E							
		F							
		8 0							
		1							
		2							
		3							
		4							
		5							
		6							
		7							
	INTEGRATED COMPUTER SYSTEMS		8						
		1							
		2							
		3							
		4							
		5							
		6							
		7							
	8								

FIGURE 4-37g

DISPLAY VOLTAGE SUBROUTINE

	A	D	D	R	CODE															
CODING SHEET	8		0																	
			1																	
			2																	
			3																	
			4																	
			5																	
			6																	
			7																	
			8																	
			9																	
MICROCOMPUTER TRAINING SYSTEM			A																	
	82E	B	CD		CALL	DBYTE				Display voltage										
		C	95																	
		D	02																	
		E	79		MOV	A, C				(A) ← voltage										
		F	21		LXI	H, 00FF				To shift ones										
	82F	0	FF							into (H)										
		1	00							Divide by 29 to										
	82F	2	D6		SUI	1D				divide voltage range										
		3	1D							into 9 segments										
	4	DA		JC	82FB				When remainder											
	5	FB							less than 0											
	6	82							go to display											
	7	29		DAD	H				Shift in ones											
	8	C3		JMP	82F2				Loop											
	9	F2																		
		A		82																
INTEGRATED COMPUTER SYSTEMS	82F	B	7C		MOV	A, H				Display result										
		C	D3		OUT	PORT1A				in LED's										
		D	04																	
		E	C9		RET															
		F	00		NOP															
INTEGRATED COMPUTER SYSTEMS	8		0																	
			1																	
			2																	
			3																	
			4																	
			5																	
			6																	
			7																	
		8																		

FIGURE 4-37h

4.7.3 Exponential Function

The charging of a capacitor leads to a voltage which increases with time at a slowing rate as the capacitor voltage approaches the source voltage. (See Figure 4-33). The capacitor voltage is given by:

$$(a) \quad v = Q/C$$

where Q is the accumulated charge and C the capacitance. The charge is accumulated as current flows into the capacitor and is calculated from the time integral of the current.

$$(b) \quad Q = \int_0^t i dt$$

The current through the resistor is proportional to the voltage across the resistor: the source voltage V_s minus the capacitor voltage at that instant.

$$(c) \quad i = \frac{1}{R} (V_s - v)$$

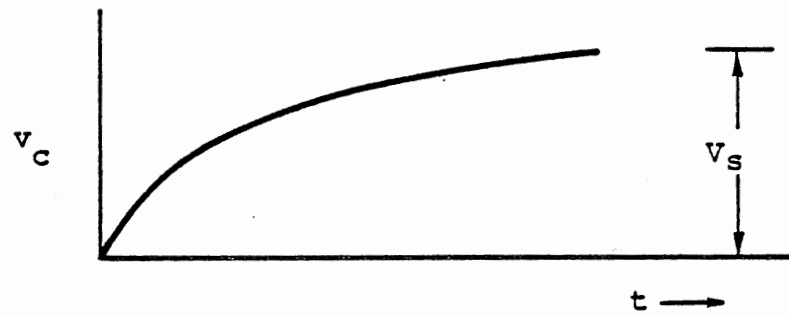
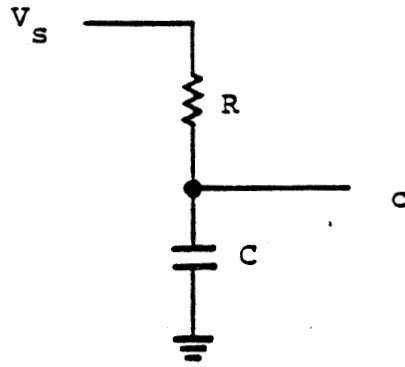
then

$$(d) \quad v' = \frac{1}{RC} \int_0^t (V_s - v) dt$$

which can be solved to give:

$$(e) \quad v' = V_s (1 - e^{-t/RC})$$

As t increases without limit the exponential term approaches zero and the capacitor voltage approaches the source voltage. With a digital computer we can solve the integral equation (d) numerically without resort to the explicit solution (e); doing so can generate the exponential function.



$$v_c = \frac{1}{RC} \int_0^t (V_s - v_c) dt$$

$$v_c = V_s (1 - e^{-t/RC})$$

EXPONENTIAL FUNCTION

FIGURE 4-38

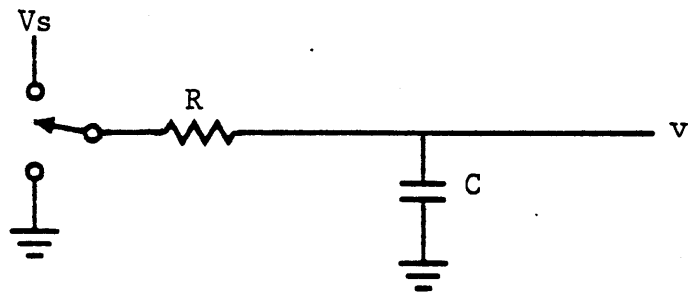
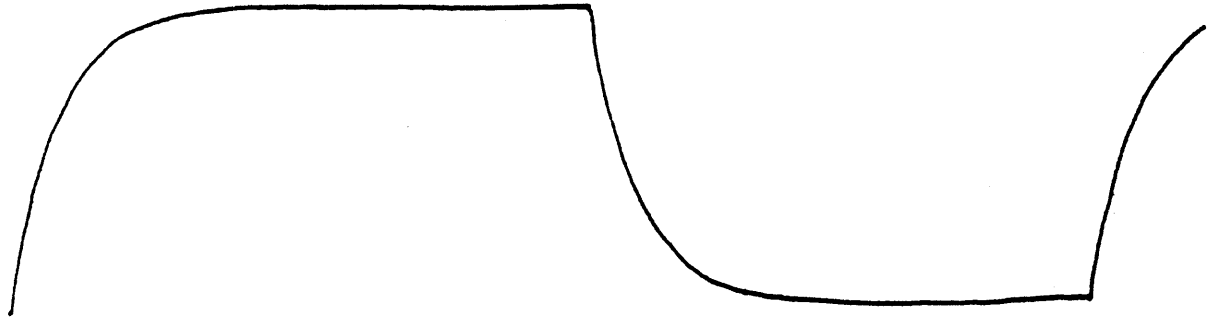
For numerical integration there is no infinitesimal time, so equation (d) is rewritten as:

$$v' = v + (V_s - v) \Delta t / RC$$

In this section we shall create a function subroutine to evaluate equation (f), to be called in place of TRIWV in our function generator program.

As the output voltage v approaches the source voltage V_s , it changes very slowly. To obtain a good representation of the exponential we must use two byte precision for the calculation. We will store the less significant byte of v in memory, and the more significant byte will be read from and output to PORT 1B.

This Page Intentionally Left Blank



SUCCESSIVE CHARGE/DISCHARGE CYCLES

FIGURE 4-39

Eventually v will stop changing, even though it is not yet quite equal to V_s . At this point we will start a discharge period, in which the source voltage is zero. Then:

$$(g) \quad v' = v + (0-v) \Delta t/RC$$

Again, after some time, v will stop changing and we will switch back to the charging function. Successive charge/discharge cycles are shown in figure 4-39.

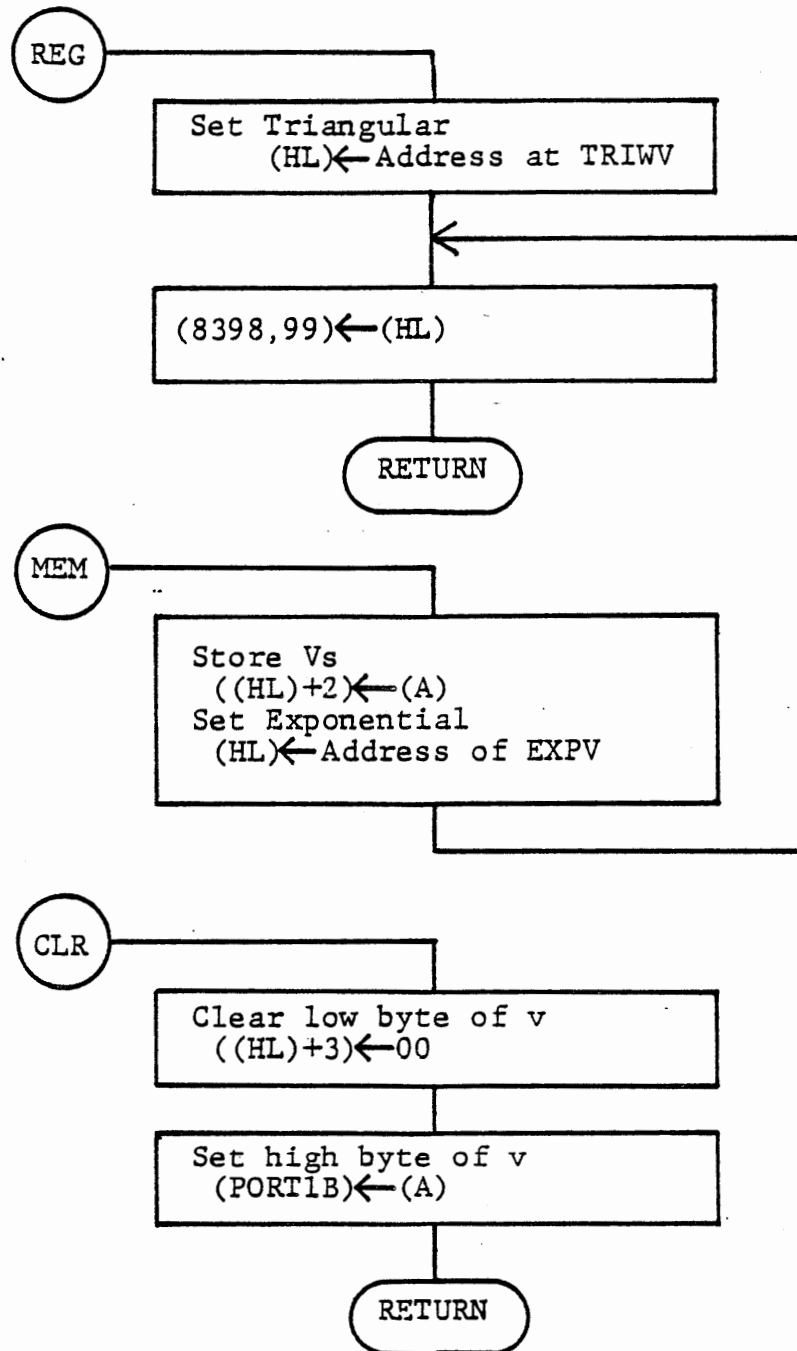
We will store two variables to control the charge or discharge. The source voltage V_s will be stored as hex data entered with MEM. A "switch variable" will represent the state of the switch shown in figure 4-39. The "target voltage" stored or complemented by NEXT (at 8392) will be used as the switch variable: if its most significant bit is one the voltage will increase toward V_s ; if the most significant bit is zero the voltage will decrease toward ground. Thus the function of the NEXT key is preserved.

4.7.3.1 Exponential Waveform Generator

EXERCISE:

Modify and add to the program of Section 4.7.2 to generate either a triangular wave or an exponential. Accept keyboard input of source voltage (V_s), time interval Δt , and the ratio $\Delta t/RC$.

Data will be entered as one byte values with command keys, as before. The keys are defined below . Key input processing is shown in Figures 4-34 and 4-40, and described in the following subsections.



KEY SELECTION OF WAVEFORM

FIGURE 4-40

4.7.3.2 Selecting the Waveform

Since the waveform is generated by interrupt service, this module must behave differently for the two different waveforms. The distinction is handled by storing a jump address in memory (at 8398, 99) according to the REG (for triangular) or MEM (for exponential) command. The interrupt service routine of the function generator program (Section 4.7.2) provides for the use of this variable subroutine address by:

```

LXI H, EXIT      PUSH exit address
PUSH H
LHLD 8398        PUSH subroutine address
PUSH H
LXI H, 8391      Load data address
RET              Dispatch to selected subroutine

```

4.7.3.3 Data Entry and Storage

As indicated in the table of 4.7.3.1 there are two new variable data bytes to be stored in addition to the function address. The less significant byte of v is calculated and stored by EXPV. It is also to be cleared by the BRK and CLR keys. This is necessary in order to obtain a consistent result each time CLR is used.

The source voltage V_s is to be entered by MEM (if a hex data byte is entered.) For two byte precision in the calculation we will use this value as the high byte, with zero for the low byte.

MEM also selects the exponential waveform by storing the address of EXPV at 8398, 99. A value for V_s must be entered with MEM. REG is to select the triangular waveform by storing the address of TRIWV at 8398, 99.

RUN, STEP and NEXT are unchanged:

RUN loads timer 0 and stores Δt at 8390.

STEP stores Δv or $\Delta t/RC$ at 8391.

NEXT stores the switch variable or target voltage at 8392. If no data were entered NEXT complements the old value.

Memory assignments are:

8390	Δt
8391	Δv or $\Delta t/RC$
8392	Target voltage or switch variable
8393	V_s
8394	v (low byte)
8398,99	Function address
8250	Entry to TRIWV
8320	Entry to EXPV

4.7.3.4 Calculating Exponential Voltage

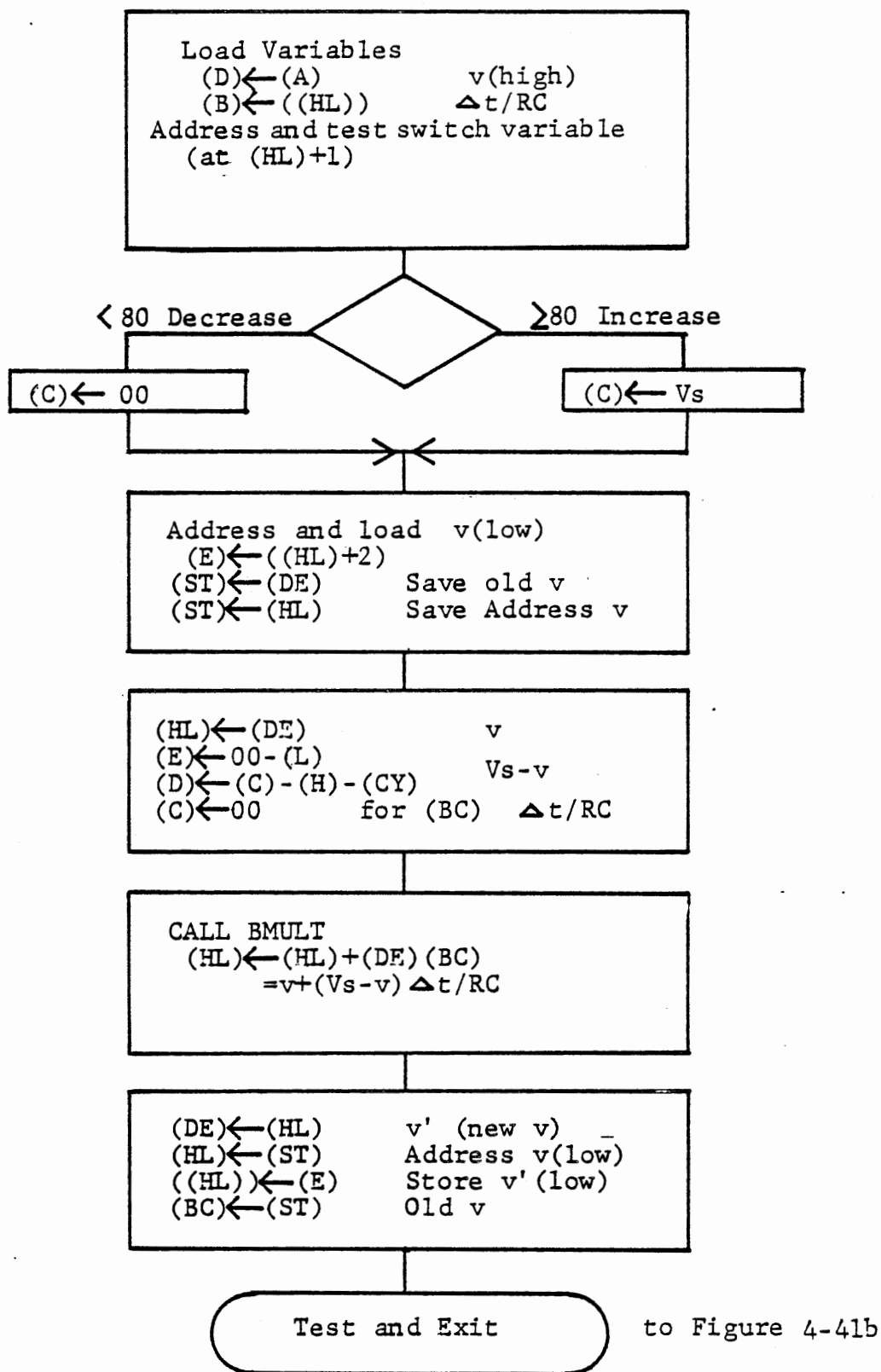
Subroutine EXPV, shown in figure 4-41a calculates

$$v' = v + (V_s - v)(\Delta t / RC)$$

At entry, register A contains the high byte of v , and register pair HL contains the address of the memory location for $\Delta t / RC$, a single byte value. The other variables are contained in successive locations:

Offset	Nominal	Variable
Address	Address	
(HL)	8391	$\Delta t / RC$
(HL)+1	8392	switch variable
(HL)+2	8393	V_s (high byte)
(HL)+3	8394	v (low byte)

Although specific memory addresses are listed above, the subroutine will use only offset addressing, so that it could be called from another program module with different data in different storage locations.



EXPV - CALCULATE VOLTAGE

FIGURE 4-41a

The subroutine moves the input voltage into register D and loads the other variables, incrementing (HL) to access successive bytes. It tests the switch variable and either loads V_s into (C) from memory if the voltage is to increase or clears C if the voltage is to decrease. Then the low byte of v is loaded to register E, so that (DE) contains the two byte value of v . Both v and its address are saved in the stack, and the calculation begins.

$$v' = v + (V_s - v)(\Delta t / RC)$$

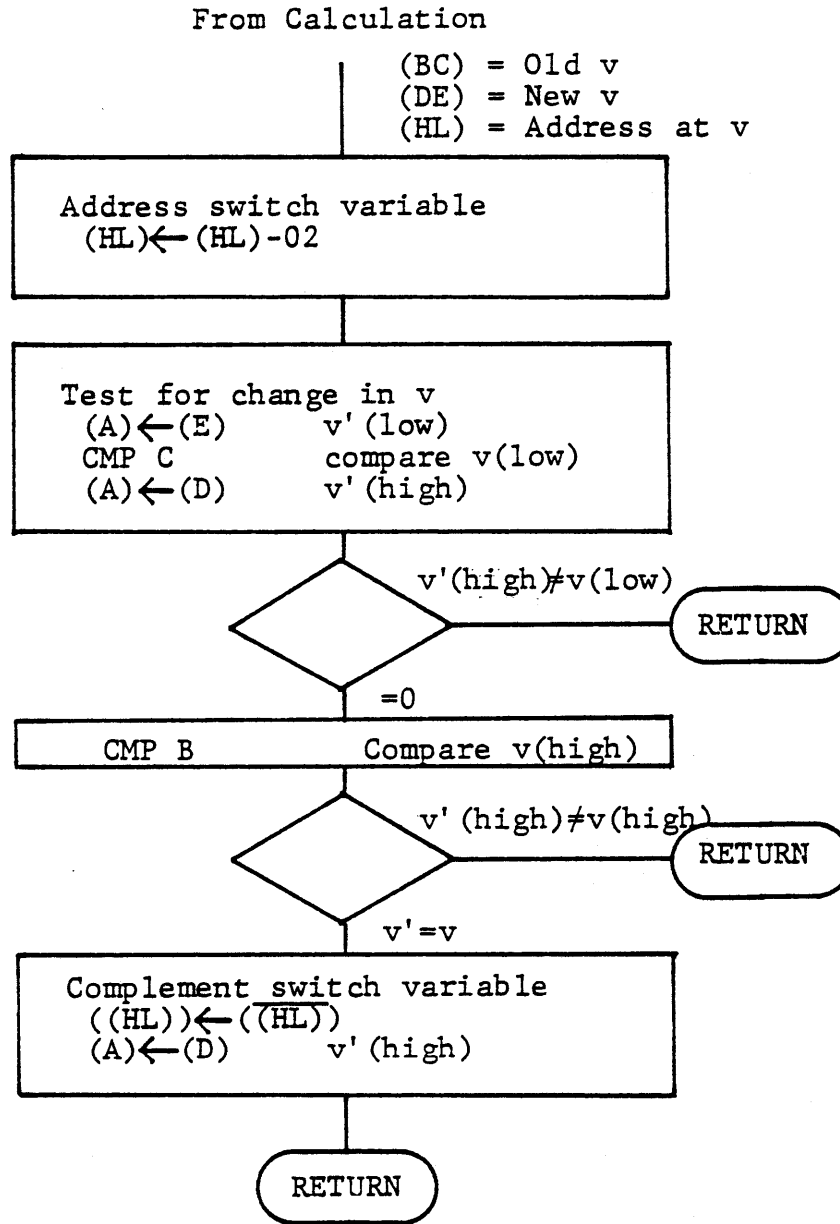
is to be evaluated. We shall develop a subroutine (BMULT) to evaluate this expression with the following entry data, all as two byte variables.

(BC)	=	$\Delta t / RC$
(DE)	=	$V_s - v$
(HL)	=	v

Before calling BMULT, the value of $V_s - v$ is calculated by EXPV as follows: Move v into (HL), and subtract its low byte from zero, placing the result in (E). Subtract the high byte of v from V_s (or zero if the voltage is decreasing) using the SBB instruction (subtract with borrow) and place the result in (D). Clear the low byte of $\Delta t / RC$ (register C) and call BMULT.

The new value of v is returned in register pair HL. It is moved to DE, the address for v (low) is popped and its new value is stored. The old value of v is popped into (BC) for comparison.

This Page Intentionally Left Blank



EXPV - TEST FOR CHANGE IN VOLTAGE

FIGURE 4-41b

Figure 4-41b shows the procedure for changing the switch variable. The new value of v is compared with the old value. If it has changed, the process will continue in the same direction. When v no longer changes, the switch variable is complemented.

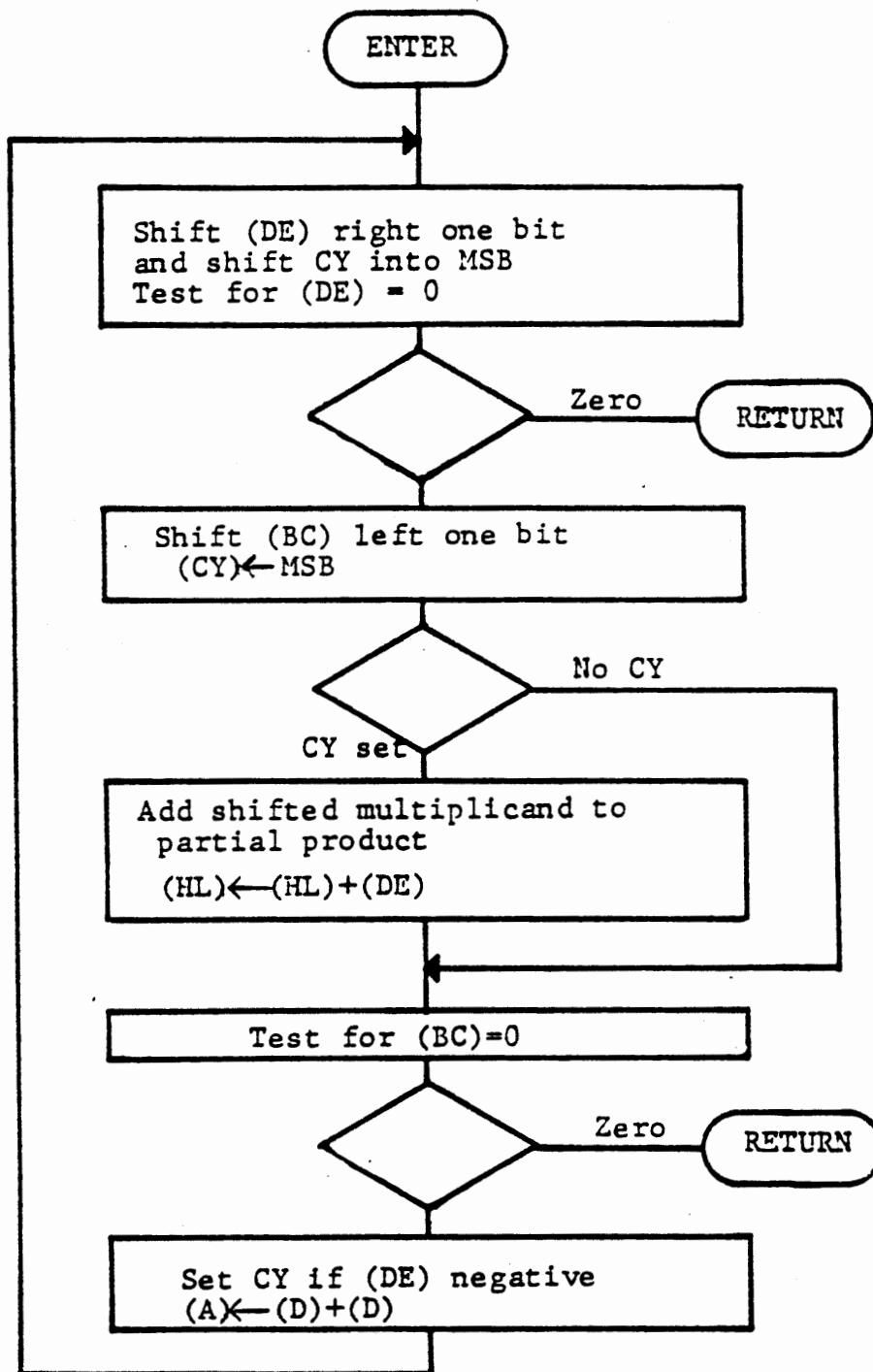
4.7.3.5 Subroutine BMULT

Both V_s and v are positive values that can range from 0000 to FFFF. $V_s - v$ can range from -FFFF to FFFF. The two byte subtract gives these values with CY set if the result is negative. For example:

V_s	v	(CY)	(DE)	$V_s - v$
FFFF	0000	0	FFFF	+FFFF
FFFF	FFFE	0	0001	+0001
0000	0001	1	FFFF	-0001
0000	FFFF	1	0001	-FFFF

Thus (CY) and (DE) represent a 17 bit twos complement value.

We will design BMULT to accept the data in this form and perform a correct multiplication with a positive or negative value in DE as marked by the CY flag. The resulting value of v' (in HL) will always be positive, since V_s , V , and $\Delta t/RC$ are all positive. The value of v' will be greater or less than v , depending on the sign of $V_s - v$.



Returns $(HL) \leftarrow (HL) + (DE) (BC)$

Subroutine BMULT

Figure 4-42

The multiplication will be done by repeatedly shifting the multiplicand (DE) right and the multiplier (BC) left. If the bit shifted out of the multiplier is a one, add the shifted multiplicand to the partial product (HL).

The first shift of the multiplicand will enter the carry bit into the high bit of the multiplicand, so that the shifted value will retain a proper twos complement form. At each loop the carry must be restored to its original state by copying the high bit from (D) into the carry. The table below shows successive values of the multiplicand for the two cases where it was initially +4000 or -4000 (represented as C000 with carry set).

CY Clear (+)	CY Set (-)
4000	C000
2000	E000
1000	F000
0800	F800
0400	FC00
0200	FE00
0100	FF00
0080	FF80
0040	FFC0
0020	FFE0
0010	FFF0
etc.	etc.

There is no rounding of the value as it is shifted. It is not necessary for the precision required, and any simple rounding procedure will lead to erroneous results.

4.7.3.6 Implementing the Program

Much of the exponential program is a modification of the previous program. The memory locations suggested in Section 4.7.2.6 allow room for these added functions, but EXPV and BMULT will not fit in page 8200. They are located at 8320 and 8300 in the solution given in Figure 4-45, but if you have more than 512 bytes of memory you may prefer to locate them elsewhere.

To debug EXPV and BMULT it is probably easiest to avoid the main loop and interrupt service altogether. Instead of running from 8200, use the debugging program shown as the first pages of Figure 4-43.

You can enter a value for v through the keyboard. With no data entered the previous value is recovered. Then you can step through the subroutine. When you are satisfied that program flow is correct and stack usage is balanced, repeatedly press NEXT, and successive values will be generated and displayed. Breakpoints can also be used in BMULT or EXPV. Note that the multiplier $\Delta t/RC$ has been set to 50 (01010000); this is a nice value for testing BMULT because the multiplication process can be observed during the first four bits, but will terminate quickly. It reaches the switch point fairly quickly (32 iterations) which is also convenient.

When you are satisfied with your subroutines restore the operating program, but with the debugging modifications suggested in Section 4.7.2.7. Check the program flow, and then try the full program in AUTO mode.

4.7.3.7 Program Operation

Initialization sets $\Delta t = 40$ and $\Delta t/RC = 01$. No value is entered for V_s , but your debugging may have left the value FF in V_s .

The numeric display, the LED's, and the voltmeter will show the exponential rising quickly at first and slowing until it seems to stop at FD. Now only the less significant byte is changing. Suddenly the voltage will drop, and then approach zero very slowly.

With the ratio $\Delta t/RC$ fixed, an increase in the time interval Δt (entered with RUN) will also represent an increase in the time constant, so charging will take the same number of steps, but more time. An increase in $\Delta t/RC$ (entered with STEP) will result in larger and fewer steps to reach the source voltage, and therefore less time. If Δt and $\Delta t/RC$ are both increased proportionately, the time constant will remain constant and a coarser approximation of the same waveform will be obtained. If an oscilloscope is available this will be interesting to observe. The total cycle time will not be constant for corresponding values of Δt and $\Delta t/RC$ because the slow final approach to the source voltage is not calculated precisely.

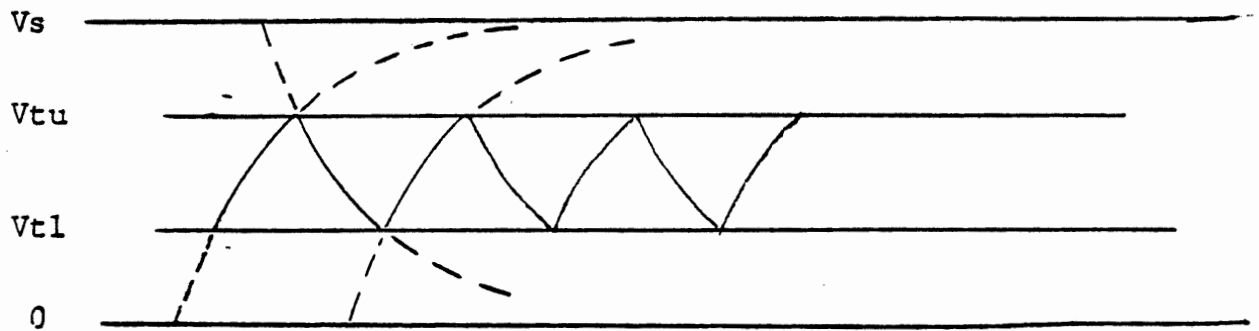
Restore the original values for Δt and $\Delta t/RC$ and enter a lower value for V_s .

40, RUN

01, STEP

80, MEM

Now the exponential will rise only to half scale, taking the same time as before, and then reverse.



EXPONENTIAL WITH THRESHOLDS

FIGURE 4-41

TEST PROGRAM FOR EXPV

4-140

	A	D	D	R	CODE							
CODING SHEET	8	3	6	0	21	LXI	H	FF01				
				1	50							
				2	FF						(8391) ← 50 for ΔE/RC	
				3	22	SHLD		8391			(8392) ← FF for switch	
				4	91							
				5	83							
				6	21	LXI	H	00FF				
				7	FF							
				8	00						(8393) ← FF for Vs	
				9	22	SHLD		8393			(8394) ← 00 for N low	
MICROCOMPUTER TRAINING SYSTEM	A				93							
	B				83							
	C				6C	MOV	L	H			(HL) ← 00 for N	
	836	D			F3	DI					LOOP	
		E			E5	PUSH	H				Save result	
		F			CD	CALL	DWORD				Display result	
	837	0			D1							
		1			02							
		2			CD	CALL	ENTWD				Accept new data	
		3			46						if desired	
INTEGRATED COMPUTER SYSTEMS		4			03							
		5			14	INR	D				Test for data	
		6			15	DCR	D				entered	
		7			D1	POP	D					
		8			FB	EI						
		9			CA	JZ		8380			Jump if no data	
	A				80							
	B				83							
	C				22	SHLD		8394			Store (L) as N low	
	D				94							
E				83								
F				EB	XCHG					(D) ← N high		
8	0											
	1											
	2											
	3											
	4											
	5											
	6											
	7											
	8											

TEST PROGRAM continued

	A	D	D	R	CODE							
CODING SHEET	8	3	8	0	7A	MOV	A, D				(A) ← N high	
				1	21	LXI	H, 8391				Address ΔT/RC	
				2	91						for EXPV	
				3	83							
				4	CD	CALL	EXPV					
				5	20							
				6	83							
				7	EB	XCHG					(HL) ← N	
				8	C3	JMP	836D					
				9	6D							
MICROCOMPUTER TRAINING SYSTEM	A				83							
	B											
	C											
	D											
	E											
	F											
	8			0								
				1								
				2								
				3								
				4								
				5								
				6								
				7								
				8								
	INTEGRATED COMPUTER SYSTEMS	A										
B												
C												
D												
E												
F												
8				0								
				1								
			2									
			3									
			4									
			5									
			6									
			7									
			8									

FUNCTION GENERATOR - INITIALIZE

4-142

		A	D	D	R	CODE			
CODING SHEET	8	20	0	3E		MVI	A, 80		Program Ports
			1	80					Port 1B Out
			2	D3		OUT	CNT1		
			3	07					
			4	3E		MVI	A, 92		
			5	92					
			6	D3		OUT	CNT2		
			7	0F					
			8	3E		MVI	A, 24		Program Timer 0
			9	24					High bits
MICROCOMPUTER TRAINING SYSTEM	A		D3		OUT	TIMCT			Mode 2
	B		17						Binary
	C		21		LXI	H, 0140			
	D		40						
	E		01						Initial values
	F		22		SHLD	8390			
	8	21	0	90					(L) ← Δt
			1	83					(8390) ← Δt
			2	AF		XRA	A		(8391) ← ΔN
			3	32		STA	8392		(8392) ← target
		4	92					(will be changed by interrupt)	
		5	83						
		6	D3		OUT	PORT1B		(D/A) ← 00	
		7	05						
		8	7D		MOV	A, L			
		9	D3		OUT	TIMD		(TIM1) ← Δt	
INTEGRATED COMPUTER SYSTEMS	A		14						
	B		21	*	LXI	H, 8320			
	C		20	*					Store address of EXPV.
	D		83	*					
	E		22		SHLD	8398			subroutine for interrupt service
	F		98						
	8	22	0	83					
			1	3E		MVI	A, 01		Enable Timer 0
			2	01					interrupt
			3	D3		OUT	CNT2		
		4	0F						
		5	C3		JMP	8280		Jump to main 1/2	
		6	80						
		7	82						
		8							

FIGURE 4-44a

FUNCTION GENERATOR - INTERRUPT SERVICE

		A	D	O	R	CODE						
CODING SHEET	8	0										
		1										
		2										
		3										
		4										
		5										
		6										
		7										
MICROCOMPUTER TRAINING SYSTEM	822	8	FS			PUSH	PSW				TIMER 0	
		9	ES			PUSH	H				Save registers	
		A	DS			PUSH	D					
		B	CS			PUSH	B					
		C	C3			JMP	8233				Jump past RST6	
		D	33									
		E	82									
		F	00			NOP						
		823	0	E7			RST4					IF RST6 occurs
			1	FB			SI					enter monitor
			2	C9			RET					
		823	3	21			LXI	H, EXIT				Push EXIT
			4	44								address for
			5	82								return from
			6	ES			PUSH	H				function module
			7	2A			LHLD	FUNC				Push function
		8	98								address for	
		9	83								dispatch to	
INTEGRATED COMPUTER SYSTEMS		A	ES			PUSH	H				function module	
			B	21		LXI	H, 8391				(HL) ← address	
			C	91							of 4 or for	
			D	83							function module	
			E	DB			IN	PORT1B			(A) ← present	
			F	05							voltage	
		824	0	C9			RET					Dispatch to
			1	00			NOP					function module
			2	00			NOP					
			3	00			NOP					
			4									
			5									
		6										
		7										
		8										

FIGURE 4-44b

FUNCTION TRIWV

A	D	D	R	CODE	FUNCTION	TRIWV	
8	2	5	0	46	MOV	B, M	(B) ← ΔN
			1	23	INX	H	Address target
			2	BE	CMP	M	Compare voltage
			3	D2	JNC	8260	If voltage ≥ target go to subtract
			4	60			
			5	82			
			6	80	ADD	B	(A) ← N' = N + ΔN
			7	DA	JC	8266	If beyond FF go to reverse
			8	66			
			9	82			
	A			BE	CMP	M	Compare new voltage to target
	B			D2	JNC	8266	If past target go to reverse
	C			66			
	D			82			
	E			C9	RET		Exit if not past
	F			00	NOP		
8	2	6	0	90	SUB	B	(A) ← N' = N - ΔN
			1	DA	JC	8266	If below 00 go to reverse
			2	66			
			3	82			
			4	BE	CMP	M	Compare new voltage to target
			5	DO	RNC		Exit if not past
	8	2	6	7E	MOV	A, M	Reverse
			7	2F	CMA		Complement target voltage
			8	77	MOV	M, A	Return with N = old target
			9	2F	CMA		
	A			C9	RET		
	B						
	C						
	D				ENTER WITH		
	E				(A) = PRESENT VOLTAGE		
	F				((HL)) = INCREMENT, ΔN		
8	0				((HL) + 1) = TARGET VOLTAGE		
	1						
	2				RETURN		
	3				(A) = NEW VOLTAGE		
	4				(HL) ADDRESSING TARGET		
	5				AT OVERFLOW RETURN		
	6				(A) = OLD TARGET		
	7				TARGET COMPLEMENTED		
	8						

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

FUNCTION - MAIN LOOP & DISPATCH

	A	D	D	R	CODE							
CODING SHEET	8	2	8	0	FB	EI					Enable after key	
				1	2A	LHLD	8390				processing	
				2	90						(L) ← Δ ←	
				3	83						(H) ← Δ, N	
				4	CD	CALL	DWORD					
				5	DI							
				6	02							
		8	2	8	7	DB	IN	PORT1B				Read output
				8	05							voltage
				9	CD	CALL	DISPL					Display voltage
MICROCOMPUTER TRAINING SYSTEM				A	EB							
				B	82							
				C	CD	CALL	SCAN				Test keyboard	
				D	57							
				E	02							
				F	D2	JNC	8287				Loop if no key	
		8	2	9	0	87						
				1	82							
				2	CD	CALL	ENTBY				(A) ← command	
				3	36						(L) ← data	
INTEGRATED COMPUTER SYSTEMS				4	03						(D) ← hex key count	
				5	5D	MOV	E, L				(E) ← data	
				6	21	LXI	H, 8280				Push address of	
				7	80						display function	
				8	82						for return from	
				9	ES	PUSH	H				process modules	
				A	CG	ADI	95				Add dispatch tabl.	
				B	95						address -10 to	
				C	6F	MOV	L, A				command key	
				D	6E	MOV	L, M				2 (ST) ← dispatch	
			E	ES	PUSH	H				address		
			F	7B	MOV	A, E				(A) ← data byte		
	8	2	A	0	21	LXI	H, 8391				Address of	
			1	91							voltage increment	
			2	83								
			3	F3	DI							
			4	C9	RET						Jump to process	
			5								module	
			6									
			7									

FIGURE 4-44e

FUNCTION - KEY PROCESSING

		A	D	D	R	CODE												
CODING SHEET	8	0																
		1																
		2																
		3																
		4																
MICROCOMPUTER TRAINING SYSTEM	82A	5	D	1	*	M	E	M									Set exponential	
		6	C	A	*	R	E	G									Set triangular	
		7	A	4		A	D	D	R									
		8	B	0		S	T	E	P									Set ΔN
		9	A	D		R	U	N										Set Δt
		A	C	0		N	E	X	T									Set or reverse target
		B	B	2		B	R	K										Set fixed voltage
		C	B	4		C	L	R										Set voltage
		82A	D	D	3		O	U	T		T	I	M	O				RUN Set Δt
			E		1	4												Load Timer 0
INTEGRATED COMPUTER SYSTEMS		F	2	B		D	C	X		H							Address Δt	
	82B	0	7	7		M	O	V		M	, A						STEP Set ΔN	
		1	C	9		R	E	T										
		82B	2	3	6		M	V	I		M	, 00						BRK set fixed N
			3	0	0													Clear ΔN
		82B	4	D	3		O	U	T		P	O	R	T	I	B		CLR set N
			5	0	5													
			6	2	3	*	I	N	X		H							Address low byte
			7	2	3	*	I	N	X		H							or N at 8394
			8	2	3	*	I	N	X		H							
		9	3	6	*	M	V	I		M	, 00							
	A	0	0	*														
	B	C	9	*	R	E	T											
	C																	
	D																	
	E				*	C	H	A	N	G	E	D					F O R E X P O N E N T I A L	
	F																	
INTEGRATED COMPUTER SYSTEMS	8	0																
		1																
		2																
		3																
		4																
		5																
		6																
		7																
	8																	

FIGURE 4-44f

FUNCTION - KEY PROCESSING cont'd 4-148

A D D R		CODE							
CODING SHEET	8 2 C 0	2 3		INX	H			NEXT Set Target	
		1 4		INR	D			Test for data entered	
		2 5		DCR	D				
		C 2		JNZ	8 2 C 8			If data entered	
		C 8						jump to store as	
		8 2						target voltage	
		7 E		MOV	A, M			Else complement	
		2 F		CMA				old target	
		8 2 C 8	7 7		MOV	M, A		Store target	
			C 9		RET				
MICROCOMPUTER TRAINING SYSTEM	8 2 C A	2 1 *		LXI	H, 8 2 5 0			REG - Set triangular	
		B 5 0						(HL) ← Address TRIWV	
		C 8 2							
		8 2 C D	2 2		SHLD	8 3 9 8			Store function
			E 9 8						address for
			F 8 3						interrupt service
		8 2 D 0	C 9		RET				
		8 2 D 1	2 3 *		INX	H			MEM - Set exponential
			2 3		INX	H			Store Vs at 8393
			7 7		MOV	M, A			
INTEGRATED COMPUTER SYSTEMS		2 1		LXI	H, 8 3 2 0			(HL) ← Address EXPV	
		5 2 0							
		6 8 3							
		7 C 3		JMP	8 2 C D				
		8 C D							
		9 8 2							
		A							
		B							
		C		*	REG AND MEM PROCESSING				
		D			ADDED FOR EXPV				
	E								
	F								
	8 0								
	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								

FIGURE 4-44g

DISPLAY VOLTAGE SUBROUTINE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE							
8				0							
				1							
				2							
				3							
				4							
				5							
				6							
				7							
				8							
				9							
				A							
	82E	B	CD	CALL	DBYTE	Display	voltage				
		C	95								
		D	02								
		E	79	MOV	A,C	(A) ←	voltage				
		F	21	LXI	H,00FF	To shift	ones				
8	2F	0	FF			into	(H)				
		1	00			Divide	by 29 to				
8	2F	2	D6	SUI	1D	divide	voltage range				
		3	1D			into 9	segments				
		4	DA	JC	82FB	When	remainder				
		5	FB			less	than 0				
		6	82			go to	display				
		7	29	DAD	H	Shift	in ones				
		8	C3	JMP	82F2	Loop					
		9	F2								
		A	82								
	82F	B	7C	MOV	A,H	Display	result				
		C	D3	OUT	PORT1A	in	LED's				
		D	04								
		E	C9	RET							
		F	00	NOP							
8				0							
				1							
				2							
				3							
				4							
				5							
				6							
				7							
				8							

FIGURE 4-44h

FUNCTION SUBROUTINE EXPV

		A	D	D	R	CODE					
CODING SHEET	8	3	2	0		57	MOV	D, A			(D) ← N (high)
				1		46	MOV	B, M			(B) ← Δt/RC
				2		23	INX	H			Address switch
				3		7E	MOV	A, M			} Set CY if ≥ 80 } to increase voltage
				4		17	RAL				
				5		3E	MVI	A, 00			
				6		00					
				7		23	INX	H			Address Vs
				8		4E	MOV	C, M			(C) ← Vs
				9		DA	JC	832D			Jump if voltage
MICROCOMPUTER TRAINING SYSTEM		A				2D					increasing
			B			83					If voltage decreasing
			C			4F	MOV	C, A			clear Vs
		8	3	2	D		23	INX	H		Address N (low)
			E				5E	MOV	E, M		(E) ← N (low)
			F				D5	PUSH	D		Save N
		8	3	3	0		E5	PUSH	H		Save address N
				1			EB	XCHG			(HL) ← N
				2			96	SUB	L		} (DE) ← Vs - N
				3			5F	MOV	E, A		
			4			79	MOV	A, C			
			5			9C	SBB	H			
			6			57	MOV	D, A			
			7			0E	MVI	C, 00		(BC) ← Δt/RC	
			8			00				as two bytes	
			9			CD	CALL	BMULT		(HL) ← (HL) + (DE)(BC)	
INTEGRATED COMPUTER SYSTEMS		A				07					N' = N + (Vs - N)Δt/RC
			B			83					
			C				EB	XCHG			(DE) ← N'
			D				E1	POP	H		Address N (low)
			E				73	MOV	M, E		Store N' (low)
			F				C1	POP	B		(BC) ← old N
		8									CONTINUED
				1				ENTER	(A) =		N (high byte)
				2				RETURN	(A) =		N' (high byte)
				3				AT ENTRY			
			4				((HL))	=		Δt/RC	
			5				((HL) + 1)	=			
			6				((HL) + 2)	=		Source voltage	
			7				((HL) + 3)	=		N (low byte)	
			8								

EXPV- SWITCH WHEN N UNCHANGED

		A	D	D	R	CODE													
CODING SHEET	8	3	4	0	2	B		DCX	H									Address switch	
					1	2	B		DCX	H									
					2	7	B		MOV	A	2	E						Compare N' (low)	
					3	B	9		CMP	C									with old N' (low)
					4	7	A		MOV	A	2	D							(A) ← N' (high)
					5	C	0		RNZ										Exit if N' changed
					6	B	8		CMP	B									Compare N' (high)
					7	C	0		RNZ										Exit if N' changed
					8	7	E		MOV	A	2	M							Complement
					9	2	F		CMA										switch variable
		A				7	7		MOV	M	2	A							
		B				7	A		MOV	A	2	D							(A) ← N' (high)
	C				C	9		RET											
MICROCOMPUTER TRAINING SYSTEM	D																		
	E																		
	F																		
	8			0															
				1															
				2															
				3															
				4															
				5															
				6															
				7															
				8															
				9															
		A																	
		B																	
		C																	
INTEGRATED COMPUTER SYSTEMS	D																		
	E																		
	F																		
	8			0															
				1															
				2															
				3															
				4															
				5															
				6															
				7															
				8															

MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 5

ANALOG TO DIGITAL INPUT

5. Analog to Digital Input

Microprocessors in instruments and control systems generally require input of analog signals, which must be converted to digital form for processing. Variables generated by sensors are likely to be any of the following:

Frequency or Pulse Interval (tachometers, flow meters and other motion sensors, and instrumentation electronics used for conversion of other variables)

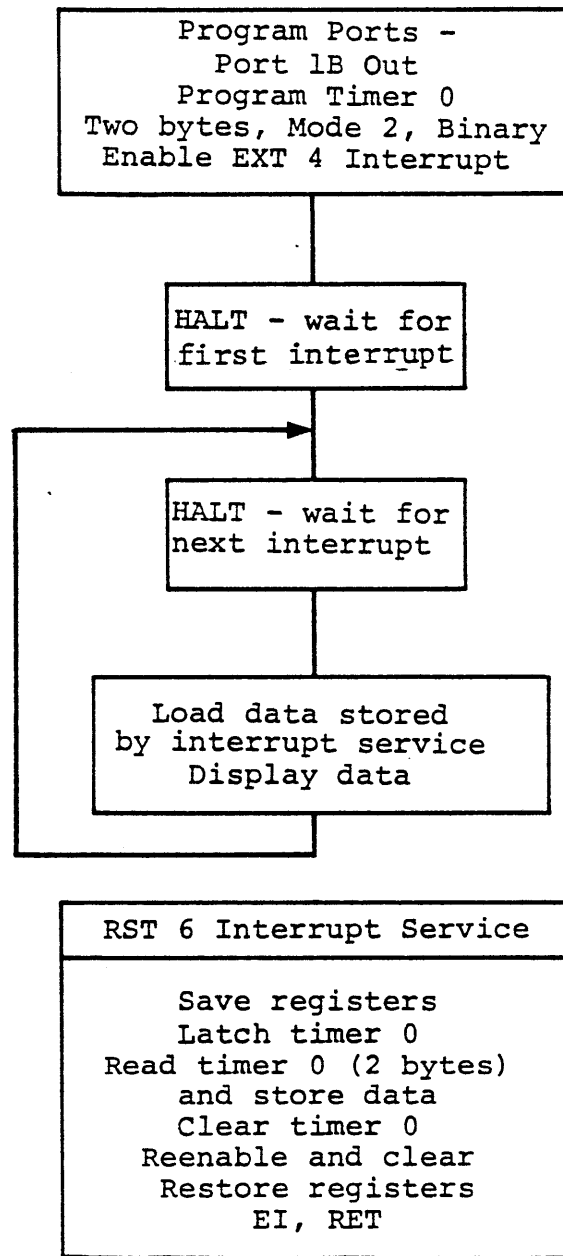
Pulse Width (instrumentation electronics - not common)

Resistance (thermistors, strain gauges, position sensors)

Capacitance or Inductance (position sensors - usually converted to frequency)

Voltage or Current (thermocouples, photovoltaic cells, or conversions of variable resistance.)

In this chapter we will consider digital conversion of pulse interval, frequency voltage, and resistance. In Section 3.6, we measured a pulse width. That section should be reviewed, and especially Section 3.6.3 which discussed reading a timer while it is running.



PULSE INTERVAL MEASUREMENT

Figure 5-1

5.1 Pulse Interval Measurement

An external input of the interface board (either EXT 4 or EXT 5) with its edge triggered latch makes it very easy to measure a pulse interval. For both exercises of this section, connect the input signal to EXT 4. The cassette modem signal is a suitable source for test purposes. It is available at a point marked `FREQ RECORD`.

5.1.1 Measuring a Steady Signal

EXERCISE

When a rising edge occurs at the EXT 4 input, it will set the latch and (provided the EXT 4 interrupt is enabled) cause a RST 6 interrupt. Each time the interrupt occurs, a timer is read and cleared, and the EXT 4 interrupt is again enabled.

The main program displays the data read from the counter by interrupt service, and loops to a Halt instruction. This stops program execution until an interrupt has occurred. Execution of the main program resumes at the next location after the interrupt has been serviced. We ignore the first measurement because it is meaningless. The counter was started when power came on, not at an edge of the signal. Thereafter, we load and display the measured data after each interrupt.

PULSE INTERVAL - MAIN

		A	D	D	R	CODE						
CODING SHEET	8	2	0	0		3E	MVI	A,	80		Program Ports	
						80					Port 1B Out	
						D3	OUT	CNT1				
						07						
						3E	MVI	A,	92			
						92						
						D3	OUT	CNT2				
						0F						
MICROCOMPUTER TRAINING SYSTEM						3E	MVI	A,	30		Program Timer 0	
						30					Two bytes	
		A				D3	OUT	TIMCT			Mode, Binary	
						17						
						3E	MVI	A,	09		Enable EXT4	
						09					interrupt	
						D3	OUT	CNT2				
						0F						
		8	2	1	0		76	HLT				Ignore first inter.
		8	2	1	1		76	HLT				LOOP -
						2A	LHLD	8300			Load measured	
						00					interval	
						83						
						CD	CALL	DWORD			Display interval	
						D1						
						02						
						C3	JMP	8211			Loop	
						11						
						82						
INTEGRATED COMPUTER SYSTEMS												

Figure 5-2a

PULSE INTERVAL - INTERRUPT SERVICE 5-5

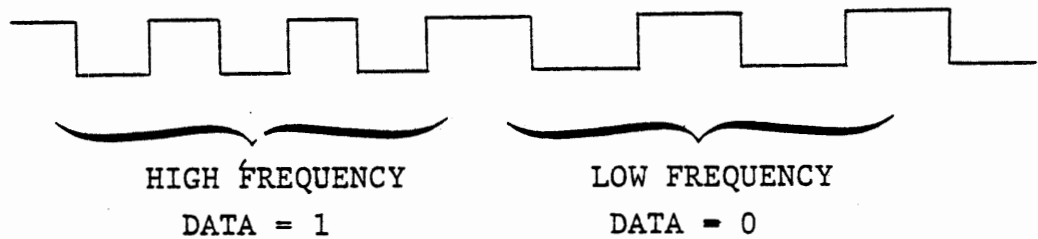
		A	D	D	R	CODE							
CODING SHEET	8	2	3	0	F5	PUSH	PSW						
				1	E5	PUSH	H						
				2	21	LXI	H, 8300					Address data	
				3	00							storage location	
				4	83								
				5	3E	MVI	A, 00					Latch Timer 0	
				6	00								
				7	D3	OUT	TIMCT						
				8	17								
				9	DB	IN	TIMO					Read Timer 0	
MICROCOMPUTER TRAINING SYSTEM		A			14							and store data	
			B			77	MOV	M, A					
			C			23	INX	H					
			D			DB	IN	TIMO					
			E			14							
			F			77	MOV	M, A					
	INTEGRATED COMPUTER SYSTEMS	8	2	4	0	AF	XRA	A					Clear Timer 0
					1	D3	OUT	TIMO					
					2	14							
					3	D3	OUT	TIMO					
				4	14								
				5	3E	MVI	A, 09					Renable and	
				6	09							clear EXT 4	
				7	D3	OUT	CNT2						
				8	0F								
				9	E1	POP	H						
		A			F1	POP	PSW						
		B			FB	EI							
		C			C9	RET							
		D											
		E											
		F											
	8			0									
				1									
				2									
				3									
				4									
				5									
				6									
				7									
				8									

Figure 5-2b

5.1.2. Measuring a Multi-Valued Interval

EXERCISE

When the cassette modem is actually in use its pulse interval is switched between two values, depending on the data being recorded.

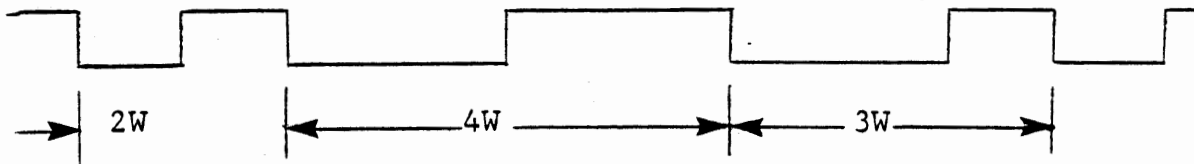


We can compare the measured interval to some threshold value to decide which frequency is present. We will use the monitor's tape recording program SEROT to generate a data train to the cassette modem, and develop an interrupt service routine to measure the intervals, decide which frequency is present, and measure an average interval for one frequency.

The low frequency is half of the high frequency, and the signal is a square wave, so if we define W as the width of a high frequency pulse, the interval for the high frequency is $2W$ and the low frequency interval is $4W$. If the bit time is constant the number of $2W$ intervals for a data bit equal to one should be twice the number of $4W$ intervals for a zero. This ratio is severely distorted however, by the interrupt service routine which interferes with the bit timing loop in SEROT. Therefore the number of cycles of each frequency is meaningless during this test. The measurement of pulse intervals is

valid, however, because the recording frequency is generated by the modem hardware, not by the program.

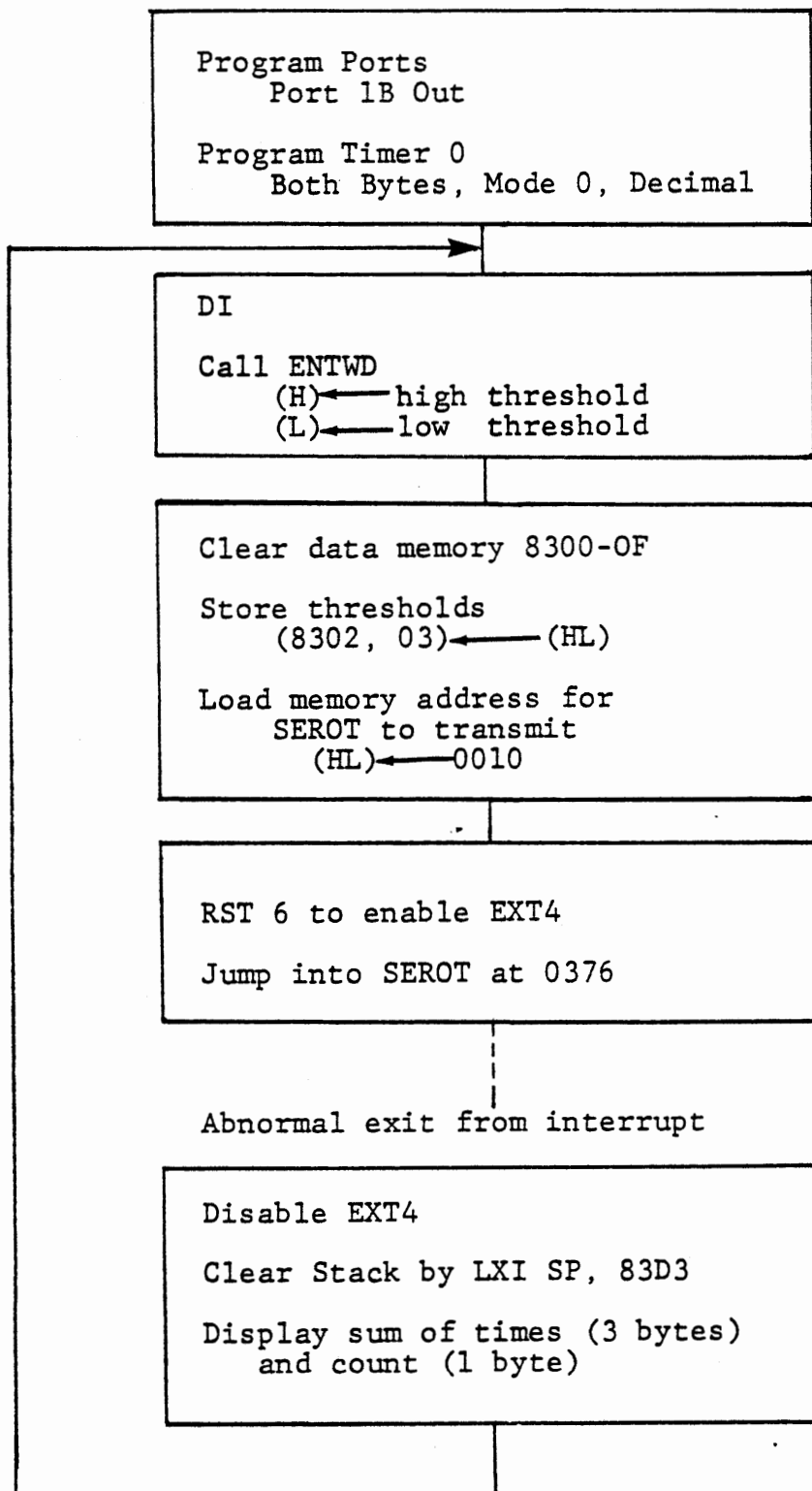
If we use a single threshold, as suggested above, an uncertainty arises in the measurements because an intermediate pulse interval can also occur. The modem changes its interval after its input data from the processor changes, at the moment when its output changes from high to low or from low to high.



The intermediate value $3W$ occurs much less frequently than the other values because it can only occur at a bit boundary, and even there it has only a 25% probability. It will be detected, however, and our program should provide for it.

5.1.2.1 Program Design

The program will accept (through keyboard entry) two different thresholds. The modem output will be sensed as before by the EXT4 interrupt. At each interrupt the preceding time interval will be measured and compared with the thresholds. If it lies between them the interval will be added into a sum, and counted. If the interval is less than the lower threshold or more than the higher, the time will be discarded. All intervals will be counted (separately from the count of those between thresholds).



MULTI-VALUED INTERVAL-MAIN

FIGURE 5-3a

The program will be stopped either when 100 values have been summed or when 65,536 interrupts have occurred. The latter stop will indicate few or no measurements have occurred between the thresholds. If the two thresholds have been selected to include one of the 2W, 3W, or 4W intervals the program will be stopped after 100 occurrences. By varying the thresholds we can measure the average interval for each nominal value. For ease of interpretation the interval measurement and summing will be in decimal.

5.1.2.2 Main Program

The main program is depicted in Figure 5-3a. Timer 0 is used again to measure the interval, but now it is programmed for decimal counting. After the two thresholds have been entered the main program clears the data memory and stores the thresholds. RST6 is used to call the interrupt service, which will start the timer and enable the EXT4 interrupt. (This leads to some invalid measurements which will be discarded by interrupt service). Now (HL) is loaded with a convenient address for the start of data transmission by SEROT. Any address can be used, except that unless quite frequent alternations of ones and zeros are present there will be very few 3W intervals. A starting address of 0010 works nicely.

A jump into SEROT causes the monitor to start transmitting data to the modem. SEROT starts at 0375 with DI; this must be avoided so we jump to 0376.

THIS PAGE INTENTIONALLY LEFT BLANK

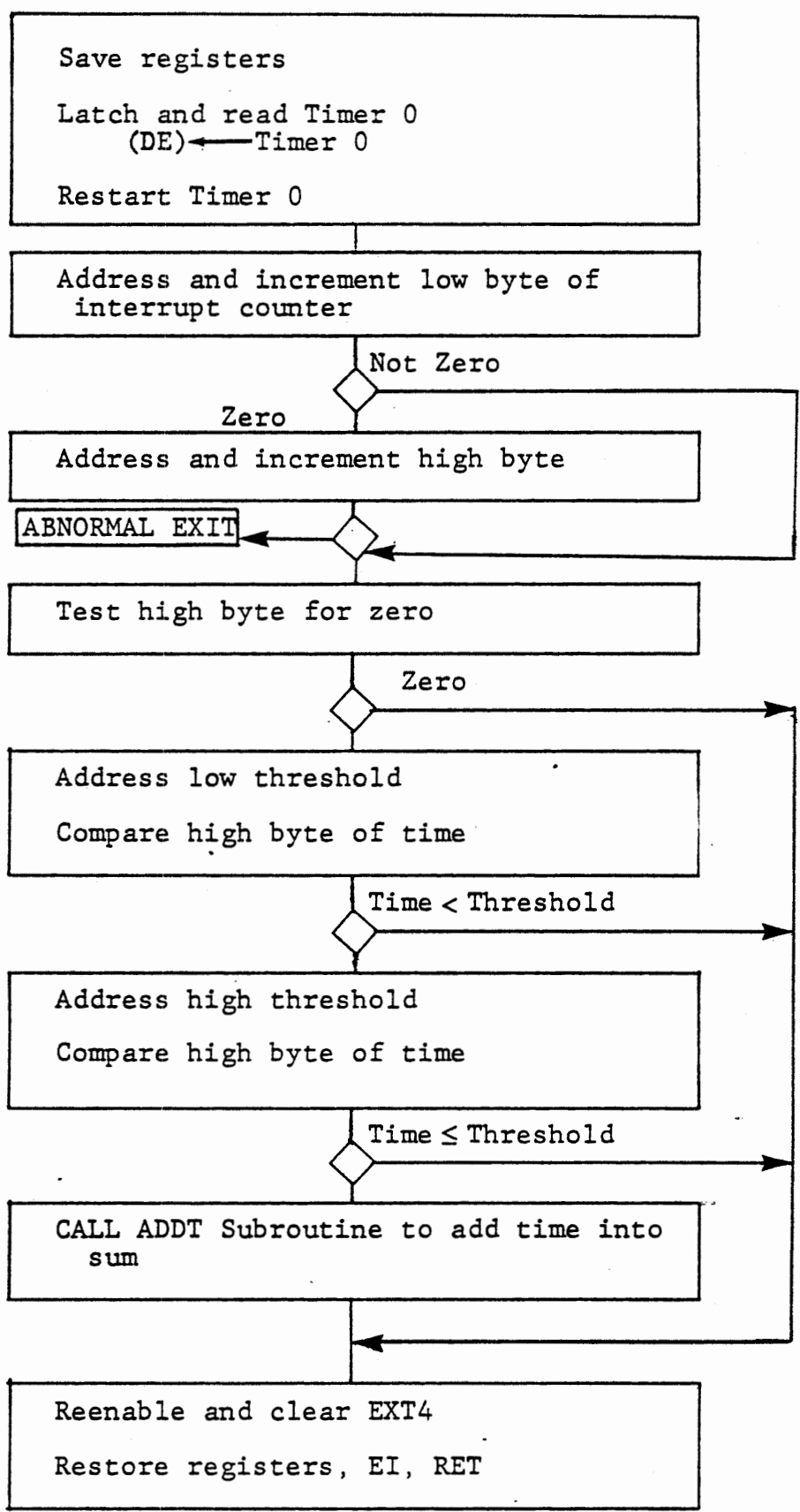
5.1.2.3 Abnormal Exit

When interrupt service for EXT4 reaches a stopping point, after 100 measured intervals or 65,536 total intervals, it makes an abnormal exit. Instead of restoring registers and returning to SEROT, the abnormal exit clears the stack and displays the measured data.

Previously we have sometimes made abnormal exits from subroutines, clearing the stack by popping the return address into a register pair. Here we cannot use that method because SEROT uses several nested subroutines and also uses the stack to save registers, and we do not know how many levels of the stack may be in use. In this program we clear the stack by reinitializing the stack pointer:

```
31      LXI      SP,83D3
      D3
      83
```

This loads the stack pointer with the same address normally loaded by the monitor. Although the stack contents have not been erased, this effectively discards the past history and gives a fresh start.



MULTI-VALUED INTERVAL - INTERRUPT SERVICE
FIGURE 5-3b

5.1.2.4 Interrupt Service

Figure 5-3b shows the interrupt service routine. After saving the registers we read and restart Timer 0. The two byte interrupt count is incremented. At 65536 interrupts the counter reaches zero and the abnormal exit is taken. To avoid recording invalid data caused by initialization both in the main program and in SEROT, we ignore the first 256 measurements, by testing the high byte of the count. The high byte of the timer data is compared to the two thresholds. Since the timer returns the hundreds complement of the interval time, the comparison is made not by CMP but by:

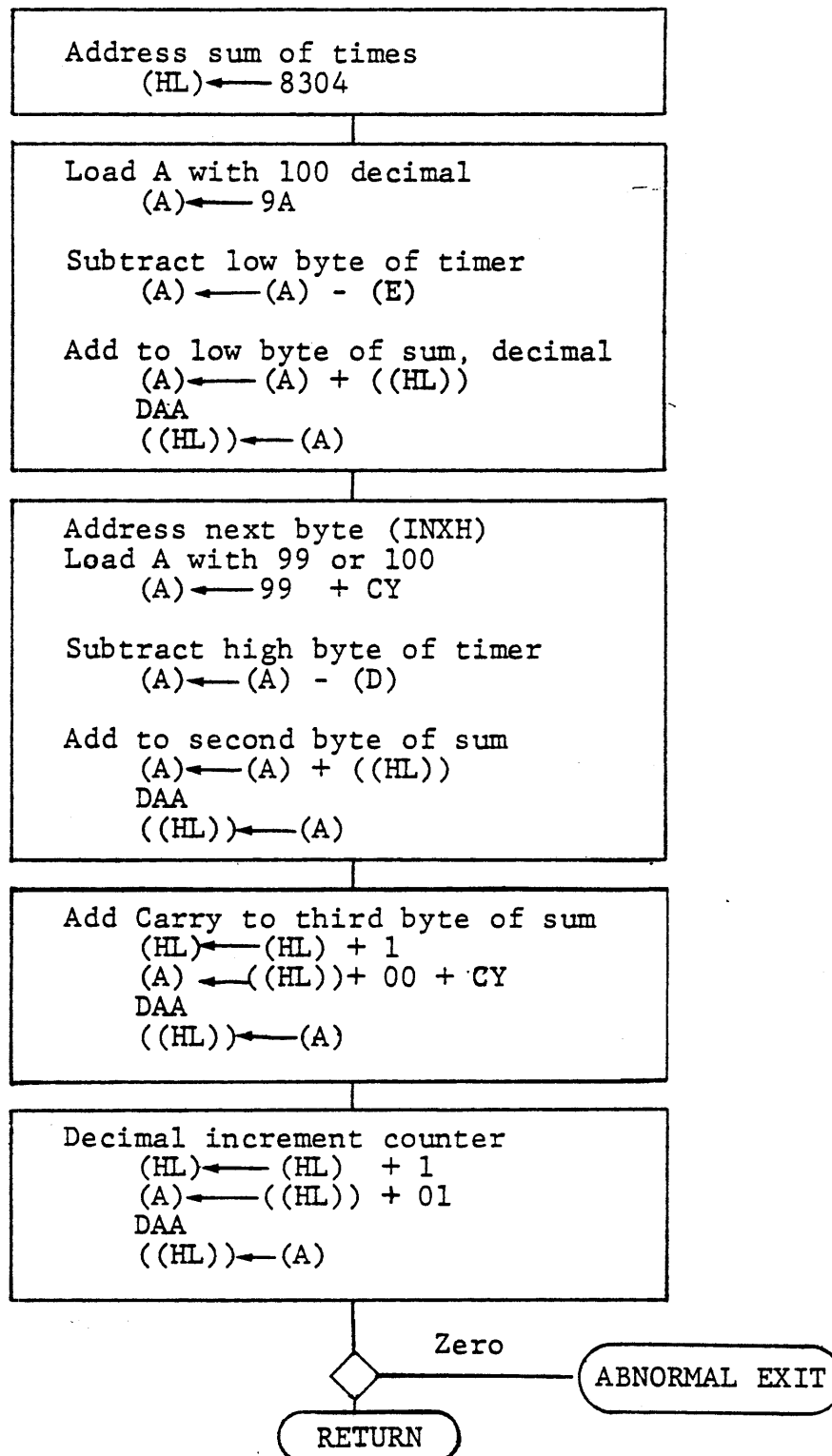
```

MOV     A,D     (A)<---high byte of time
ADD     M       Add threshold
DAA

```

This sets CY if the time is less than the threshold.

If the time lies between the two thresholds a subroutine is called to add the hundreds complement of the timer data into the sum, and to increment the decimal counter.



ADD DECIMAL TIME TO MEMORY

FIGURE 5-3c

5.1.2.5 Decimal Addition Subroutine

Figure 5-3c shows the addition subroutine. Since the timer data represents the hundreds complement of the time interval we must complement it before adding it into the sum. Here we combine the two functions. The Intel 8080 (or the NEC 8080AF) does not allow DAA after subtraction, but only after addition.

```

MVI    A,9A    (A)<---100
SUB    E       Subtract low bytes
ADD    M       Add to low sum
DAA
MOV    M,A

```

Subtracting a decimal value from 9A or 99 cannot generate any carry. After adding the old sum, DAA will make the proper adjustment to a decimal value, generating a carry if the result exceeds 99. For the second byte we load A with 99 and add the carry from the first byte, so it contains either 99 or 9A (= 100 decimal).

```

INX    H       Address second byte
MVI    A,99    Load A with 99 or 100
ACI    00
SUB    D       Subtract timer
ADD    M       Add into old sum
DAA
MOV    M,A

```

If this generates a carry it must be added into the third byte.

Now the decimal counter is incremented and we return to increment the binary count unless the decimal counter reaches 100. Then the abnormal exit is taken. Note that the use of LXI SP permits the abnormal exit from any subroutine level without regard to the stack.

Write your program and test it. Section 5.1.2.6 describes the use of the program. Memory assignments in the solution given in Figure 5-4 are:

8300,01	Binary count
8302	Low threshold
8303	High threshold
8304,05,06	Sum of times
8307	Decimal count

MULTI-VALUED INTERVAL MEASUREMENT 5-17

A D D R		CODE					
CODING SHEET	8 20 0	3E	MVI	A,	80		Program Ports
	1	80					Port 1B Out
	2	D3	OUT	CNT1			
	3	07					
	4	3E	MVI	A,	92		
	5	92					
	6	D3	OUT	CNT2			
	7	0F					
	8	3E	MVI	A,	31		Program Timer 0
	9	31					Both bytes
MICROCOMPUTER TRAINING SYSTEM	A	D3	OUT	TIMCT			Mode 0
	B	17					Decimal
	C	F3	DI				
	D	CD	CALL	ENTWD			Enter thresholds
	E	46					(H) ← high
	F	03					(L) ← low
	8 21 0	AF	XRA	A			Clear data memory
	1	11	LXI	D,	8310		8300 - 830F
	2	10					
	3	83					
MICROCOMPUTER TRAINING SYSTEM	8 21 4	1D	DCR	E			
	5	12	STAX	D			
	6	C2	JNZ	8214			
	7	14					
	8	82					
	9	22	SHLD	8302			Store thresholds
	A	02					
	B	83					
	C	21	LXI	H,	0010		Address for start
	D	10					of transmission
INTEGRATED COMPUTER SYSTEMS	E	00					
	F	00	NOP				Use EI during debug
	8 22 0	F7	RST6				To enable EXT4
	1	C3	JMP	0376			Jump into SEROT
	2	76					after DI
	3	03					
	4						
	5						
	6						
	7						
8							

Figure 5-4a

INTERRUPT SERVICE

	A	D	D	R	CODE							
CODING SHEET	8	2	3	0	FS		PUSH	PSW				
				1	ES		PUSH	H				
				2	DS		PUSH	D				
				3	CS		PUSH	B				
				4	3E		MVI	A, 00			Latch and read	
				5	00						Timer 0	
				6	D3		OUT	TIMCT				
				7	17							
				8	DB		IN	TIMO				
				9	14							
MICROCOMPUTER TRAINING SYSTEM		A			5F		MOV	E, A				
					DB		IN	TIMO				
					14							
					57		MOV	D, A				
					AF		XRA	A			Restart Timer 0	
					D3		OUT	TIMO				
	INTEGRATED COMPUTER SYSTEMS	8	2	4	0	14						
					1	D3		OUT	TIMO			
					2	14						
					3	21		LXI	H, 8300			Address binary
				4	00						counter for	
				5	83						interrupts	
				6	34		INR	M			Increment low byte	
				7	23		INX	H			Address high byte	
				8	C2		JNZ	824F				
				9	4F							
			A	82						At zero increment		
			B	34		INR	M			high byte		
			C	CA		JZ	8290			At 0000 go to		
			D	90						abnormal exit		
			E	82								
	8	2	4	F	B6		ORA	M			Test for high byte	
				0							greater than zero	
				1								
				2								
				3								
				4								
				5								
				6								
				7								
				8								

Figure 5-4b

INTERRUPT SERVICE CONTINUED

A D D R		CODE					
CODING SHEET	8 25 0	CA	JZ	8261			Until high byte
	1	61					> 0 go to exit
	2	82					
	3	23	INX	H			Address low threshold
	4	7A	MOV	A, D			Add to complement
	5	86	ADD	M			of time interval
	6	27	DAA				
	7	DA	JC	8261			Exit if interval
	8	61					less than lower
	9	82					threshold
MICROCOMPUTER TRAINING SYSTEM	A	23	INX	H			Test higher
	B	7A	MOV	A, D			threshold
	C	86	ADD	M			
	D	27	DAA				If time between
	E	DC	CC	8270			thresholds add
	F	70					to sum of times
	8 26 0	82					
	826 1	3E	MVI	A, 09			Normal Exit
	2	09					
	3	D3	OUT	CNT2			
INTEGRATED COMPUTER SYSTEMS	4	0F					
	5	C1	POP	B			
	6	D1	POP	D			
	7	E1	POP	H			
	8	F1	POP	PSW			
	9	FB	EI				
	A	C9	RET				
	B						
	C						
	D						
E							
F							
8	0						
1							
2							
3							
4							
5							
6							
7							
8							

Figure 5-4c

BETWEEN THRESHOLDS

		A	D	D	R	CODE									
CODING SHEET	8	2	7	0	2	1	L	X	I	H	8	3	0	4	Address sum of
		1	0	4											intervals
		2	8	3											
		3	3	E			M	V	I	A	9	A			Add hundreds
		4	9	A											complement of
		5	9	3			S	U	B	E					timer data
		6	8	6			A	D	D	M					to sum
		7	2	7			D	A	A						
		8	7	7			M	O	V	M	A				
		9	2	3			I	N	X	H					
MICROCOMPUTER TRAINING SYSTEM	A	3	E			M	V	I	A	9	9				
	B	9	9												
	C	C	E			A	C	I	0	0					
	D	0	0												
	E	9	2			S	U	B	D						
	F	8	6			A	D	D	M						
	8	2	8	0	2	7	D	A	A						
		1	7	7			M	O	V	M	A				
		2	2	3			I	N	X	H					
		3	7	E			M	O	V	A	M				
INTEGRATED COMPUTER SYSTEMS	4	C	E			A	C	I	0	0					
	5	0	0												
	6	2	7			D	A	A							
	7	7	7			M	O	V	M	A					
	8	2	3			I	N	X	H						Increment count
	9	7	E			M	O	V	A	M					in decimal
	A	C	6			A	D	I	0	1					
	B	0	1												
	C	2	7			D	A	A							
	D	7	7			M	O	V	M	A					
E	C	0			R	N	Z							Return to normal	
F	0	0			N	O	P							exit unless 100	
	8	0													intervals summed
	1														
	2														
	3														
	4														
	5														
	6														
	7														
8															

Figure 5-4d

ABNORMAL EXIT

	A	D	D	R	CODE				
CODING SHEET	8	2	9	0	3E	MVI	A,	08	Disable EXT4
					08				
					D3	OUT	CNT2.		
					0F				
					31	LXI	SP,	83D3	
					D3				Clear Stack
					83				
					21	LXI	H,	8304	Address sum
					04				of times
					83				
	A				11	LXI	D,	83FF	Address right
		B			FF				hand digit
		C			83				
MICROCOMPUTER TRAINING SYSTEM	8	2	9	D	7E	MOV	A,	M	Display 4 bytes
				E	CD	CALL	DBY2		
				F	98				
MICROCOMPUTER TRAINING SYSTEM	8	2	A	0	02				
				1	23	INX	H		
				2	7B	MOV	A,	E	After four bytes
				3	FE	CPI	F8		DBY2 returns
				4	F8				(DE) = 83F7
				5	D2	JNC	829D		Loop until four
				6	9D				bytes displayed
				7	82				
				8	C3	JMP	820C		Jump to get
				9	0C				new thresholds
INTEGRATED COMPUTER SYSTEMS		A			82				
			B						
			C						
			D						
			E						
			F						
		8		0					
				1					
			2						
			3						
			4						
			5						
			6						
			7						
			8						

Figure 5-4e

5.1.2.6 Program Debugging and Operation

The use of RST6 to call the interrupt service routine not only provides an easy way to start the timer and enable the EXT4 interrupt, but also allows you to step through interrupt service for debugging. You can step through with a normal return or force the abnormal exit by preloading the decimal counter with 99. If the thresholds are set at 99 and 00 then interrupt service will call the decimal addition subroutine for any timer data.

For meaningful results the program must be run in AUTO mode. Enter thresholds of 99 and 00 (9900,NEXT). Almost immediately a decimal value for the time is displayed as three bytes, with a count of 00. For instance:

```

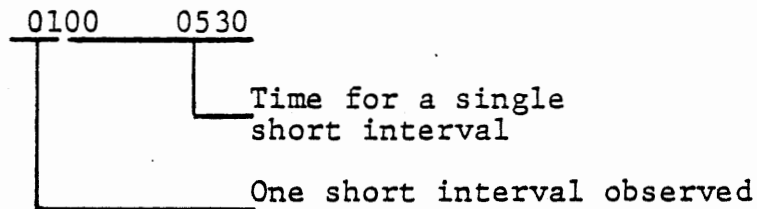
0007 1332
┌───┬───┐
│   │   │
│   │   │
│   │   │
│   │   │
│   │   │
│   │   │
│   │   │
│   │   │
│   │   │
└───┴───┘
      TOTAL TIME FOR 100
      INTERVALS
      DECIMAL COUNT (=100)

```

If equal numbers of wide and narrow intervals were received this would be a central or average value. In fact SEROT generates enough leading high frequency intervals that this measurement includes no 3W or 4W intervals. Prove this by entering thresholds of 09 and 00 (0900,NEXT). A similar average will be obtained.

Try thresholds of 06 and 00. No intervals of fewer than 600 clocks should occur, so the program will run for 65,536 interrupts, and then display 000 000. Possibly one or a few short intervals will occur.

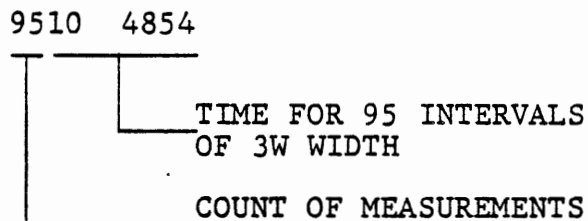
The display might show:



Try thresholds of 07 and 00. Depending on the time constant of the cassette modem oscillator there may be no intervals, a few, or many intervals in this range.

Now exclude the short intervals by entering thresholds of 99 and 09. The 2W intervals will be excluded and the sum of times will include mostly 4W intervals and possibly a few 3W intervals. The average will be close to the 4W interval. The lower threshold can be raised to exclude the 3W intervals. Try 99 and 13, which should include only the 4W intervals.

Finally, set thresholds to include only 3W intervals. Try 13 and 09 (1309,NEXT). There may, or may not, be 100 measurements made before the 65536 interrupts have been counted. In one experiment the result was:



The average time is $104854/95$ or 1104 clock times.

THIS PAGE INTENTIONALLY LEFT BLANK

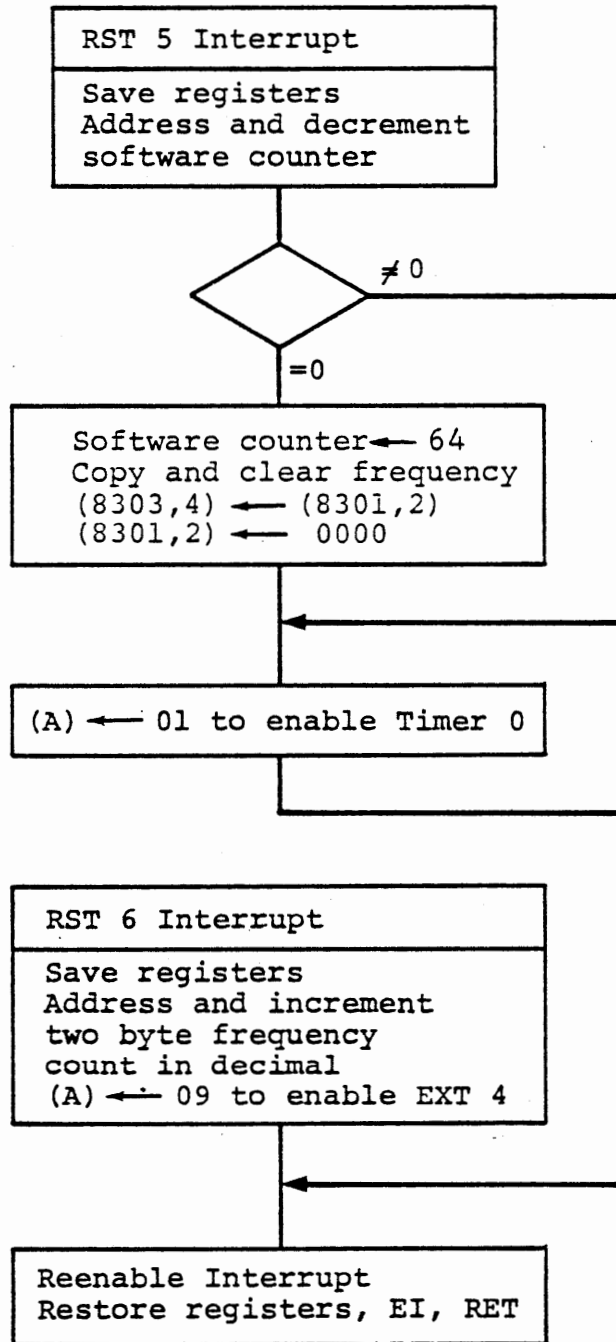
5.1.3 Measuring Received Pulse Intervals

EXERCISE

The program of 5.1.2 can also be used to measure the pulse intervals returned by the cassette recorder. Here we do not need (or want) the SEROT program running, so change the JMP 0376 instruction to a jump to itself. In the solution given in Figure 5-4:

```
821F    JMP    821F
```

Create a tape, or use one you already have. Connect the EXT 4 input to the point on the circuit board labelled `FREQ RECORD`. Connect the cassette, start it in playback mode, and wait until you hear the steady tone from the leader. Now start the program; when it has received 256 zero bits it will display the average pulse intervals. Compare these with those observed for recording. This gives a measure of the speed stability of your recorder.



FREQUENCY MEASUREMENT - INTERRUPT

Figure 5-5

5.2 Frequency Measurement

Clearly a frequency can be obtained from a pulse interval measurement by inverting the measured data. This gives the instantaneous frequency, which may be needed in some instances, especially when the rate of change is important. Often the rate of change is small compared to the frequency, and we can measure frequency by counting pulses over some period of time such as one or ten seconds. This method is used in the two following exercises.

5.2.1 Logic Level Frequency Measurements

Exercise

Measure the frequency of a logic level signal. Use EXT 4 (as in the preceding exercise) to detect the rising edge of the signal, and count the occurrences (in decimal). Use timer 0 with a software counter to measure one second intervals. The Timer 0 interrupt decrements a software counter, which starts at 64 to count 1000 intervals of 10 milliseconds each. At zero, the counter is reloaded. The frequency count is copied to another pair of memory locations and the counter locations are cleared.

The main program does only initialization and display. Program port 2 and timer 0. Load timer 0 with 5000 for a 10 millisecond interrupt interval and enable EXT 4 and Timer 0 interrupts. Then repetitively load the copy of the frequency count and display it.

FREQUENCY MEASUREMENT

	A	D	D	R	CODE						
CODING SHEET	8	2	0	0	3E	MVI	A,	92			Program port 2
			0	1	92						A in Bin Cout
			0	2	D3	OUT	CNT	2			
			0	3	0F						
			0	4	3E	MVI	A,	24			Program timer 0
			0	5	24						high byte, mode 2
			0	6	D3	OUT	TIMCT				
			0	7	17						
			0	8	3E	MVI	A,	50			Load timer 0
			0	9	50						for 10 millisecond
			0	A	D3	OUT	TIM0				interrupt
	MICROCOMPUTER TRAINING SYSTEM			0	B	14					
			0	C	3E	MVI	A,	11			Enable timer 0
			0	D	11						and EXT4
			0	E	D3	OUT	PORT2C				
			0	F	0E						
		8	2	1	0	2A	LHLD	8303			
			1	1	03						
			1	2	83						
			1	3	CD	CALL	DWORD				
			1	4	D1						
			1	5	02						
			1	6	C3	JMP	8210				
INTEGRATED COMPUTER SYSTEMS			1	7	10						
			1	8	82						
			1	9							
			1	A							
			1	B							
			1	C							
			1	D							
			1	E							
			1	F							
		8	2	2	0						
			2	1							
			2	2							
		2	3								
		2	4								
		2	5								
		2	6								
		2	7								
		2	8								

Figure 5-6a

FREQUENCY MEASUREMENT - INTERRUPTS

5-29

	A	D	D	R	CODE						
CODING SHEET	8	0									
		1									
		2									
		3									
		4									
		5									
		6									
		7									
MICROCOMPUTER TRAINING SYSTEM	822	8	F5		PUSH PSW					RSTS - Timer 0	
		9	E5		PUSH H						
		A	21		LXI H, 8300						
		B	00								
		C	83								
		D	C3		JMP 8250						
		E	50								
		F	82								
MICROCOMPUTER SYSTEMS	823	0	F5		PUSH PSW					RST 6 - EXT 4	
		1	E5		PUSH H						
		2	21		LXI H, 8301					Address frequency	
		3	01							count	
		4	83								
		5	37		STC					To add 1	
	823	6	7E		MOV A, M					(A) ← count	
		7	CE		ACI 00					Add carry into	
INTEGRATED COMPUTER SYSTEMS		8	00							count	
		9	27		DAA					Decimal	
		A	77		MOV M, A					Store count	
		B	23		INX H					Next address	
		C	DA		JC 8236					If carry, add	
		D	36							into high byte	
		E	82								
		F	3E		MVI A, 09					To enable EXT 4	
	824	0	09								
	824	1	D3		OUT CNT2				EXIT		
		2	0F								
		3	E1		POP H						
		4	F1		POP PSW						
		5	FB		EI						
		6	C9		RET						
		7									
		8									

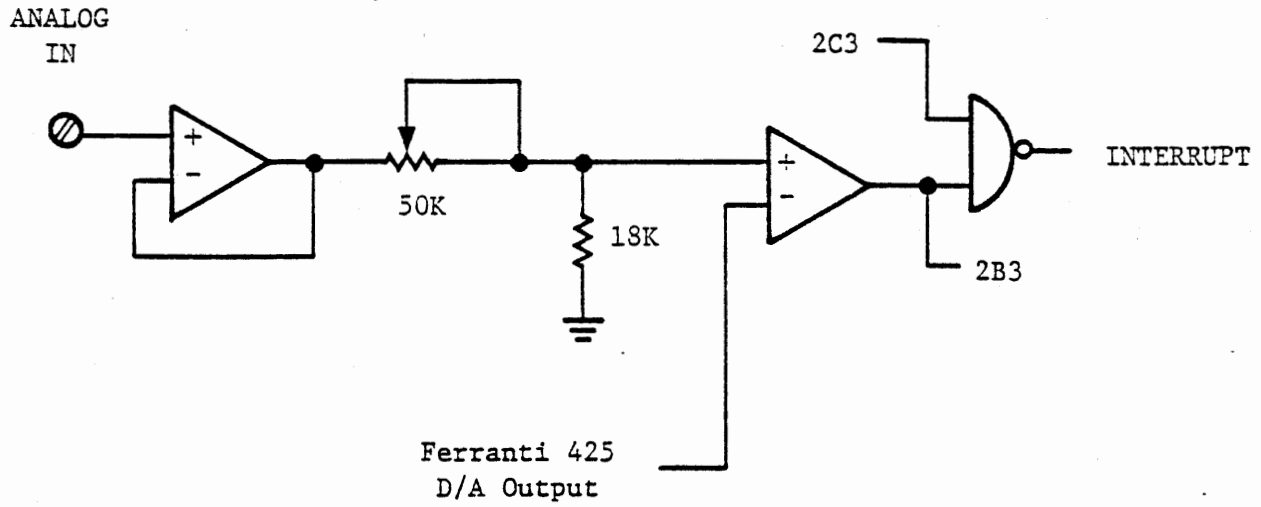
Figure 5-6b

5.2.2 AC Input Signal

Exercise

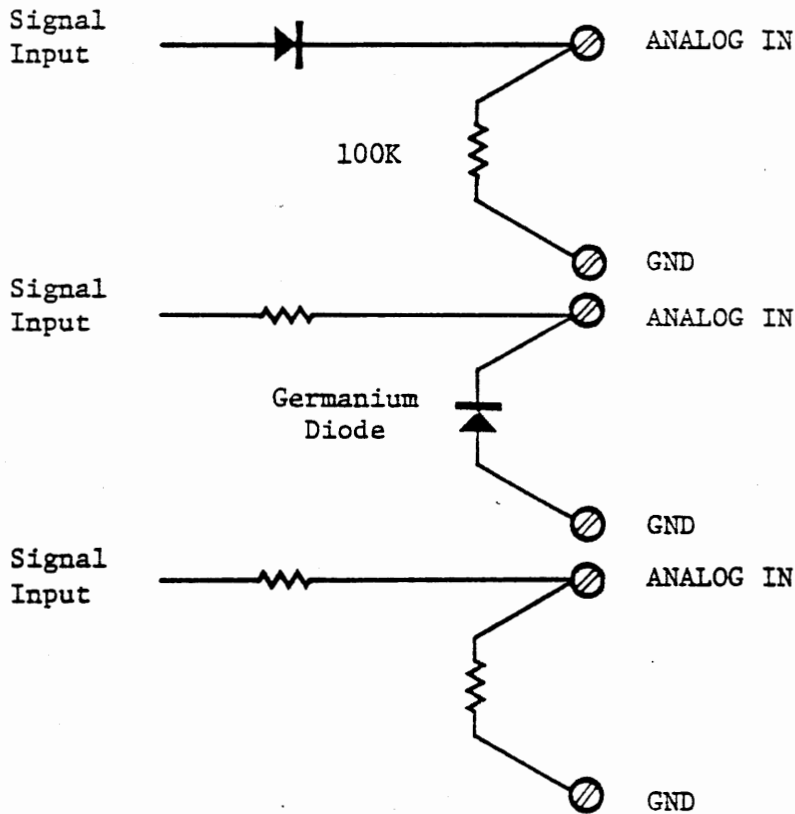
In the preceding section, we measured the frequency of a logic level signal. Often the variable input may be an ac signal, without sharply defined edges. For accurate results, the input signal must be squared. A sinusoidal input may not be detected at the same point in its cycle every time. Squaring can be accomplished with an integrated circuit comparator or an op-amp connected as a comparator. The interface board includes a comparator in the analog input circuit which can be used in this way. Figure 5-7a shows the circuit.

CAUTION: The input to the op-amp must not go more negative than -0.3 volts. If the signal is alternating above and below ground, a protection circuit must be provided as indicated in Figure 5-7b or else the signal must be attenuated to swing within ± 0.25 volts.



COMPARATOR CIRCUIT

Figure 5-7a



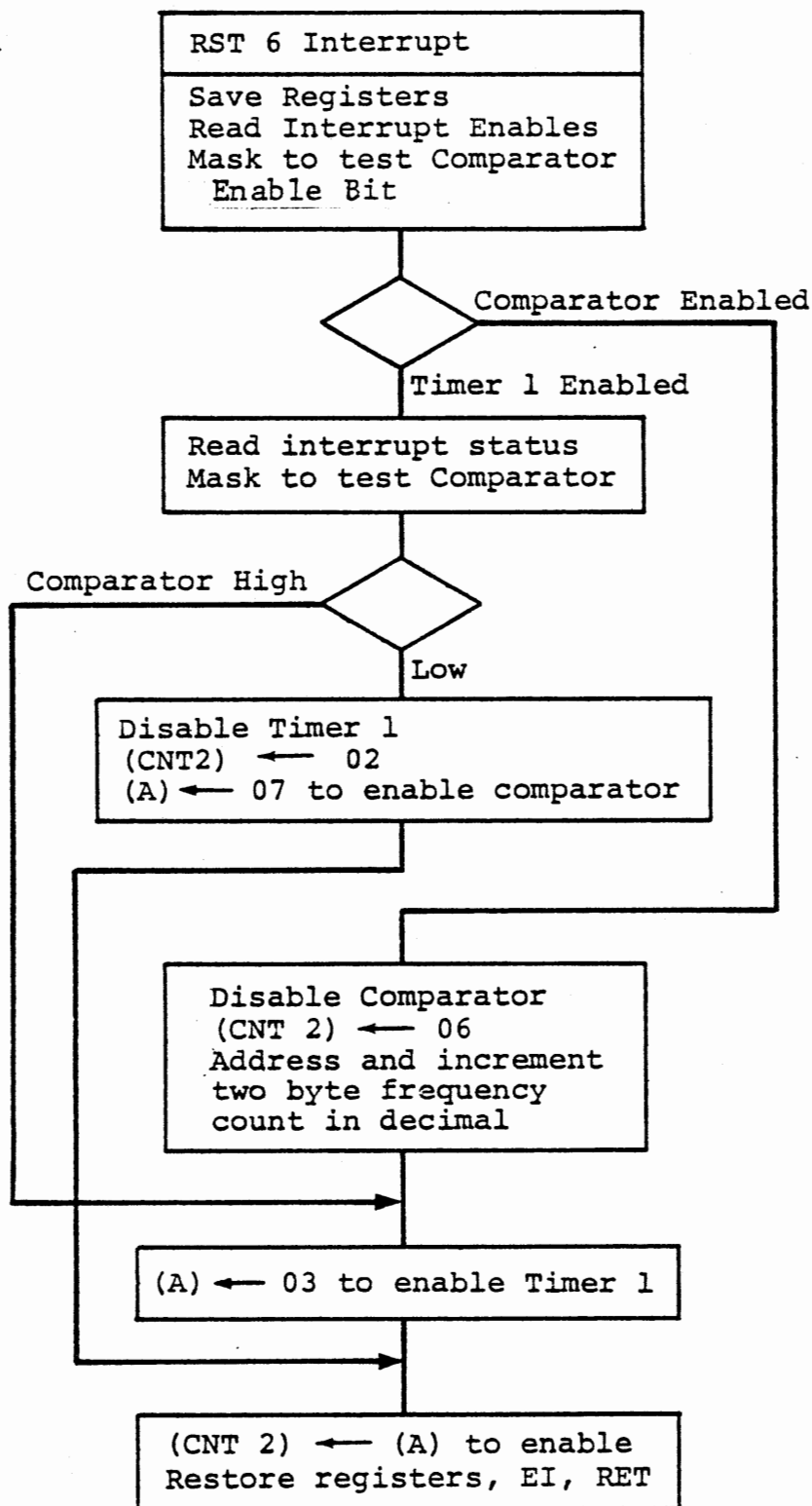
PROTECTION CIRCUITS FOR AC SIGNALS

Figure 5-7b

The input signal to ANALOG IN is amplified (with unity voltage gain) by the first op-amp, attenuated if necessary by the pot, and compared with the output signal from the D/A converter. A threshold signal is provided by the converter. This can be set very close to 0 volts, or to some more positive value. When the input signal is greater than the threshold, the output of the second op-amp is a low logic level. When the input signal is less than the threshold, the logic signal goes high and can generate an interrupt or be sensed at port 2B3.

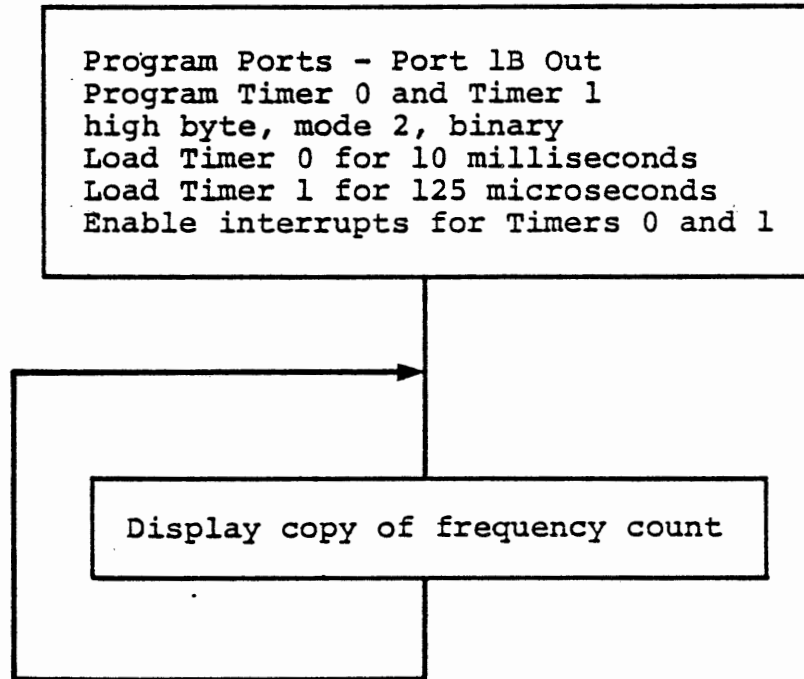
The cassette modem output provides a suitable signal to test this program. It has an amplitude of about ± 0.3 volts with a very high source impedance. Connect a 100K resistor from analog input to ground to ensure that the signal stays within the safe range for the op-amp. Set the ANALOG IN pot to the far right, and connect the CASSETTE AUX output to ANALOG IN.

With this arrangement, a RST 6 interrupt will occur whenever the external signal goes below the threshold. Since there is no latch for this interrupt, the program must monitor port 2B3, and not enable the interrupt again until this signal has become low. In the program of Figure 5-8, timer 1 is used to interrupt the main program often enough to detect when the comparator output goes low and then enable the A/D comparator interrupt. Timer 0 again counts time. The frequency is counted in response to comparator interrupts instead of EXT 4 interrupts.



SINUSOIDAL MEASUREMENT - RST 6 INTERRUPT

Figure 5-8a.



AC SIGNAL FREQUENCY - MAIN

Figure 5-8b

THIS PAGE INTENTIONALLY LEFT BLANK

Interrupt service for this program (Figure 5-8a) introduces a primitive interrupt manager. The occurrence of RST 6 does not by itself tell the program which interrupt source must be serviced. We read the interrupt enable byte (port 2C) and test whether the comparator or timer 1 was enabled to create the interrupt. (We know that only one has been enabled.) If the comparator interrupt was enabled, we know that the comparator caused the interrupt and now must be disabled and timer 1 enabled, and the frequency count should be incremented. If timer 1 was enabled, we read the interrupt status byte (port 2B) to decide whether it is now time to enable the comparator (and disable timer 1) or whether timer 1 should be reenabled and the comparator remain disabled.

Clearly, the function of testing the comparator signal could be relegated to the main program, which has very little to do. We used the two RST 6 interrupts in order to demonstrate one means of distinguishing the source.

AC SIGNAL FREQUENCY MEASUREMENT

5-38

A		D		D		R		CODE								
CODING SHEET	8	2	0	0	3	E		M	V	I	A,	80	Program Ports			
			0	1	8	0							Port 1B Out			
			0	2	D	3		O	U	T	C	N	T	1		
			0	3	0	7										
			0	4	3	E		M	V	I	A,	92				
			0	5	9	2										
			0	6	D	3		O	U	T	C	N	T	2		
			0	7	0	F										
			0	8	3	E		M	V	I	A,	24	Program Timers			
			0	9	2	4							High byte			
			0	A	D	3		O	U	T	T	I	M	C	T	
			0	B	1	7								Mode 2		
			0	C	3	E		M	V	I	A,	64		Binary		
			0	D	6	4										
			0	E	D	3		O	U	T	T	I	M	C	T	
		0	F	1	7											
MICROCOMPUTER TRAINING SYSTEM	8	2	1	0	3	E		M	V	I	A,	50	10 millisecond			
		1	1	5	0								interval			
		1	2	D	3		O	U	T	T	I	M	0	to timer 0		
		1	3	1	4											
		1	4	3	E		M	V	I	A,	01	125 microsecond				
		1	5	0	1								interval			
		1	6	D	3		O	U	T	T	I	M	1	to timer 1		
		1	7	1	5											
		1	8	3	E		M	V	I	A,	03	Enable timer 0				
		1	9	0	3									and timer 1		
		1	A	D	3		O	U	T	P	O	R	T	2	C	
		1	B	0	E											
	INTEGRATED COMPUTER SYSTEMS	82	1	C	2	A		L	H	L	D	8	3	0	3	Loop - Load
			1	D	0	3										copy of frequency
			1	E	8	3										
		1	F	C	D		C	A	L	L	D	W	O	R	D	Display
8		2	2	0	D	1										
		2	1	0	2											
		2	2	C	3		J	M	P	8	2	1	C			
		2	3	1	C											
		2	4	8	2											
		2	5													
	2	6														
	2	7														
	2	8														

Figure 5-9a

AC FREQUENCY MEASUREMENT - INTERRUPTS

		A	D	D	R	CODE					
CODING SHEET	8	0									
		1				*	S	A	M	E	A
		2					E	X	C	E	P
		3					M	A	R	K	E
		4									
		5									
		6									
		7									
MICROCOMPUTER TRAINING SYSTEM	822	8	F	5			P	U	S	H	P
		9	E	5			P	U	S	H	H
		A	2	1			L	X	I	H,	8
		B	0	0							
		C	8	3							
		D	C	3			J	M	P	8	2
		E	5	0							
		F	8	2							
MICROCOMPUTER TRAINING SYSTEM	823	0	F	5			P	U	S	H	P
		1	E	5			P	U	S	H	H
		2	C	3	*		J	M	P	8	2
		3	7	0	*						
		4	8	2	*						
	823	5	3	7			S	T	C		
	823	6	7	E			M	O	V	A,	M
		7	C	E			A	C	I	0	0
INTEGRATED COMPUTER SYSTEMS		8	0	0							
		9	2	7			D	A	A		
		A	7	7			M	O	V	M,	A
		B	2	3			I	N	X	H	
		C	D	A			J	C	8	2	3
		D	3	6							
		E	8	2							
	823	F	3	E			M	V	I	A,	0
824	0	0	3	*							
824	1	D	3			O	U	T	C	N	
	2	0	F								
	3	E	1			P	O	P	H		
	4	F	1			P	O	P	P	S	
	5	F	B			E	I				
	6	C	9			R	E	T			
	7										
	8										

Figure 5-9b

FREQUENCY - TIMER 0 INTERRUPT cont'd 5-40

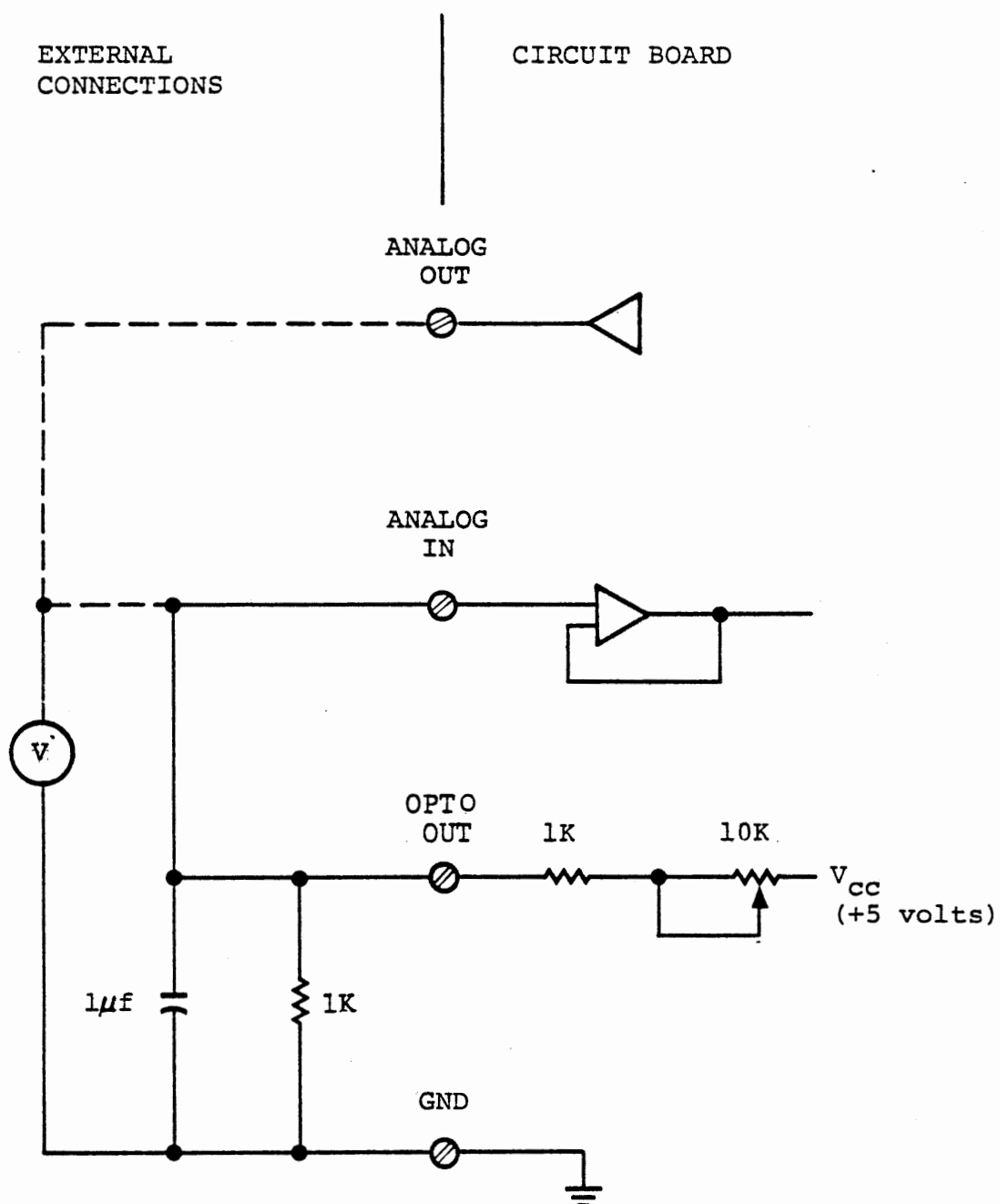
A D D R		CODE					
CODING SHEET	8 25	0 35	DCR	M			Decrement counter
		1 3E	MVI	A, 01			To reenable Timer 0
		2 01					
		3 C2	JNZ	8241			Exit unless
		4 41					counter reaches 00
		5 82					
		6 D5	PUSH	D			
		7 36	MVI	M, 64			Reload counter
		8 64					for 100 ₁₀ intervals
		9 23	INX	H			
MICROCOMPUTER TRAINING SYSTEM	A 5E		MOV	E, M			Copy and clear
	B 36		MVI	M, 00			count of EXT4
	C 00						interrupts
	D 23		INX	H			
	E 56		MOV	D, M			
	F 36		MVI	M, 00			
	8 26	0 00					
		1 23		INX	H		
		2 73		MOV	M, E		Store count of EXT4
		3 23		INX	H		interrupts for
	4 72		MOV	M, D		display as frequency	
	5 D1		POP	D			
	6 C3		JMP	8241		Exit	
	7 41						
	8 82						
	9						
INTEGRATED COMPUTER SYSTEMS	A						
	B		SAME AS FIGURE 5-6 C				
	C						
	D						
	E						
	F						
	8	0					
		1					
	2						
	3						
	4						
	5						
	6						
	7						
	8						

Figure 5-9c

INTERRUPT SERVICE PATCH FOR AC SIGNAL 5-41

		A	D	D	R	CODE					
CODING SHEET	8	27	0	DB		IN	PORT2C				Read interrupt
			1	0E							enables
			2	E6	ANI	08					Test for comparator
			3	08							interrupt enabled
			4	CA	JZ	8281					Jump if not
			5	81							
			6	82							COMPARATOR
			7	3E	MVI	A, 06					Disable comparator
			8	06							interrupt
			9	D3	OUT	CNT2					
MICROCOMPUTER TRAINING SYSTEM		A		0F							
		B		21	LXI	H, 8301					Address frequency
			C		01						count
			D		83						
			E		C3	JMP	8235				Jump to increment
			F		35						frequency count
		8	28	0	82						
			82	81	DB	IN	PORT2B				TIMER 1
				2	0D						Read interrupt status
				3	E6	ANI	08				Test comparator
INTEGRATED COMPUTER SYSTEMS			4	08							
			5	C2	JNZ	823F					If still high go
			6	3F							to reenable
			7	82							timer 1
			8	3E	MVI	A, 02					If low disable
			9	02							timer 1
			A		D3	OUT	CNT2				
			B		0F						
			C		3E	MVI	A, 07				and enable
			D		07						comparator
		E		C3	JMP	8241				interrupt	
		F		41							
	8	29	0	82							
			1								
			2								
			3								
			4								
			5								
			6								
			7								
			8								

Figure 5-9d



CONNECTIONS FOR VOLTMETER EXPERIMENTS

Figure 5-10

5.3 A/D Input - Voltage

Conversion of a voltage input to a digital value is generally performed by comparing the input signal with a voltage generated by digital to analog conversion. The result of the comparison is used to adjust the digital value until the two voltages are alike. A/D converters differ in the adjustment procedure, three principal methods being repetitive ramp, tracking, and successive approximation. We will experiment with each of these.

The comparison between the D/A output and the analog input is the primary purpose of the comparator circuit and gating that were introduced in Section 5.2.2. Refer again to Figure 5-7a, which shows the circuit. For direct measurement of a voltage that is within the 0 to +2.55 volt range of the D/A converter, the ANALOG IN pot can be set for no attenuation of the input signal. For a signal between 2.5 and 5.0 volts, the pot can be set to attenuate the signal by a known amount. Signals greater than 5.0 volts must be attenuated externally, because the op-amp cannot handle a signal greater than its supply voltage. (It will not be damaged by any signal up to +30 volts, but remember that signals lower than -0.3 volts will damage the op-amp.) Since our OPTO OUT will be less than 2.5 volts, we can set the ANALOG IN pot for no attenuation (rotate fully to the left).

A variable DC voltage is needed for these experiments. The OPTO OUT of the interface board is connected through a pot to 5 volts (see Figure 5-10). With an external 1K resistor to ground, a voltage between 0.4 and 2.4 volts can be obtained. A capacitor from the output

to ground is needed to remove noise from the signal. The signal is to be connected to ANALOG IN, and your voltmeter will be connected to either ANALOG IN or ANALOG OUT.

Note: If you are familiar with A/D conversion, you may want to skip the exercises of this section and proceed to Section 5.4, where the use of the automatic A/D input feature is described.

5.3.1 Output, Input and Display Subroutine

EXERCISE

All of the voltmeter programs involve changing the digital value repetitively, comparing its analog conversion with the input signal, and making a decision on that comparison. This operation involves the following steps (with the digital value kept in register L):

MOV A,L (A) <--- digital value

OUT PORT 1B To D/A converter

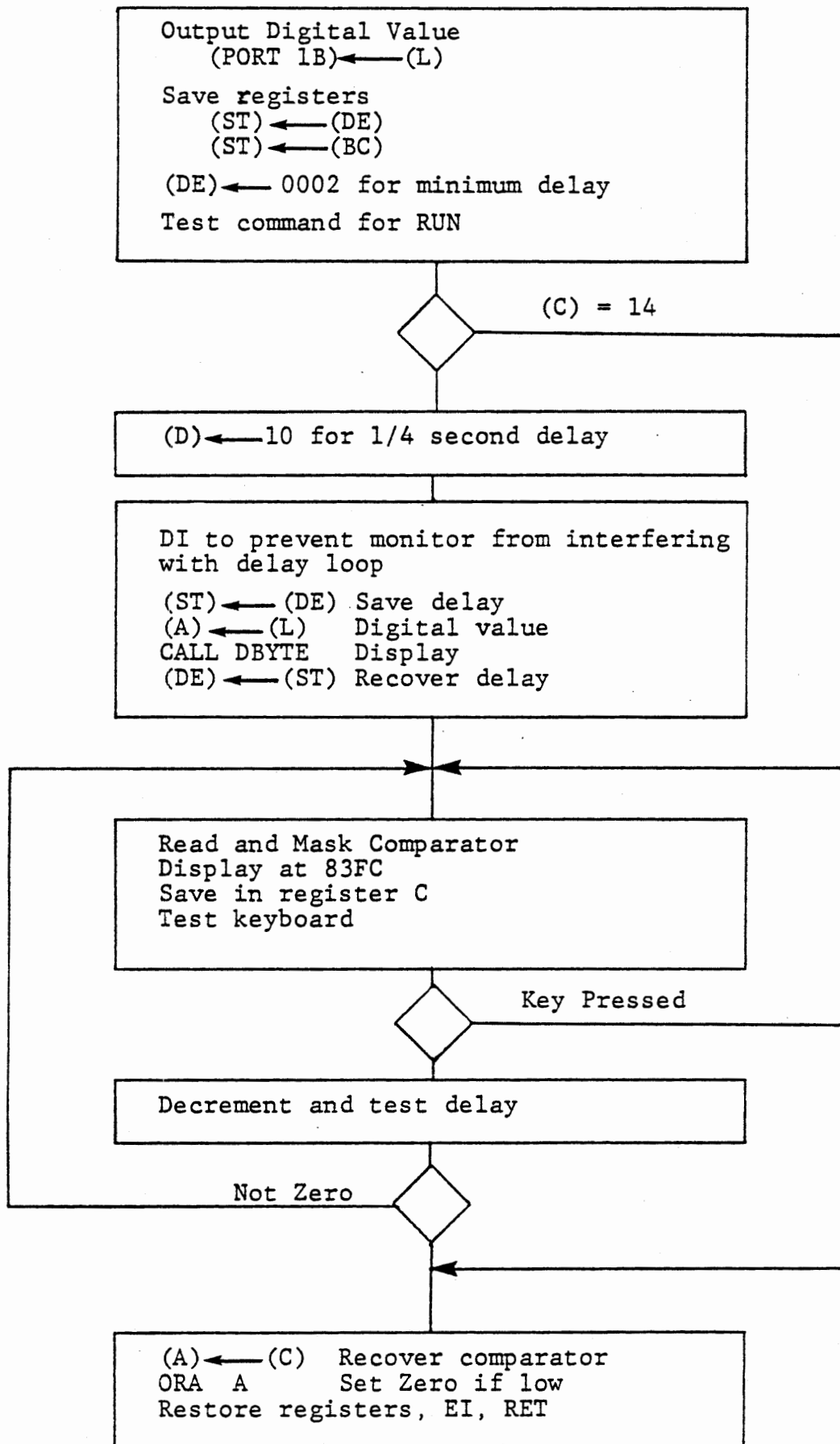
(about 40 micro-seconds delay is required between OUT and IN)

IN PORT 2B Read interrupt status

ANI 08 Mask for comparator

Both the digital to analog converter and the comparator require some settling time before the comparison of D/A output voltage to input voltage is valid. This delay should be at least 40 microseconds. Usually some other function can usefully be accomplished, but otherwise a delay loop can be used.

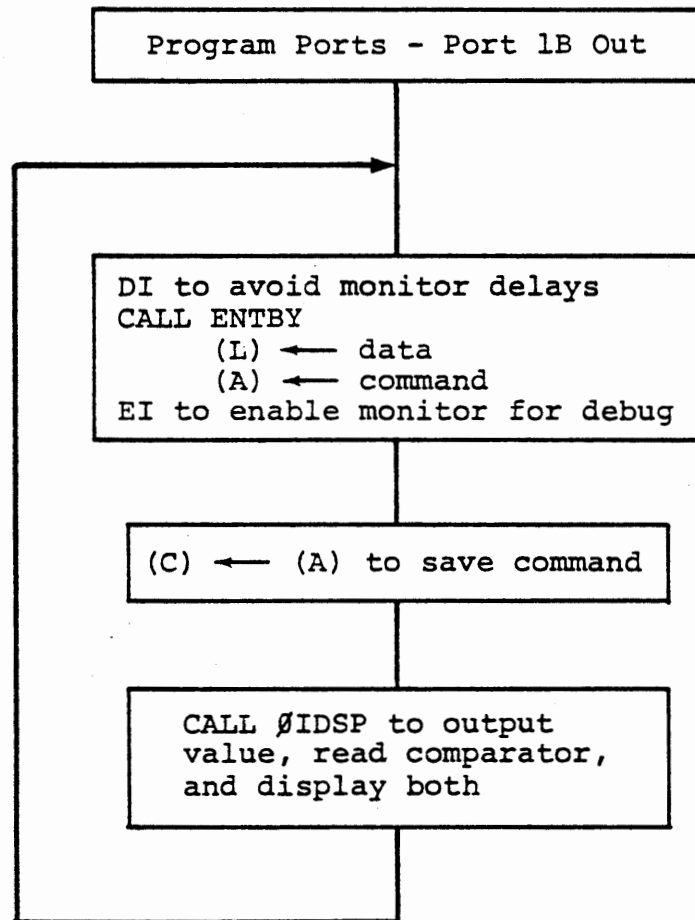
These steps will usually be followed by jump if zero, or jump if not zero for the decision.



OUTPUT, INPUT AND DISPLAY SUBROUTINE
FIGURE 5-11

For convenience in debugging several voltmeter programs, we will create a subroutine that will perform output and input, and also display the digital value and the result of the comparison for some fixed length of time before allowing the main program to proceed with its operations. It will save registers so that it will have exactly the effect of the above process when it returns. In addition, it will make two tests to defeat the delay, as shown in Figure 5-11. If a key input command previously entered and saved in register C is RUN (=14) a minimum delay is set and the display is bypassed. Otherwise a 1/4 second delay is entered and the digital value is displayed. Within the delay loop the keyboard is tested, and if any key is pressed the delay is abandoned.

Note that saving registers, loading the delay and testing the command provide marginally enough time for the comparator to settle after the digital output. To guarantee enough time when RUN is used, a delay count of 0002 is used in this case.



TEST PROGRAM FOR ØIDSP

Figure 5-12

The comparator is read, displayed and saved within the delay loop, by:

IN	PORT2B	Read interrupt status
ANI	08	Mark comparator
STA	83FC	Display
MOV	C,A	Save comparator bit

At exit from the loop, either when the delay count reaches zero or when a key is pressed, the comparator bit is recovered from (C). Since the flag set by masking has been lost by the keyboard test and delay count, ORA A is executed to set or clear the zero flag according to the content of (A). At exit the zero flag is set if the digital value is less than the input voltage.

A trivial test program, shown in Figure 5-12, is suitable for debugging your Output/Input/Display subroutine (OIDSP), and also for calibration of the potentiometers. Connect the voltmeter to ANALOG OUT initially. When you key in a numeric value, with any command, it will be output to the D/A converter. Key in FA, STEP, and adjust the ANALOG OUT pot to obtain 2.50 volts on the voltmeter. Key in other values, and find the value at which the comparator output changes from low to high, as shown on the display. When the digital value is greater than the input signal, the comparator bit is set, and will be displayed as a bottom horizontal bar, indicating that the digital value must be reduced.

THIS PAGE INTENTIONALLY LEFT BLANK

TEST PROGRAM FOR VOLTAGE DISPLAY 5-51

		A	D	D	R	CODE				
CODING SHEET	8	20	0	3E		MVI	A,	80		Program Ports
			1	80						Port 1B out
			2	D3		OUT	CNT	1		
			3	07						
			4	3E		MVI	A,	92		
			5	92						
			6	D3		OUT	CNT	2		
			7	0F						
MICROCOMPUTER TRAINING SYSTEM	820	8	F3		DI					
		9	CD		CALL	ENTBY				
		A	36							(L) ← digital value
		B	03							(A) ← command
		C	FB		EI					
		D	4F		MOV	C,	A			(C) ← command
		E	CD		CALL	Φ	IDSP			
		F	00							
INTEGRATED COMPUTER SYSTEMS	821	0	82							
		1	C3		JMP	8208				
		2	08							
		3	82							
		4								
		5								
		6								
		7								
	8	0								
	1									
	2									
	3									
	4									
	5									
	6									
	7									
	8									

Figure 5-13a

OIDSF - Output, Input, Display

A	D	D	R	CODE						
8	2C	0		7D	MOV	A, L				Output digital value
	1			D3	OUT	PORT1B				to D/A converter
	2			05						
	3			D5	PUSH	D				Save registers
	4			C5	PUSH	B				
	5			11	LXI	D, 0002				Minimum delay
	6			02						for RUN command
	7			00						
	8			79	MOV	A, C				Test for RUN
	9			FE	CPI	14				
	A			14						
	B			CA	JZ	82D7				Jump past display
	C			D7						if RUN
	D			82						
	E			16	MVI	D, 10				For 1/4 second
	F			10						delay if not RUN
8	2D	0		F3	DI					Disable monitor
	1			D5	PUSH	D				during long delay
	2			7D	MOV	A, L				
	3			CD	CALL	DBYTE				
	4			95						
	5			02						
	6			D1	POP	D				
8	2D	7		DB	IN	PORT2B				Read Interrupt
	8			0D						Status Byte
	9			E6	ANI	08				Mask for
	A			08						A/D Comparator
	B			32	STA	83FC				Display
	C			FC						comparator bit
	D			83						
	E			4F	MOV	C, A				Save comparator
	F			00	NOOP					
8		0								
	1				ENTER	(L) =	DIGITAL VALUE			
	2				RETURN					
	3				(A) =	00	AND ZERO FLAG SET			
	4				IF (L) <	INPUT VOLTAGE				
	5									
	6				(A) =	08	AND NOT ZERO			
	7				IF (L) >	INPUT VOLTAGE				
	8									

CODING SHEET

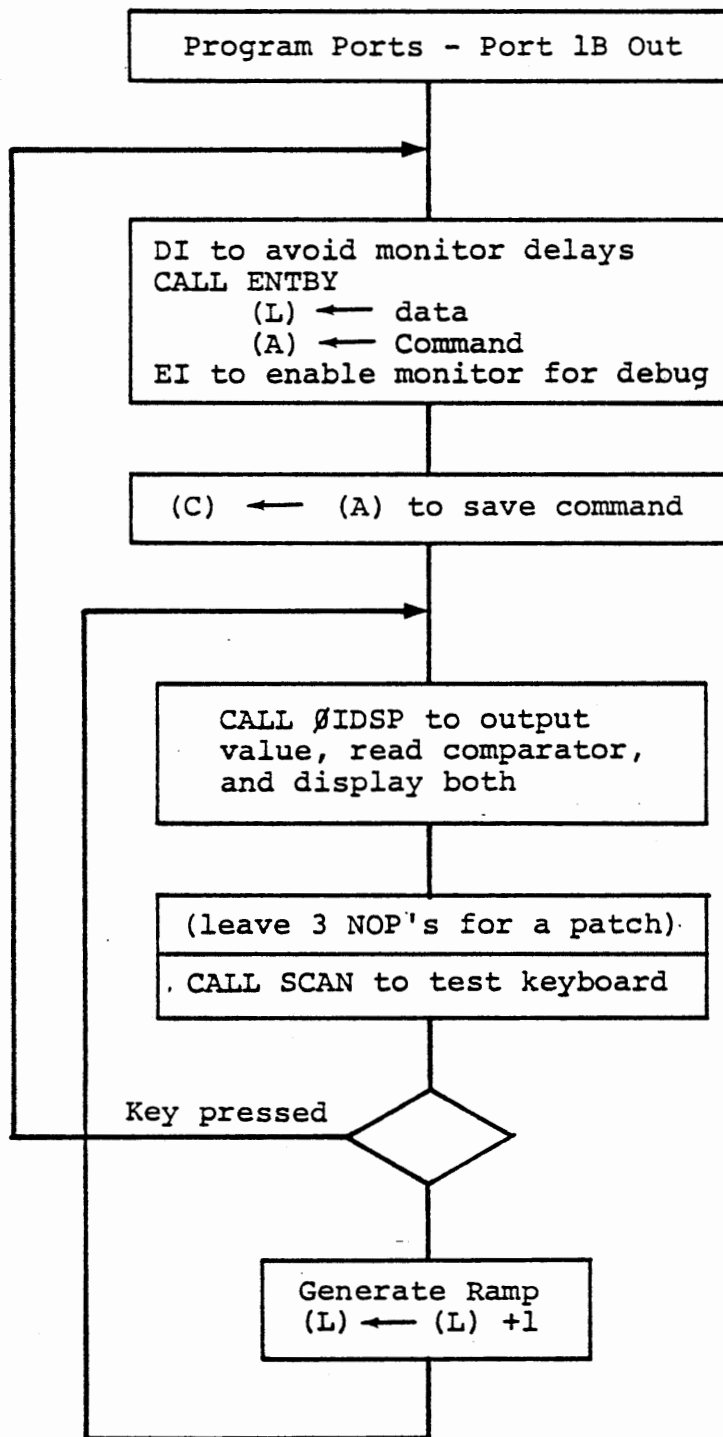
MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

Figure 5-13b

		A	D	D	R	CODE														
CODING SHEET	8	2E	0			DB		IN				PORT	0A						Test Keyboard	
			1			00													(FF if no key)	
			2			3C		INR		A										
			3			C2		JNZ		82EC									Exit if key pressed	
			4			EC														
			5			82														
			6			1B		DCX		D									Decrement and	
			7			7B		MOV		A, E									test delay count	
			8			B2		ORA		D										
			9			C2		JNZ		82D7										Loop until
MICROCOMPUTER TRAINING SYSTEM		A				D7													delay finished	
			B			82														
			C			79		MOV		A, C									(A) ← comparator bit	
			D			B7		ORA		A									Set zero if low	
			E			C1		POP		B									Exit.	
			F			D1		POP		D										
	INTEGRATED COMPUTER SYSTEMS	8	2F	0			FB		EI											
				1			C9		RET											
				2																
				3																
			4																	
			5																	
			6																	
			7																	
			A																	
			B																	
		C																		
		D																		
		E																		
		F																		
	8	0																		
		1																		
		2																		
		3																		
		4																		
		5																		
		6																		
		7																		
		8																		

Figure 5-13c



VOLTAGE RAMP GENERATOR

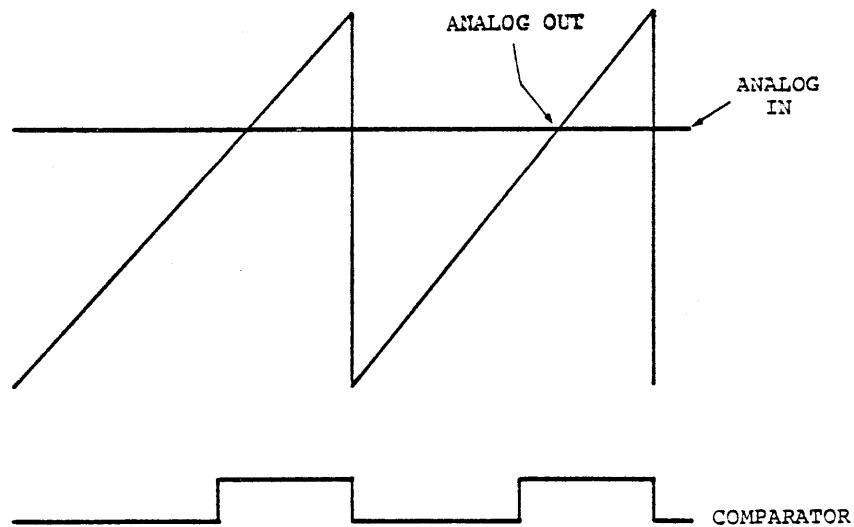
Figure 5-14

5.3.2 Ramping Voltmeter

Exercise

Modify the test program as shown in Figure 5-14, to generate an output voltage ramp. After output and delay, test the keyboard and go to CALL ENTBY only if a key is pressed, otherwise increment the digital value and go again to OIDSF. (Remember that OIDSF exits immediately when a key is pressed, but it does not indicate whether a key was pressed. Therefore, the main program must test the keyboard independently.)

Now the program will cycle the digital value in register L, counting from 00 through FF and back to 00. This will generate a voltage ramp at the D/A output, as shown in Figure 5-15. When the output is less than the input, the comparator bit will be low. When the output becomes greater, the comparator bit will be high. Your voltmeter on ANALOG OUT will show the ramp.



D/A OUTPUTS AND INPUTS

Figure 5-15

VOLTAGE RAMP GENERATOR

5-56

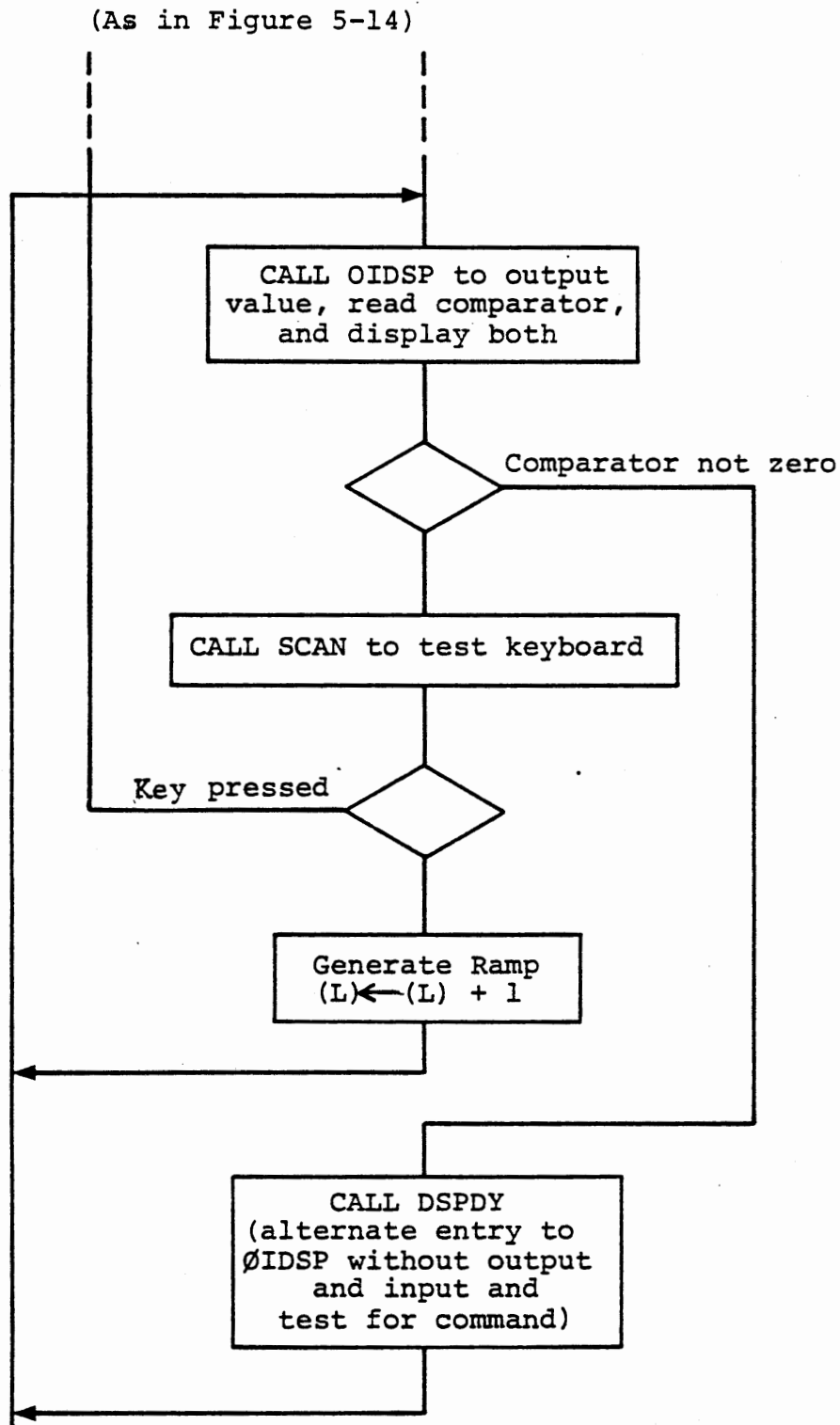
	A	D	D	R	CODE						
CODING SHEET	8	2	0		3E	MVI	A,	80			Program Ports
				1	80						Port 1B out
				2	D3	OUT		CNT1			
				3	07						
				4	3E	MVI	A,	92			
				5	92						
				6	D3	OUT		CNT2			
MICROCOMPUTER TRAINING SYSTEM	820			8	F3	DI					
				9	CD	CALL		ENTBY			
				A	36						(L) ← digital value
				B	03						(A) ← command
				C	FB	EI					
				D	4F	MOV	C,	A			(C) ← command
		820			E	CD	CALL		ΦIDSP		Output, Input
				F	00					Display, Delay	
MICROCOMPUTER SYSTEMS	821			0	82						
				1	00	NOP					Leave space
				2	00	NOP					for a patch
				3	00	NOP					
				4	CD	CALL		SCAN			Test keyboard
				5	57						
				6	02						
			7	DA	JC		8208			If key pressed	
			8	08						go to CALL ENTBY	
			9	82							
			A	2C	INR	L				Else increment	
			B	C3	JMP		820E			the digital value	
			C	0E						and go to output,	
			D	82						input and display	
			E								
			F								
INTEGRATED COMPUTER SYSTEMS	8			0							
				1							
				2							
				3							
				4							
				5							
				6							
				7							
			8								

Figure 5-16

You may want to shorten the delay for each step by reducing the value loaded for the delay loop in OIDSF. With a value of 1000 the full cycle of the ramp will take about 60 seconds.

Now watch the display of the comparator bit. It will be blank until the D/A output exceeds the input voltage. As soon as it appears, press NEXT and hold it down. This will stop the program (ENTBY will wait for release of the key) and the output voltage will be displayed. When you release the key ENTBY will return with 00 in register L to start a new ramp.

Obviously this function need not depend on your finger, since the processor has the decision bit available. OIDSF returns with the zero flag set when the comparator is low, and cleared when it is high. Insert JNZ after the return from OIDSF, to a patch that will display the result and restart the ramp. The program of Figure 5-17 calls an alternate entry to OIDSF that bypasses the digital value output and the check on the stored command, and loads a different delay time.



RAMPING VOLTMETER

Figure 5-17

After the result display, the digital value is set to zero to start a new ramp. You can see the ramp on your voltmeter. If you press RUN, the display and delay for intermediate results will be inhibited by OIDSF, and only the final value will be displayed.

Now move the voltmeter to ANALOG IN to observe the input signal. Compare the value displayed by the program with the measured voltage. They should agree closely, but if some error exists, you can adjust the ANALOG IN pot to compensate for it. Adjust the OPTO SENSE pot to change the input value and observe the value measured by the program, comparing it with the voltmeter.

RAMPING VOLTMETER

5-60

A	D	D	R	CODE						
8	2	0		3E	MVI	A,	80			Program Ports
			1	80						Port 1B out
			2	D3	OUT	CNT1				
			3	07						
			4	3E	MVI	A,	92			
			5	92						
			6	D3	OUT	CNT2				
			7	0F						
8	2	0	8	F3	DI					
			9	CD	CALL	ENTBY				
			A	36						(L) ← digital value
			B	03						(A) ← command
			C	FB	EI					
			D	4F	MOV	C,	A			(C) ← command
8	2	0	E	CD	CALL	DISP				Output, Input
			F	C0						Display, Delay
8	2	1	0	82						
			1	C2	JNZ	8220				Jump to
			2	20						display output
			3	82						
			4	CD	CALL	SCAN				Test keyboard
			5	57						
			6	02						
			7	DA	JR	8208				If key pressed
			8	08						go to CALL ENTBY
			9	82						
			A	2C	INR	L				Else increment
			B	C3	JMP	820E				the digital value
			C	0E						and go to output,
			D	82						input and display
			E							
			F							
8			0							
			1							
			2							
			3							
			4							
			5							
			6							
			7							
			8							

Figure 5-18a

RAMPING VOLTMETER - CONVERSION COMPLETE 5-61

	A	D	D	R	CODE							
CODING SHEET	8	2	2	0	CD	CALL	DSPDY	Display result				
		1			F2							
		2			82							
		3			2E	MVI	L, 00	Clear digital				
		4			00			value				
		5			C3	JMP	820E	Start new				
		6			0E			conversion				
	7			82								
	8											
	9											
	A											
	B											
	C											
MICROCOMPUTER TRAINING SYSTEM		D										
		E										
		F										
	8	0										
		1										
		2										
		3										
	4											
	5											
	6											
	7											
	8											
	9											
INTEGRATED COMPUTER SYSTEMS		A										
		B										
		C										
		D										
		E										
		F										
	8	0										
	1											
	2											
	3											
	4											
	5											
	6											
	7											
	8											

Figure 5-18b

OIDSF - Output, Input, Display

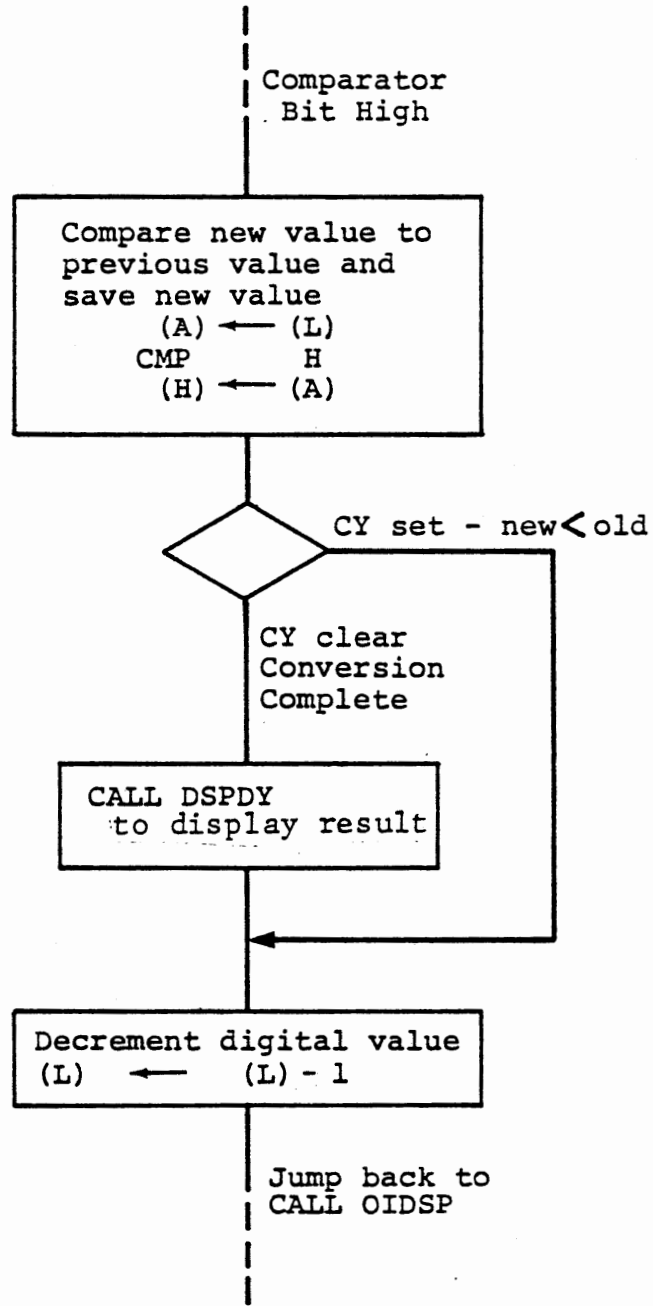
5-62

	A	D	R	CODE						
CODING SHEET	8	2C	0	7D	MOV	A, L				Output digital value
			1	D3	OUT	PORT 1 B				to D/A converter
			2	05						
			3	D5	PUSH	D				Save registers
			4	C5	PUSH	B				
			5	11	LXI	D, 0002				Minimum delay
			6	02						for RUN command
			7	00						
			8	79	MOV	A, C				Test for RUN
			9	FE	CPI	14				
MICROCOMPUTER TRAINING SYSTEM	A			14						
	B			CA	JZ	82D7				Jump past display
	C			D7						if RUN
	D			82						
	E			16	MVI	D, 10				For 1/4 second
	F			10						delay if not RUN
	8	2D	0	F3	DI					Disable monitor
			1	D5	PUSH	D				during long delay
			2	7D	MOV	A, L				
			3	CD	CALL	DBYTE				
INTEGRATED COMPUTER SYSTEMS			4	95						
			5	02						
			6	D1	POP	D				
	8	2D	7	DB	IN	PORT 2 B				Read Interrupt
			8	0D						Status Byte
			9	E6	ANI	08				Mask for
			A	08						A/D Comparator
			B	32	STA	83FC				Display
			C	FC						comparator bit
			D	83						
		E	4F	MOV	C, A				Save comparator	
		F	00	NOP						
	8		0							
			1		ENTER	(L) =	DIGITAL VALUE			
			2		RETURN					
			3		(A) = 00	AND	ZERO FLAG SET			
			4		IF (L) <	INPUT VOLTAGE				
			5							
			6		(A) = 08	AND	NOT ZERO			
			7		IF (L) >	INPUT VOLTAGE				
			8							

Figure 5-18c

	A	D	D	R	CODE								
CODING SHEET	8	2E	0		DB		IN			PORT	0A	Test keyboard (FF if no key)	
			1		00								
			2		3C		INR	A					
			3		C2		JNZ	82EC				Exit if key pressed	
			4		EC								
			5		82								
			6		1B		DCX	D					Decrement and test delay counter
			7		7B		MOV	A, E					
			8		B2		ORA	D					
			9		C2		JNZ	82D7					Loop until delay finished
		A		D7									
		B		82									
MICROCOMPUTER TRAINING SYSTEM	82E	C			79		MOV	A, C				(A) ← comparator bit	
		D			B7		ORA	A				Set zero if low	
		E			C1		POP	B				Exit	
		F			D1		POP	D					
	82F	0			FB		EI						
		1			C9		RET						
INTEGRATED COMPUTER SYSTEMS	82F	2			D5	*	PUSH	D				DSPDY -	
		3			C5		PUSH	B				Display final result for	
		4			11		LXI	D, 8000					
		5			00								
		6			80								
		7			C3		JMP	82D0					
		8			D0								
		9			82								
		A											
		B						* ALTERNATE ENTRY DSPDY DISPLAYS FINAL RESULT FOR 2 SECONDS					
	C												
	D												
	E												
	F												
	8			0									
				1									
				2									
				3									
				4									
				5									
				6									
				7									
				8									

Figure 5-18d



TRACKING VOLTMETER

Figure 5-19

5.3.3 Tracking Voltmeter

Exercise

If an analog to digital conversion is being performed on only one input signal, after the first conversion is complete, it is not necessary to generate repetitive ramps. Instead, the digital value can be decreased when the comparator indicates that it is greater than the input, and increased when it is less. Now the D/A voltage will track the input signal. Modify the ramping voltmeter program so that it enters a tracking mode when a conversion is complete. When a key is pressed, it should start a new conversion. As before, the RUN command will cause display of completed conversion only, while NEXT will call for display of intermediate results. When the program is in tracking mode, the conversion is complete when the comparator bit becomes high after an increase in the digital value.

Figure 5-19 shows the modification to the ramping voltmeter. The program is identical except for the action taken when the comparator bit is high. Now after each test by OIDSF, if the comparator is low the digital value is incremented as before, but if it is high the digital value is decremented. If the comparator remains high for successively lower digital values the decrementing continues, so as to track a decreasing voltage. When the comparator is high at a digital value equal to or greater than the previous value, the conversion is complete and the long display is made. Now the digital value output will alternately increase and decrease by one bit. A complete new conversion, starting at zero will be started only when a command key

is entered.

Connect your voltmeter to the ANALOG OUT signal, and watch it track the input as you adjust the SENSE pot.

TRACKING VOLTMETER -

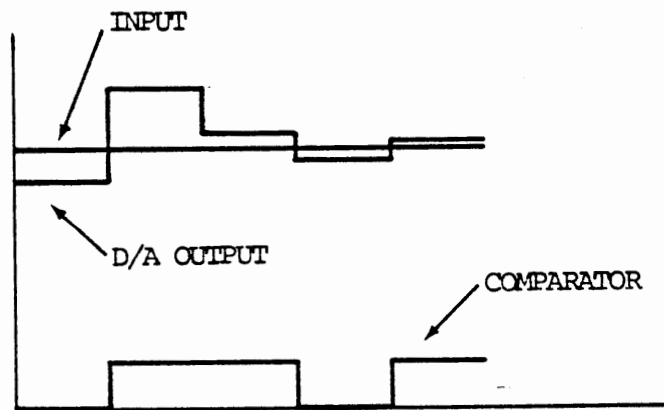
A D D R		CODE						
CODING SHEET	8	220	7D	MOV	A, L		Compare present	
		1	BC	CMP	H		digital value with	
		2	67	MOV	H, A		previous value in H	
		3	D4	CNC	DISPDY		Display result if	
		4	F2				present value	
		5	82				greater (ramping up)	
		6	2D	DCR	L			
		7	C3	JMP	820E			
		8	0E					
		9	82					
MICROCOMPUTER TRAINING SYSTEM	A			PROGRAM	AT	8200-821F		
	B			AND	DISP	82C0-82FF		
	C			IDENTICAL	TO	SIMPLE		
	D			RAMPING	VOLTMETER			
	E							
	F							
	INTEGRATED COMPUTER SYSTEMS	8	0					
			1					
			2					
			3					
		4						
		5						
		6						
		7						
		8						
		9						

Figure 5-20

5.3.4 Successive Approximation Voltmeter

Exercise

The ramping voltmeter is relatively slow in reaching equality of output and input, and the time required for a conversion varies according to the input voltage. The successive approximation method overcomes both of these drawbacks. Instead of starting at zero and ramping upward, the D/A converter output is started at one half of full scale and increased or decreased according to the comparator signal by successively smaller amounts ($1/2$, $1/4$, $1/8$ - - -) until the increment or decrement of one least significant bit has been processed. Figure 5-21, below, shows the signals.

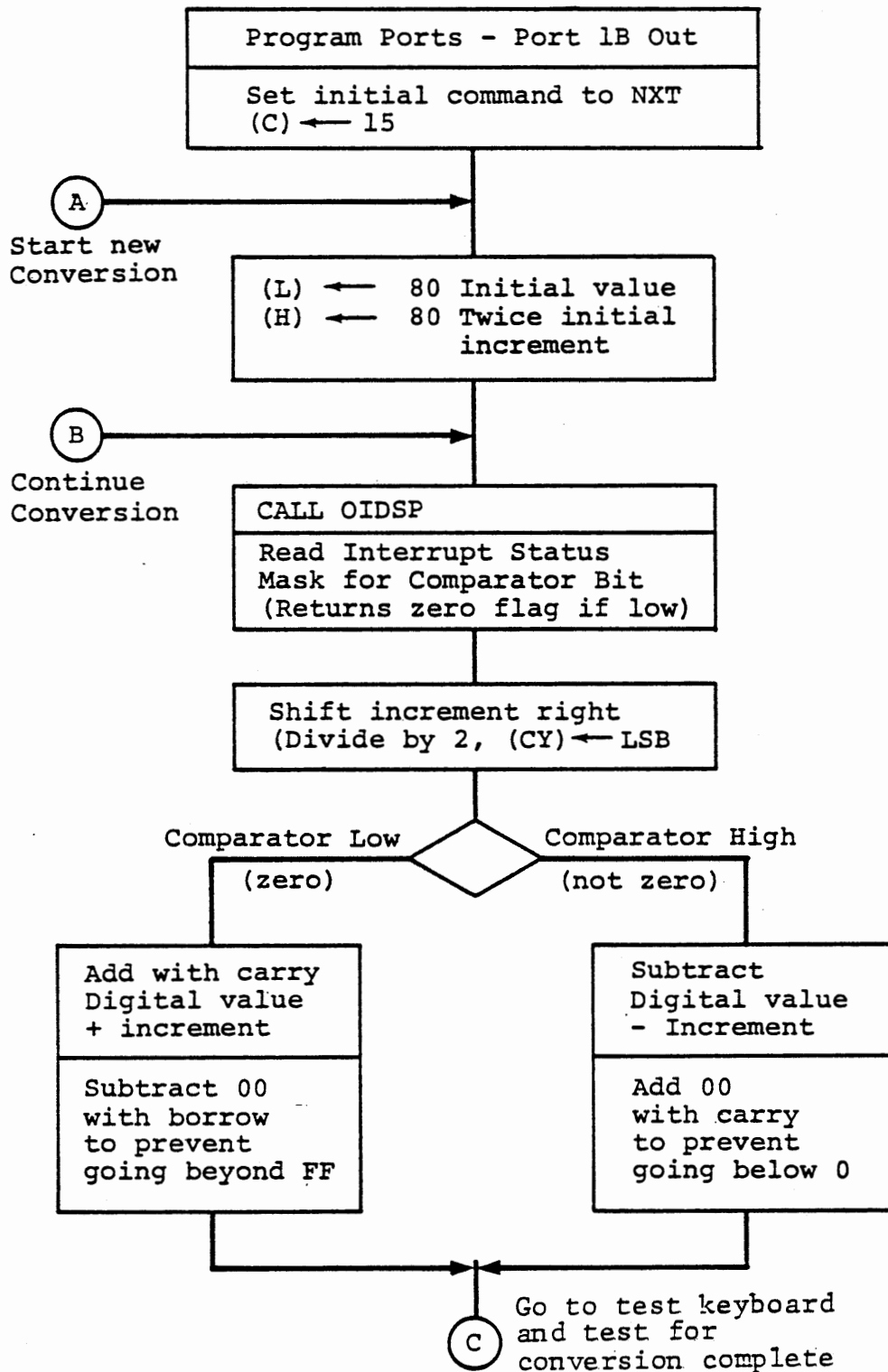


SUCCESSIVE APPROXIMATION SIGNALS

Figure 5-21

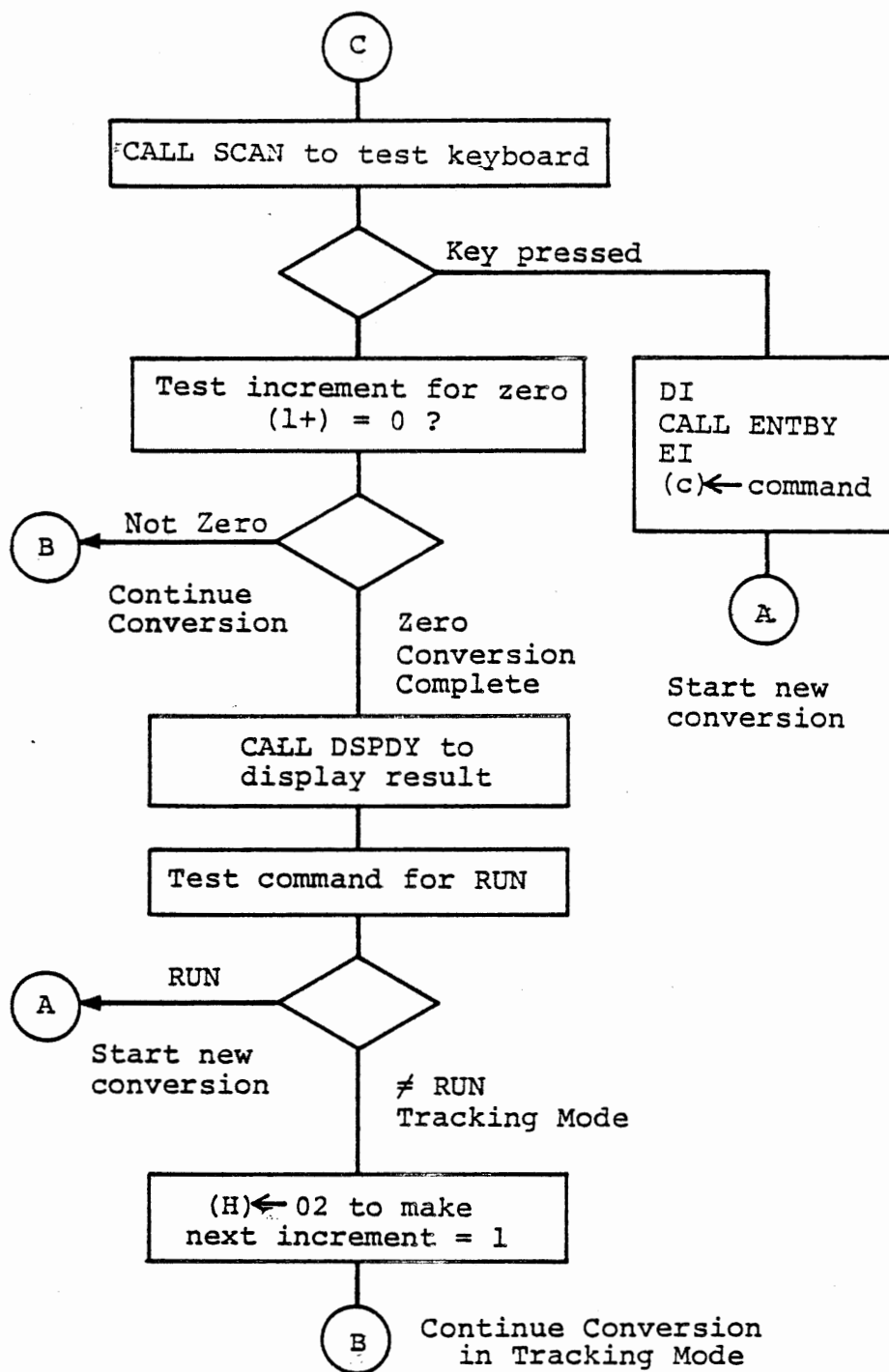
After all of the successively smaller increments or decrements down to one least significant bit have been processed, the digital value is within + or - 1 bit of the analog input. If the process were stopped here, the result would always be an odd number, since in the last step the digital value, up to here an even number, was either increased or decreased by one. The procedure of Figure 5-22 avoids this problem by shifting the increment right just before adding or subtracting, and doing an ADC if the comparator is low, but SUB if it is high. Thus when the increment reaches zero, the digital value may still be increased, but will not be decreased. The result is therefore within $\pm 1/2$ bit of the input value.

As in the preceding program, we will enter a tracking mode when the conversion is complete, and start a new conversion when NEXT is pressed. With the RUN key, however, we will start a new conversion as soon as the old conversion has been completed.



SUCCESSIVE APPROXIMATION VOLTMETER

Figure 5-22a



SUCCESSIVE APPROXIMATION VOLTMETER - continued

Figure 5-22b

SUCCESSIVE APPROXIMATION VOLTMETER

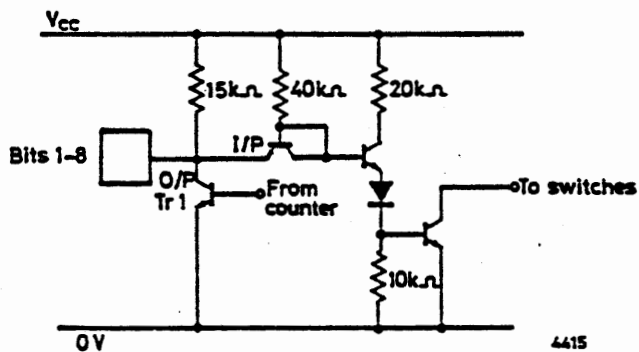
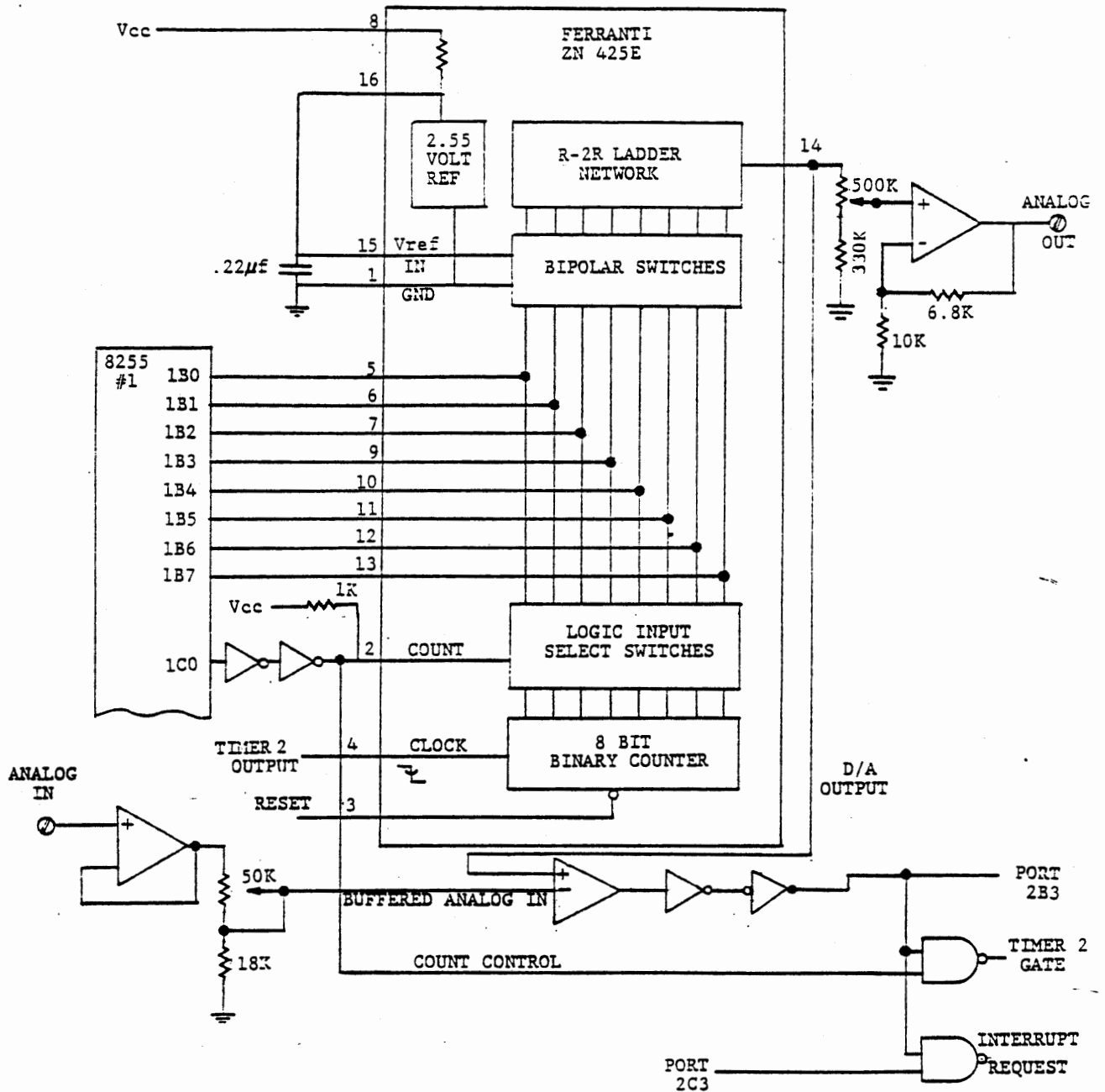
	A D D R	CODE							
CODING SHEET	8 2 0 0	3 E		M V I	A ,	8 0			
		1	8 0						
		2	D 3		O U T	C N T 1			
		3	0 7						
		4	3 E		M V I	A ,	9 2		
		5	9 2						
		6	D 3		O U T	C N T 2			
MICROCOMPUTER TRAINING SYSTEM	8 2 0 8	3 E		M V I	A ,	1 5			Make command NXT
		9	1 5						for initial conversion
	8 2 0 A	4 F		M O V	C ,	A			(C) ← command
	8 2 0 B	2 1		L X I	H ,	8 0 8 0			
		C	8 0						(L) ← initial value
		D	8 0						(H) ← 2 x initial increment
	8 2 0 E	C D		C A L L	Φ I D S P				
		F	C 0						
	8 2 1 0	8 2							
		1	7 C		M O V	A ,	H		
	2	1 F		R A R					
	3	6 7		M O V	H ,	A			
	4	7 D		M O V	A ,	L			
	5	C 2		J N Z	8 2 1 E				
	6	1 E							
	7	8 2							
	8	8 C		A D C	H				
	9	D E		S B I	0 0				
	A	0 0							
	B	C 3		J M P	8 2 2 1				
	C	2 1							
	D	8 2							
INTEGRATED COMPUTER SYSTEMS	8 2 1 E	9 4		S U B	H				
		F	C E		A C I	0 0			
	8 2 2 0	0 0							
	8 2 2 1	6 F		M O V	L ,	A			
		2							
		3							
		4							
		5			NOTE: Φ I D S P (FIG 5-18 c, d)				
	6			I S R E Q U I R E D					
	7								
	8								

Figure 5-23a

SUCCESSIVE APPROXIMATION VM - cont'd

A D D R		CODE							
8	0								
	1								
822	2	CD		CALL	SCAN				Test keyboard
	3	57							
	4	02							
	5	DA		JC	823B				Jump to CALLENTBY
	6	3B							if key pressed
	7	82							
	8	7C		MOV	A, H				Test increment
	9	B7		ORA	A				for zero
A	C2			JNZ	820E				Loop to CALLODSP
B	0E								if conversion
C	82								not complete
D	CD			CALL	DSPDY				Display result
E	F2								when conversion
F	82								is complete
823	0	79		MOV	A, C				Test command
	1	FE		CPI	14				for RUN
	2	14							
	3	CA		JZ	820B				If RUN go to
	4	0B							start new
	5	82							conversion
	6	26		MVI	H, 02				To force next
	7	02							increment to 01
	8	C3		JMP	820E				Continue in
	9	0E							tracking mode
A	82								
823	B	F3		DI					
	C	CD		CALL	ENTBY				
	D	36							
	E	03							
	F	FB		EI					
824	0	C3		JMP	820A				
824	1	0A							
824	2	82							
	3								
	4								
	5								
	6								
	7								
	8								

Figure 5-23b



Logic Input Circuitry

FERRANTE A/D LOGIC

Figure 5-24

5.4 Automatic A/D Input

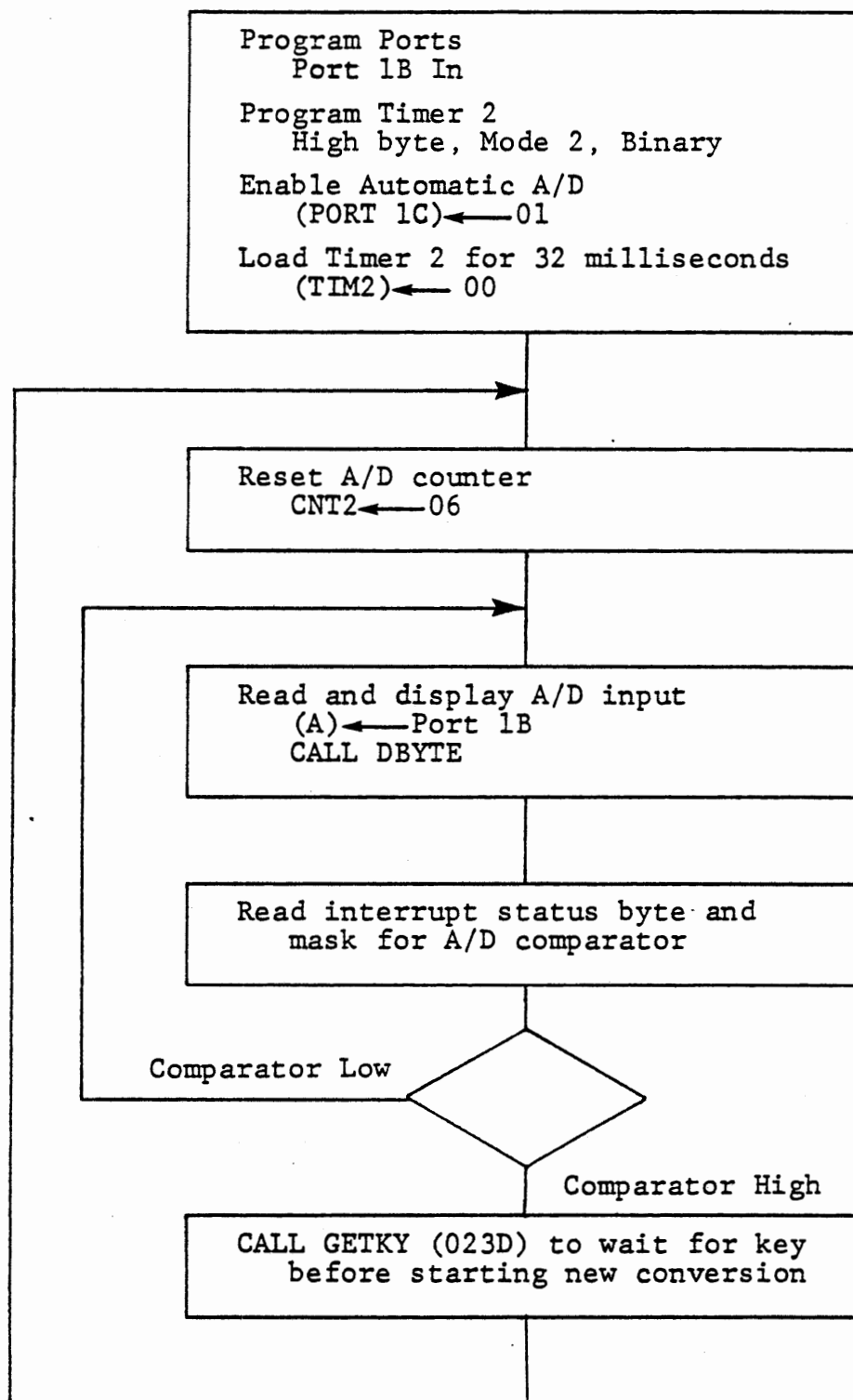
In Section 5.3, we have been using the microprocessor's arithmetic capability to control the D/A output for comparison with an input. Generating a voltage ramp requires such simple logic that our Ferranti ZN 425E D/A converter includes the necessary counter and switches.

Two control inputs of the 425 must be operated for automatic A/D input, and a clock signal must be provided for the internal counter. Figure 5-24 shows the logic of the 425. The internal counter is driven by its clock signal and cleared by its reset signal. If the "count" control signal is low, as we have been using it, the counter outputs are isolated by the open collector transistor switches. Then the R-2R ladder is controlled by the data output from Port 1B. If the "count" control signal is high, internal gating allows the internal counter to control the data on the I/O port and the switches of the R-2R ladder. Port 1B must be programmed for input, and the data from the A/D converter can be read there.

The clock for the internal counter is taken from timer 2. When it generates clock pulses, the Ferranti counts up, generating a voltage ramp at the D/A output. If it exceeds the analog input signal, the comparator output goes high. This signal is NAND gated with count control, so that when both signals are high the timer 2 gate input becomes low, inhibiting further counting. The Ferranti's counter now contains the digital value corresponding to the analog input voltage. This value, available at port 1B, is held until the counter is reset.

THIS PAGE INTENTIONALLY LEFT BLANK

The reset input to the 425 counter is generated when the A/D comparator interrupt is enabled or disabled. The D/A output becomes zero, so the comparator output goes low and clears the interrupt. Therefore this interrupt behaves as though it had a latch cleared by the enable or disable interrupt command. There is an important constraint on treating this as a latch, as we shall see in the following exercise. First we will demonstrate the automatic ramp generation.



AUTOMATIC A/D INPUT

FIGURE 5-25

5.4.1 Reading A/D Input

EXERCISE

Figure 5-25 shows a program to operate the Ferranti 425 in its automatic A/D input mode. Port 1B is programmed for input and Port 1C0 is set high to enable the counter. Timer 2 provides the clock to the Ferranti: here it is set to a maximum delay to make the ramp visible.

A new conversion is started by resetting the A/D counter. Since we are not using an interrupt system, the disable command is given by writing 06 to CNT2. The content of the A/D counter is read and displayed, and then the comparator input is tested by:

```
IN      PORT2B      Read interrupt status
ANI     08          Mark for A/D comparator
```

The input, display and test procedure is repeated until the comparator is high. The increasing ramp is readily observed in the display, and can also be seen by connecting your voltmeter to ANALOG OUT. When the comparator becomes high the program waits for a key to be pressed, and then starts a new conversion.

AUTOMATIC A/D INPUT

5-80

	A	D	D	R	CODE						
CODING SHEET	8	2	0	0	3E	MVI	A,	82			Program Ports
				1	82						Port 1B In
				2	D3	OUT	CNT1				for A/D
				3	07						
				4	3E	MVI	A,	92			
				5	92						
				6	D3	OUT	CNT2				
				7	0F						
				8	3E	MVI	A,	A4			Program Timer 2
				9	A4						High byte
MICROCOMPUTER TRAINING SYSTEM				A	D3	OUT	TIMCT			Mode 2	
				B	17					binary	
				C	3E	MVI	A,	01			Set Port 1C0 high
				D	01						to enable
				E	D3	OUT	PORT1C				automatic A/D
				F	06						
		8	2	1	0	AF	XRA	A			Load Timer 2
				1	D3	OUT	TIM2				for 32 milliseconds
				2	16						
		8	2	1	3	3E	MVI	A,	06		Reset A/D Counter
			4	06						Disable interrupt	
INTEGRATED COMPUTER SYSTEMS				5	D3	OUT	CNT2				
				6	0F						
		8	2	1	7	DB	IN	PORT1B			Read A/D Input
				8	05						
				9	CD	CALL	DBYTE				Display A/D
				A	95						
				B	02						
				C	DB	IN	PORT2B				Read interrupt
				D	0D						status byte
				E	E6	ANI	08				Mask for
			F	08						A/D Comparator	
	8	2	2	0	CA	JZ	8217			Loop to read	
			1	17						and display input	
			2	82						unless comparator	
			3	CD	CALL	GETKY				high. Then wait	
			4	3D						for key, and	
			5	02						start a new	
			6	C3	JMP	8213				conversion	
			7	13							
			8	82							

Figure 5-26

Now interchange the test of the comparator and the input and display of the A/D value.

```

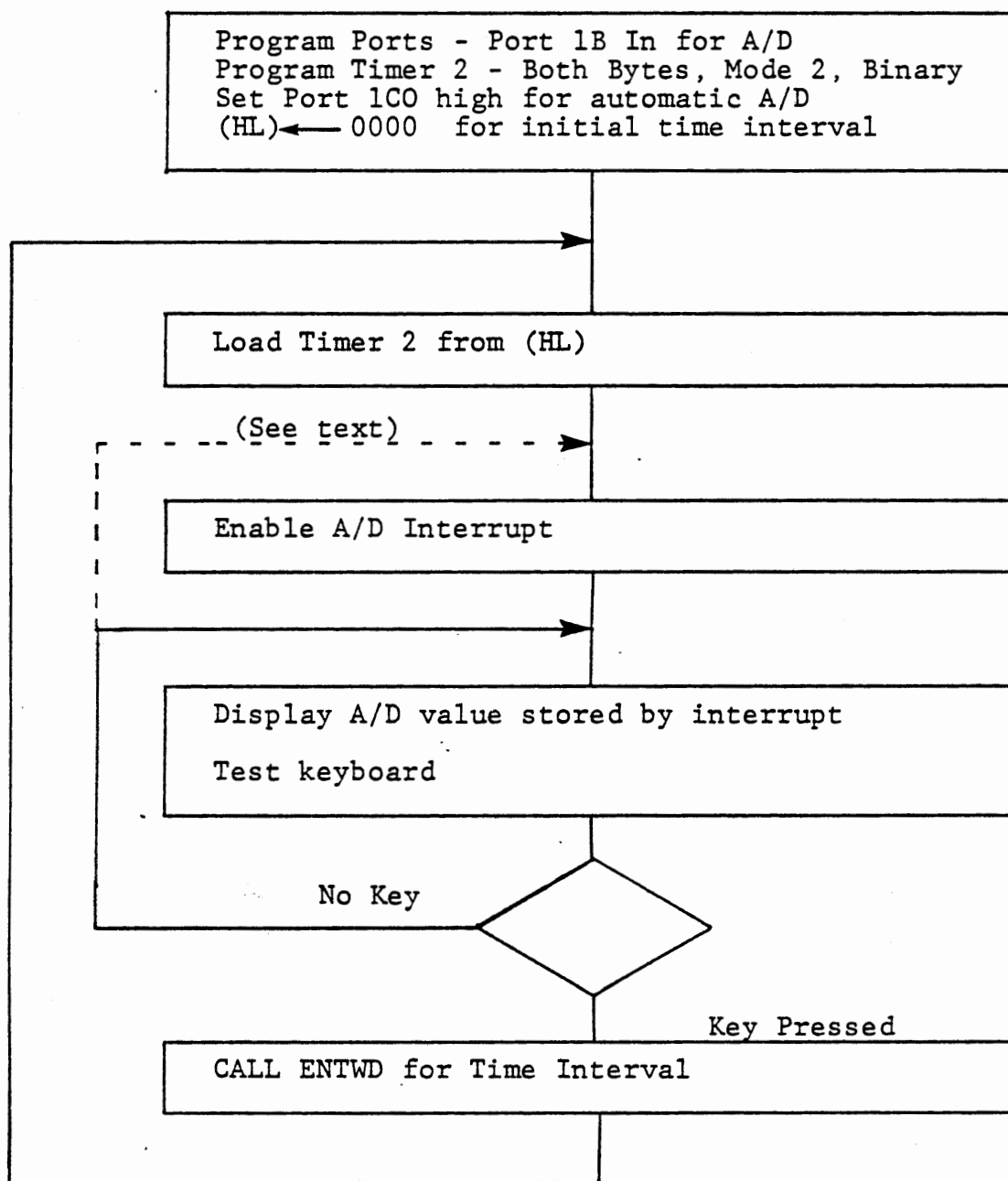
CNVRT      MVI      A,06          Reset A/D counter
           OUT      CNT2
TEST       IN       PORT2B       Wait for comparator
           ANI      08           to become high
           JZ       TEST
           IN       PORT1B       Read A/D
           CALL     DBYTE        Display voltage
           CALL     GETKY        Wait for key
           JMP      CNVRT

```

It appears that this should display only the final voltage, since we read it when the comparator is high. Instead, it sometimes displays 00! This demonstrates the constraint previously mentioned. The A/D comparator does not reset instantly, as does a latch. After the reset, the D/A output goes low very quickly, but the comparator takes several microseconds to respond. Therefore reading the comparator immediately after a reset may find it still high, and the program above falsely supposes that the conversion is finished. If you press a key quickly twice in succession, or press it again as soon as 00 is displayed, then the program will make the conversion properly. A better scheme is to rearrange the program again, placing the reset before CALL GETKY. Then, if you like, replace CALL GETKY with CALL DELAY (0236) for a fully automatic voltmeter. You can speed it up by loading Timer 2 with 02 instead of 00, to give a clock interval of 125

THIS PAGE INTENTIONALLY LEFT BLANK

microseconds instead of 32 milliseconds. In the solution given in Figure 5-26, this can be done by deleting the XRA A instruction before loading Timer 2. We shall investigate the effect of the clock interval in the following exercise.



RST6 Interrupt Service

PUSH PSW only
 Read and Store A/D Input
 Reset A/D Counter
 POP PSW, EI, RET

A/D INPUT WITH INTERRUPT

FIGURE 5-27

5.4.2 A/D Input with Interrupt

EXERCISE

Since the A/D comparator generates an interrupt signal, the input can be accepted with an interrupt service routine. In this exercise we shall demonstrate two risks with the interrupt service and a technique that protects against both.

The main program (Figure 5-27) programs Timer 2 for two bytes and initially loads it for 32 milliseconds. The A/D interrupt is enabled. Now when a conversion is complete the interrupt service routine reads the A/D input, stores the value in memory, and resets the A/D counter. The main program displays the stored value. Now if a key is pressed ENTWD accepts a new interval for the clock to the Ferranti and loads Timer 2. Otherwise it merely displays the voltage repeatedly.

Do you see the danger in this program? Write and test the program and try to identify the problem that may occur.

Interrupt service only provides the time for POP PSW and EI after resetting the A/D counter. If the comparator does not respond in that time, the interrupt is still present when RET is executed. The main program is never allowed to execute an instruction after enabling the A/D interrupt. (This problem is not certain to occur, however).

A/D INPUT WITH INTERRUPT

5-86

	A	D	D	R	CODE							
CODING SHEET	8	2	0		3E	MVI	A,	82			Program Ports	
				1	82						Port 1B In	
				2	D3	OUT	CNT1					
				3	07							
				4	3E	MVI	A,	92				
				5	92							
				6	D3	OUT	CNT2					
				7	0F							
				8	3E	MVI	A,	B4			Program Timer 2	
				9	B4						Both bytes	
MICROCOMPUTER TRAINING SYSTEM				A	D3	OUT	TIMCT				Mode 2	
				B	17						Binary	
				C	3E	MVI	A,	01				
				D	01							
				E	D3	OUT	PORTIC					
				F	06							
		8	2	1	0	21	LXI	H,	0000			Initial value
					1	00						for clock interval
					2	00						
			8	2	1	3	7D	MOV	A,	L		Load Timer 2
				4	D3	OUT	TIM2				with clock	
				5	16						interval for	
				6	7C	MOV	A,	H			A/D counter	
				7	D3	OUT	TIM2					
				8	16							
				9	C3	JMP	8240				Jump past	
INTEGRATED COMPUTER SYSTEMS				A	40						interrupt service	
				B	82							
				C								
				D								
				E								
				F								
		8			0							
					1							
					2							
					3							
				4								
				5								
				6								
				7								
				8								

Figure 5-28a

This problem is easily solved. Before POP PSW, EI, RET insert CALL DELAY (0236) in the interrupt service routine. This monitor subroutine generates a delay of about one millisecond, using only the A register. Now there is plenty of time for the A/D comparator to settle. The voltmeter program should now operate correctly, although it is very slow because of the long clock interval. Try shorter intervals: 4000, 1000, 0400, 0100, 0050. Adjust the OPTO SENSE pot and see the display follow it.

If you make the clock interval short enough, and the voltage low enough, the main loop will not be able to operate because a correct A/D conversion is completed before DELAY returns. Again the interrupt is already there before EI, RET is executed.

To solve both problems, reset the A/D counter with a disable command instead of enable:

```
MVI    A,06
OUT    CNT2
```

In the main program, enable the A/D interrupt by writing 08 to Port 2C, and include this in the short loop. Now an interrupt can occur only once for each pass through the main loop, so it is guaranteed to run, with or without the delay in interrupt service.

The slow response of the comparator introduces another problem which can be investigated with this program. With a fast clock, several counts may occur after the D/A output actually exceeds the input voltage, but before the comparator responds and inhibits the clock.

This leads to slightly different results, depending on the clock rate. For any given rate, however, the error is fixed and can be compensated for. In our experiments it is small enough to be ignored. We shall generally use a time interval of 20 (hex), which gives a 16 microsecond clock.

The LM324N op-amp was selected here because it is able to operate on a single +5 volt supply and still work with signals down to zero volts. Faster op-amps cannot handle signals as low as the negative supply voltage, so would require an additional power supply. A specialized voltage comparator circuit such as National Semiconductor LM339N would provide fast response with the single supply voltage, but would not serve for the other op-amp functions needed here.

5.5 DIGITAL NOISE FILTER

If the voltage at the analog input is very close to a particular D/A output value, it is likely that the least significant bit of the measured voltage will change from one reading to another. If noise is present on the analog signal, several bits may change. A filter is needed to reduce the noise. We connected a capacitor at the input (Figure 5-10) for this purpose.

Filtering can also be done by digital processing. An excellent technique for estimating the present value of a signal that is changing with time, but also includes noise is to calculate a running average that gives less and less weight to older data measurements.

$$E_i = f_0V_i + f_1V_{i-1} + f_2V_{i-2} + \dots$$

Where the $V(i)$ are successive measurements and the $f(j)$ are weights applied to them. Obviously, the weights must be selected so that if V does not change $E(i)$ will equal V . This is achieved by the following expression, which also minimizes the data to be stored.

$$E_i = fV_i + (1-f) E_{i-1}$$

Storage is required only for the present estimate, $E(i)$, as a variable and a single value of f as a constant, yet is exactly equivalent to the infinite series of the first expression with:

$$\begin{aligned}
 f_0 &= f \\
 f_1 &= f(1-f) \\
 f_2 &= f(1-f)^2 \\
 f_3 &= f(1-f)^3 \quad \text{etcetera}
 \end{aligned}$$

If we were to use $f = 0.25$, these would be:

$$\begin{aligned}
 f_0 &= 0.25 \\
 f_1 &= 0.1875 \\
 f_2 &= 0.140625 \\
 f_3 &= 0.10546875 \quad \text{etcetera}
 \end{aligned}$$

Greater values of f lead to faster response of the estimate to new data, while smaller values give more noise filtering.

5.5.1 Filter Program Algorithm

The filter calculations will be performed by a subroutine FILTR. To simplify the calculations, we will restrict the value of f to $1/2$, $1/4$, $1/8$ or $1/16$. With such power of 2 fractions, the multiplication and divisions required become simple shifts by n bits, where $f = 1/2^n$. The expression to be calculated then becomes:

$$E_i = \frac{V_i + (2^n - 1) E_{i-1}}{2^n}$$

Data read from the A/D input have single byte precision, but to avoid the loss of the less significant bits of each measurement, we will carry out calculations and store results with two byte precision. Rather than storing $E(i)$ after each new input and calculation, we will

store $2^n E_i$ which will be used at the next calculation. Suppose that we choose $f = 1/4$, or $n=2$. Then the stored value is $4E_i$, which represents $4E_{i-1}$ when a new measured value V_i is obtained. The algorithm is shown below. (For convenience in notation let $m = 2^n$).

STEP	RESULT	
	$m = 4$	general
Recover stored data	$4E_{i-1}$	mE_{i-1}
Multiply by m	$16E_{i-1}$	$m^2 E_{i-1}$
Subtract stored data	$12E_{i-1}$	$m(m-1)E_{i-1}$
Divide by m	$3E_{i-1}$	$(m-1)E_{i-1}$
Add new measurement	$4E_i$	mE_i
Store result		
Divide by m	E_i	E_i

The multiplications and divisions by m are simple shifts of n bits, so the constant to be stored is n . To use this for single byte precision for the measured value and double byte precision for the arithmetic, n must be no greater than 4 ($m=16$), since one intermediate result has the data shifted left by $2n$ bits from the measured value. In fact, $n=4$, making $f = 1/16$, gives rather slower response than we will generally want.

5.5.2 Program Definitions

Subroutine FILTR and a local subroutine SHFTN are defined below and depicted in Figure 5-29.

The program in Figure 5-30 displays both filtered voltage, E_i , and inputted voltage, V_i . The leftmost two hex digits are E_i and the next two digits are V_i . The value for n (1,2,3 or 4) is entered via the keyboard and displayed in the rightmost two display digits.

5.5.2.1 Subroutine FILTR

Calculates estimated value of a variable with repetitive measurements containing noise.

Enter with new data in register (HL) addressing a four byte memory area:

$$((HL)) = n$$

$$((HL) + 1,2) = 2^n E_{i-1}$$

$$((HL + 3) = E_i \text{ (not used)}$$

Calculates and stores (in the same two locations)

$$2^n E_i = (2^n - 1) E_i + V_i$$

Returns

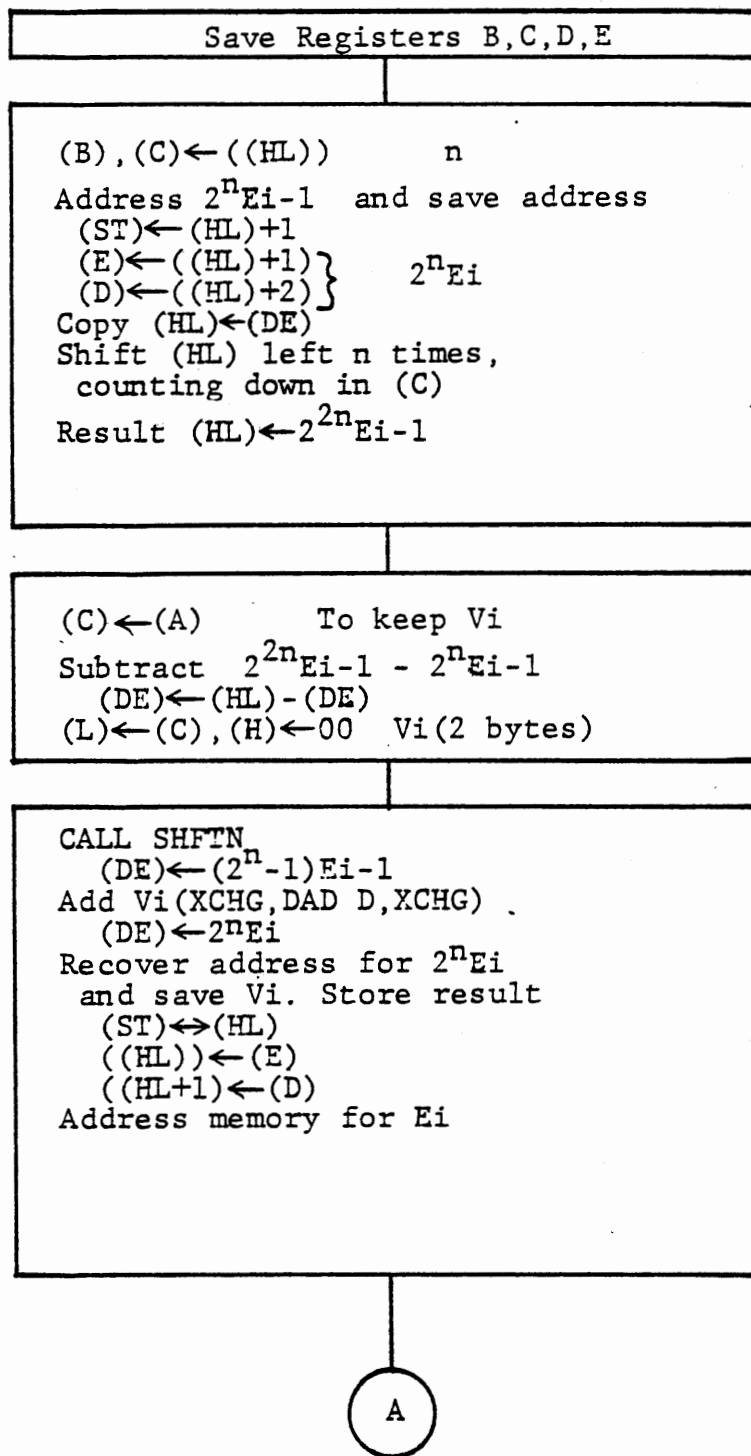
$$(A) = E_i = \frac{(2^n - 1) E_i + V_i}{2^n}$$

$$(H) = E_i$$

$$(L) = V_i$$

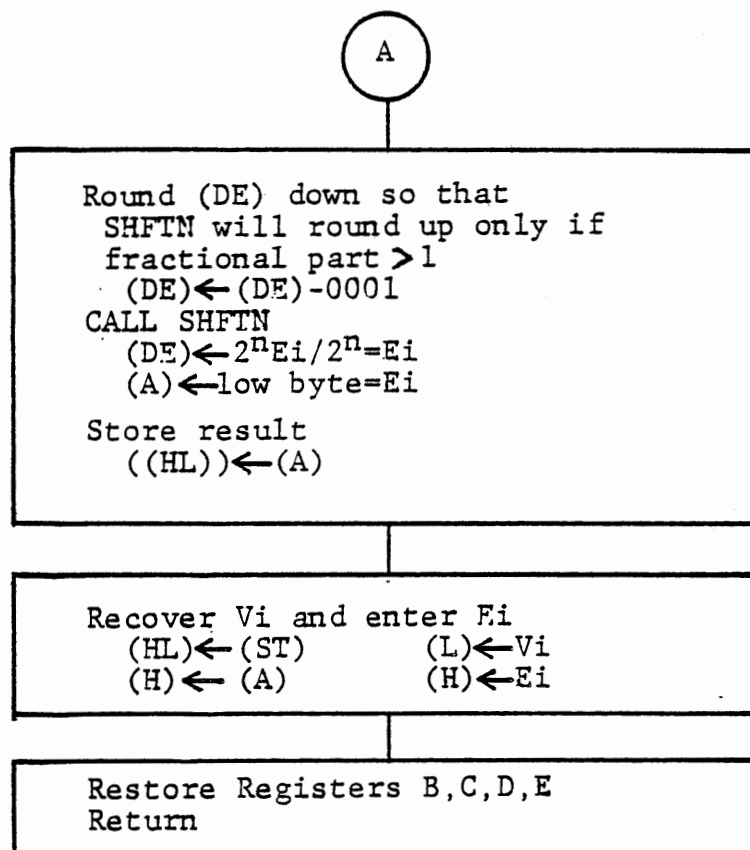
addressing high byte of storage also loaded with $E(i)$.

Registers B,C,D and E are preserved.



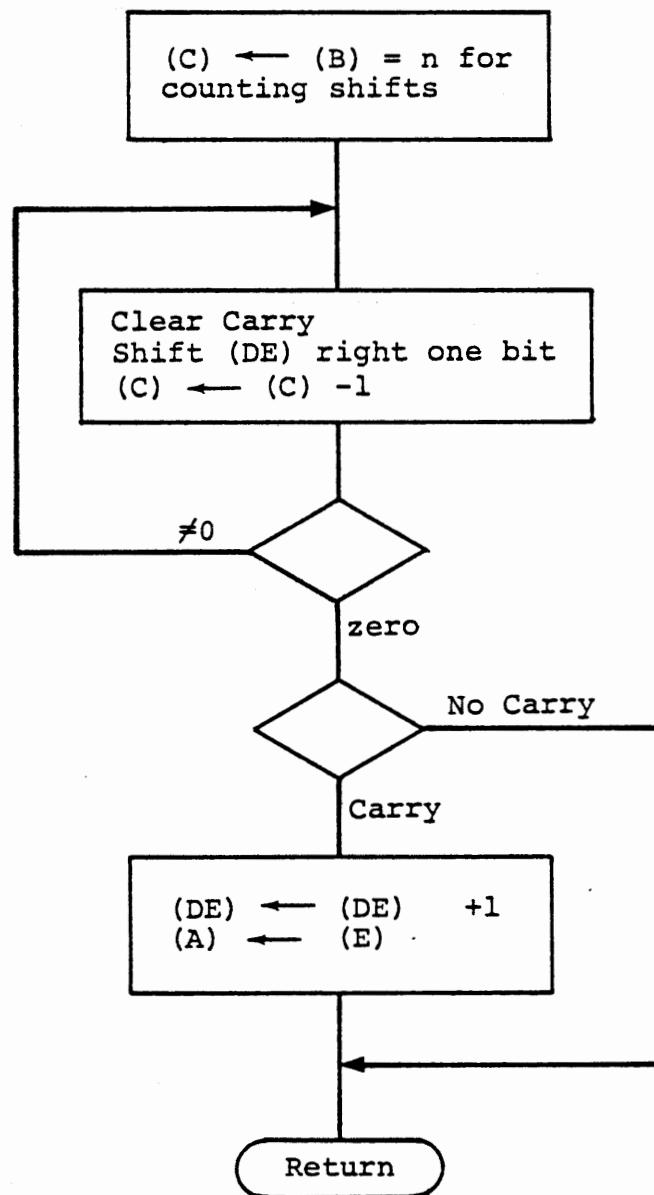
SUBROUTINE FILTR

FIGURE 5-29a



SUBROUTINE FILTR (continued)

FIGURE 5-29b



SUBROUTINE SHFTN

Figure 5-29c

5.5.2.2 Subroutine SHFTN

Shifts a data word right n bits with rounding.

Enter with (B) = n

(DE) = data word

Return with

(B) = n

(B) = n (unchanged);

(DE) = data word / 2^n ; (A) = less

Register C is cleared. Registers H and L are preserved. The carry flag is set if roundup has occurred.

Roundup occurs if the highest bit shifted out is 1. That is, if the fractional part of $(\text{data}/(2^n))$ is greater than or equal to one half.

THIS PAGE INTENTIONALLY LEFT BLANK

A comment on rounding of the calculations is necessary, because the requirement is not at all obvious. When the division of $m*(m-1)*E(i-1)$ is done, round up should occur if the highest bit shifted out is 1. That is, if the fractional part of the result is equal to or greater than $1/2$. If this roundup is not done, the final result, with constant input, is always one less than the measured value. When the division of $m*E(i)$ is done to obtain $E(i)$ for display, the roundup must occur only if the fractional part is greater than $1/2$. Otherwise the filter can never reach a value of zero. This is handled by a DCX D before calling SHFTN the second time.

Modify the A/D input with interrupt program (Figure 5-28) to use FILTR. Display both the measured value and the filtered value, when an interrupt occurs. Use the keyboard input to accept a new value of n for the filter. When a new value is entered, clear the existing $mE(i)$ from memory, to ensure that the high bits (beyond the precision being used) will not be left with data.

Test the program with the existing connections (Figure 5-10). Then remove the filter capacitor and test it again.

A/D INPUT WITH FILTER - INITIALIZE

	A	D	D	R	CODE					
CODING SHEET	8	2	0	0	3E	MVI	A,	82		Initialization
				1	82					identical to
				2	D3	OUT	CNT1			Figure 5-28a
				3	07					except as marked
				4	3E	MVI	A,	92		
				5	92					
				6	D3	OUT	CNT2			
				7	0F					
				8	3E	MVI	A,	B4		
				9	B4					
MICROCOMPUTER TRAINING SYSTEM				A	D3	OUT	TIMCT			
				B	17					
				C	3E	MVI	A,	01		
				D	01					
				E	D3	OUT	PORTIC			
				F	06					
		8	2	1	0	21	LXI	H,	0020	Fixed value for
				1		20	*			clock interval-
				2		00				16 μ sec clock
				3		7D	MOV	A,	L	
			4		D3	OUT	TIM2			
			5		16					
			6		7C	MOV	A,	H		
			7		D3	OUT	TIM2			
			8		16					
			9		C3	JMP	8250		Jump past	
INTEGRATED COMPUTER SYSTEMS				A	50	*			interrupt service	
				B	82					
				C						
				D						
				E						
				F						
										* CHANGED FROM FIGURE 5-28a
		8		0						
				1						
				2						
			3							
			4							
			5							
			6							
			7							
			8							

Figure 5-30a

A/D INTERRUPT WITH FILTR

	A	D	D	R	CODE					
CODING SHEET	8	2	3	0	FS		PUSH	PSW		
				1	ES		PUSH	H		
				2	DS		PUSH	D		
				3	CS		PUSH	B		
				4	DB		IN	PORT1B		Read A/D Value
				5	OS					(A) ← Vi
				6	21		LXI	H, 8300		Address memory
				7	00					locations for FILTR
				8	83					
				9	CD		CALL	FILTR		(A) ← Ei
MICROCOMPUTER TRAINING SYSTEM		A			70					(H) ← Ei
		B			82					(L) ← Vi
		C			CD		CALL	DWORD		Display Ei at left
		D			DI					and Vi
		E			02					
		F			3E		MVI	A, 06		Reset and disable
	8	2	4	0	06					A/D interrupt
				1	D3		OUT	CNT2		
				2	0F					
				3	C1		POP	B		
			4	D1		POP	D			
			5	E1		POP	H			
			6	F1		POP	PSW			
			7	FB		EI				
			8	C9		RET				
INTEGRATED COMPUTER SYSTEMS		A								
		B								
		C								
		D								
		E								
		F								
	8			0						
				1						
			2							
			3							
			4							
			5							
			6							
			7							
			8							

Figure 5-30b

A/D INPUT WITH FILTR - MAIN LOOP

		A	D	D	R	CODE			
CODING SHEET	8 25	0	3E			MVI	A,	01	Initial value for m
		1	01						in FILTR
	8 25	2	F3			DI			No interrupts while
		3	21			LXI	H,	8300	changing data
		4	00						for FILTR
		5	83						
		6	77			MOV	M,	A	(8300) ← m
		7	AF			XRA	A		} Clear 2 ^m Ei-1 (8301, 8302)
		8	23			INX	H		
		9	77			MOV	M,	A	
	A	23			INX	H			
MICROCOMPUTER TRAINING SYSTEM		B	77			MOV	M,	A	
		C	FB			EI			Enable A/D
	8 25	D	3E			MVI	A,	08	interrupt without
		E	08						resetting counter
		F	D3			OUT	PORT	2C	
	8 26	0	0E						
		1	DB			IN	PORT	0A	Test keyboard
		2	00						(FF if no key)
		3	3C			INR	A		
		4	CA			JZ	825D		Reenable A/D
INTEGRATED COMPUTER SYSTEMS		5	5D						until key pressed
		6	82						
		7	CD			CALL	ENTBY		Accept new m
		8	36						(must be
		9	03						1, 2, 3 or 4)
		A	7D			MOV	A,	L	(A) ← m
		B	C3			JMP	8252		Go to store m
		C	52						and clear 2 ^m Ei
		D	82						
		E							
	F								
	8	0							
	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								

Figure 5-30c

SUBROUTINE FILTR

A D D R CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

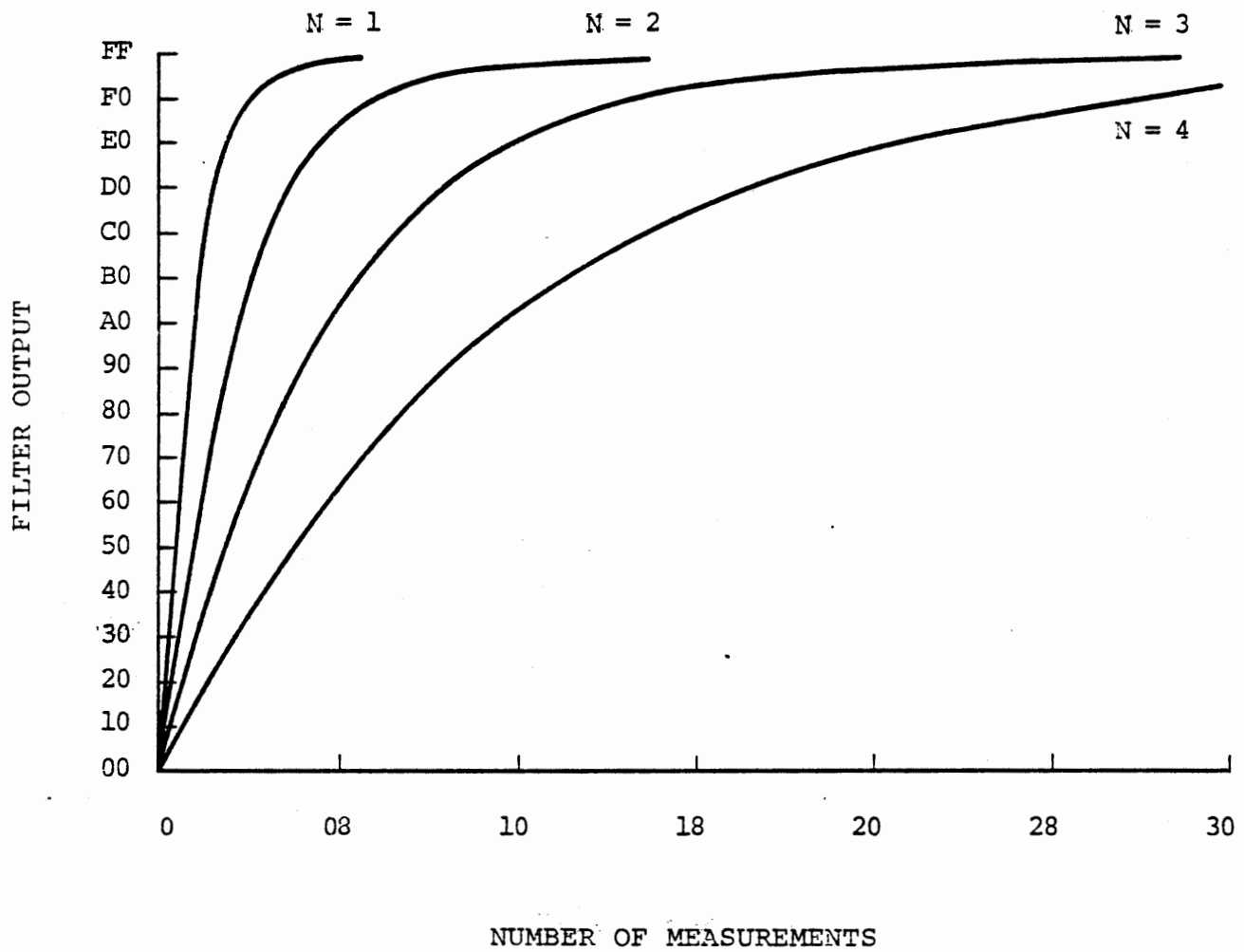
8	27	0	D5	PUSH	D			Save (DE)
		1	C5	PUSH	B			Save (BC)
		2	46	MOV	B, M			(B) ← m
		3	48	MOV	C, B			(C) ← m
		4	23	INX	H			Address $2^m E_i - 1$
		5	E5	PUSH	H			Save address
		6	5E	MOV	E, M			} (DE) ← $2^m E_i - 1$
		7	23	INX	H			
		8	56	MOV	D, M			} (HL) ← $2^m E_i - 1$
		9	6B	MOV	L, E			
		A	62	MOV	H, D			
827	B	29		DAD	H			} (HL) ← $2^{2m} E_i - 1$
	C	0D		DCR	C			
	D	C2		JNZ	827B			
	E	7B						
	F	82						
828	0	4F		MOV	C, A			(C) ← V_i
	1	7D		MOV	A, L			} (DE) ← (HL) - (DE) = $2^m(2^m - 1) E_i - 1$
	2	93		SUB	E			
	3	5F		MOV	E, A			
	4	7C		MOV	A, H			
	5	9A		SBB	D			
	6	57		MOV	D, A			} (HL) ← V_i
	7	69		MOV	L, C			
	8	26		MVI	H, 00			
	9	00						
	A	CD		CALL	SHFTN			Divide by 2^m
	B	A0						(DE) ← $(2^m - 1) E_i - 1$
	C	82						
	D	EB		XCHG				} (DE) ← $V_i + (2^m - 1) E_i - 1$ = $2^m E_i$
	E	19		DAD	D			
	F	EB		XCHG				
8	0			CONTINUED AT 8290				
	1							
	2			ENTER	(A)	=	V_i (new value)	
	3			(HL)		=	m (filter constant)	
	4			(HL) + 1		?	$2^m E_i - 1$	
	5			(HL) + 2		}		
	6			RETURN				
	7			(HL) + 3		=	(A) = (H) = E_i	
	8						(L) = V_i	

Figure 5-30d

A	D	D	R	CODE											
8	2	9	0	E3		X	T	H	L		(HL) ← Address $2^m E_i$				
			1	73		M	O	V		M, E	} Store $2^m E_i$				
			2	23		I	N	X		H					
			3	72		M	O	V		M, D					
			4	23		I	N	X		H	Address E_i				
			5	1B		D	C	X		D	To roundup only if				
			6	CD		C	A	L	L	S	H	F	T	N	Fractional part $> 1/2$
			7	A0											
			8	82											
			9	77		M	O	V		M, A	Store E_i				
			A	E1		P	O	P		H	(L) ← V_i				
			B	67		M	O	V		H, A	(H) ← E_i				
			C	C1		P	O	P		B	Restore BCDE				
			D	D1		P	O	P		D					
			E	C9		R	E	T			Exit				
			F	00		N	O	P							
8	2	A	0	48		M	O	V		C, B	SHFTN (C) ← M				
			1	AF		X	R	A		A	Loop - clear carry				
			2	7A		M	O	V		A, D	} Shift (DE) right to divide by 2				
			3	1F		R	A	R							
			4	57		M	O	V		D, A					
			5	7B		M	O	V		A, E					
			6	1F		R	A	R							
			7	5F		M	O	V		E, A					
			8	0D		D	C	R		C		Loop m times			
			9	C2		J	N	Z		82A1	to divide by 2^m				
			A	A1											
			B	82											
			C	D0		R	N	C			Exit if LSB = 0				
			D	13		I	N	X		D	Else roundup				
			E	7B		M	O	V		A, E	(A) ← less				
			F	C9		R	E	T			significant byte				
8			0												
			1												
			2												
			3												
			4												
			5												
			6												
			7												
			8												

Figure 5-30e

THIS PAGE INTENTIONALLY LEFT BLANK



FILTER RESPONSE FOR VARIOUS N

Figure 5-31

5.5.3 Filter Response

The behavior of a filter may be described in terms of frequency response, or as response to a step function. Each contains the same information, mathematically speaking, but one or the other is more convenient depending on the purpose or the structure of the filter. For a programmed digital filter, it is far easier, in general, to obtain the step function response, since only a two valued input is required. The response of FILTR to a full scale step input is shown in Figure 5-31. You can obtain similar data by a simple process of programmed calls to FILTR. Start with the memory locations for $2^{*nE(i)}$ cleared. Load register A with FF (or some other value), call FILTR, and display the result. Wait for a key, then repeat the load and call.

THIS PAGE INTENTIONALLY LEFT BLANK

5.6 Temperature Measurement

Two important devices are used for temperature measurement in conjunction with microprocessors: thermocouples and thermistors. A thermocouple is a junction of two dissimilar metals. When heated the junction develops a voltage which can be measured and converted to temperature. The thermocouple is highly precise, requires no calibration, and is extremely rugged. It has the disadvantage that the voltage generated is small, and no current may be allowed to flow in its circuit, because resistive voltage drop in the wire would mask the thermal voltage.

5.6.1 Thermistor Characteristics

A thermistor is a semiconductor device that appears as a variable resistance dependent on temperature. Figure 5-32 is a plot of resistance versus temperature for the thermistor supplied with your interface board. The manufacturer's data for this device is

Part No. RL2012-5506-120-D1

Resistance	10000 ohms at 25 degrees C	Temperature coefficient
	5506 ohms at 37.8 degrees C	4.84% per degree C
	251 ohms at 125 degrees C	at 25 degrees

The plots of Figure 5-32 were obtained by fitting these data with a curve generated numerically from:

$$R_{i+1} = R_i - (1 - C_i \Delta T)$$

$$C_{i+1} = f^{\Delta T} C_i$$

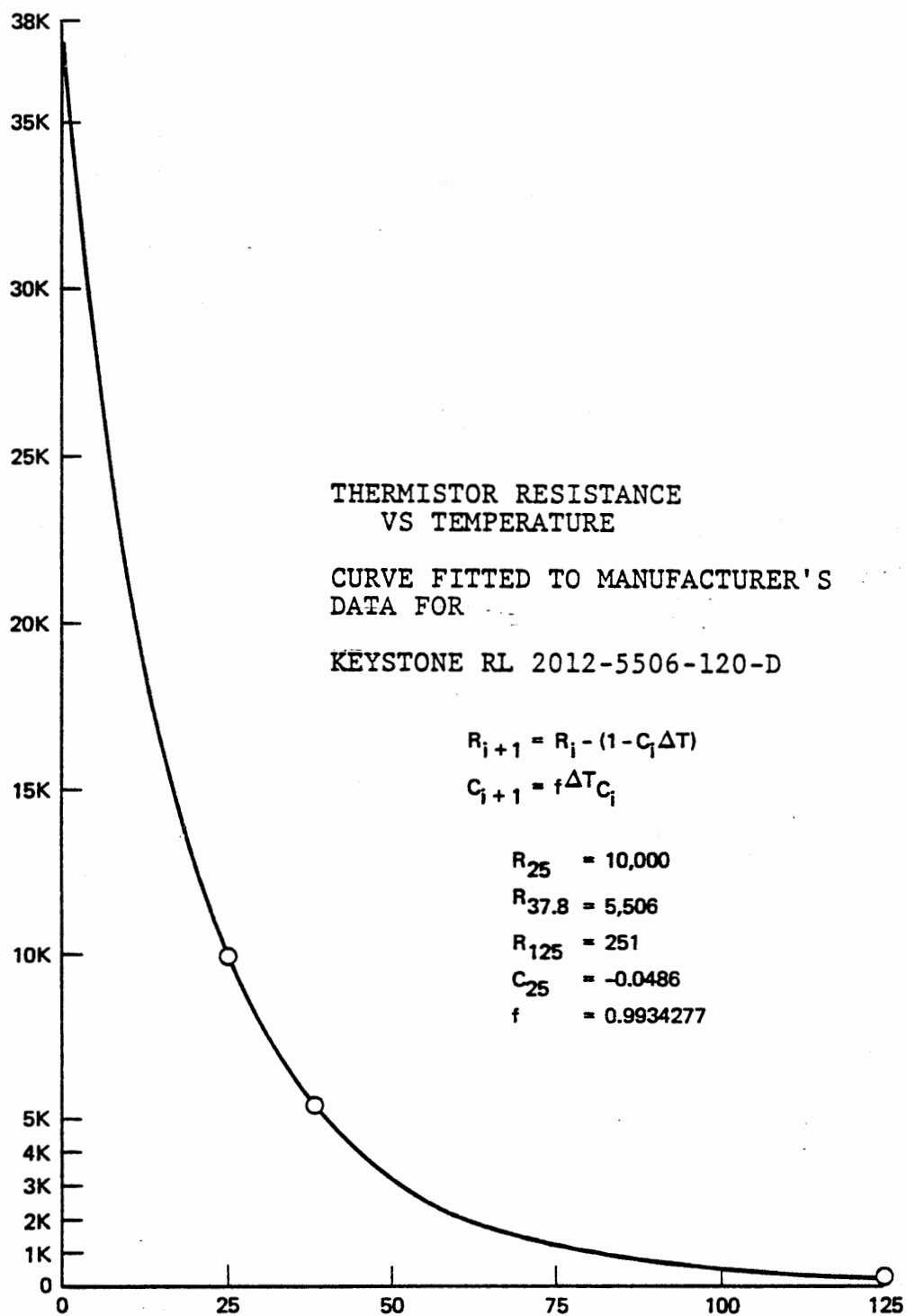


FIGURE 5-32a

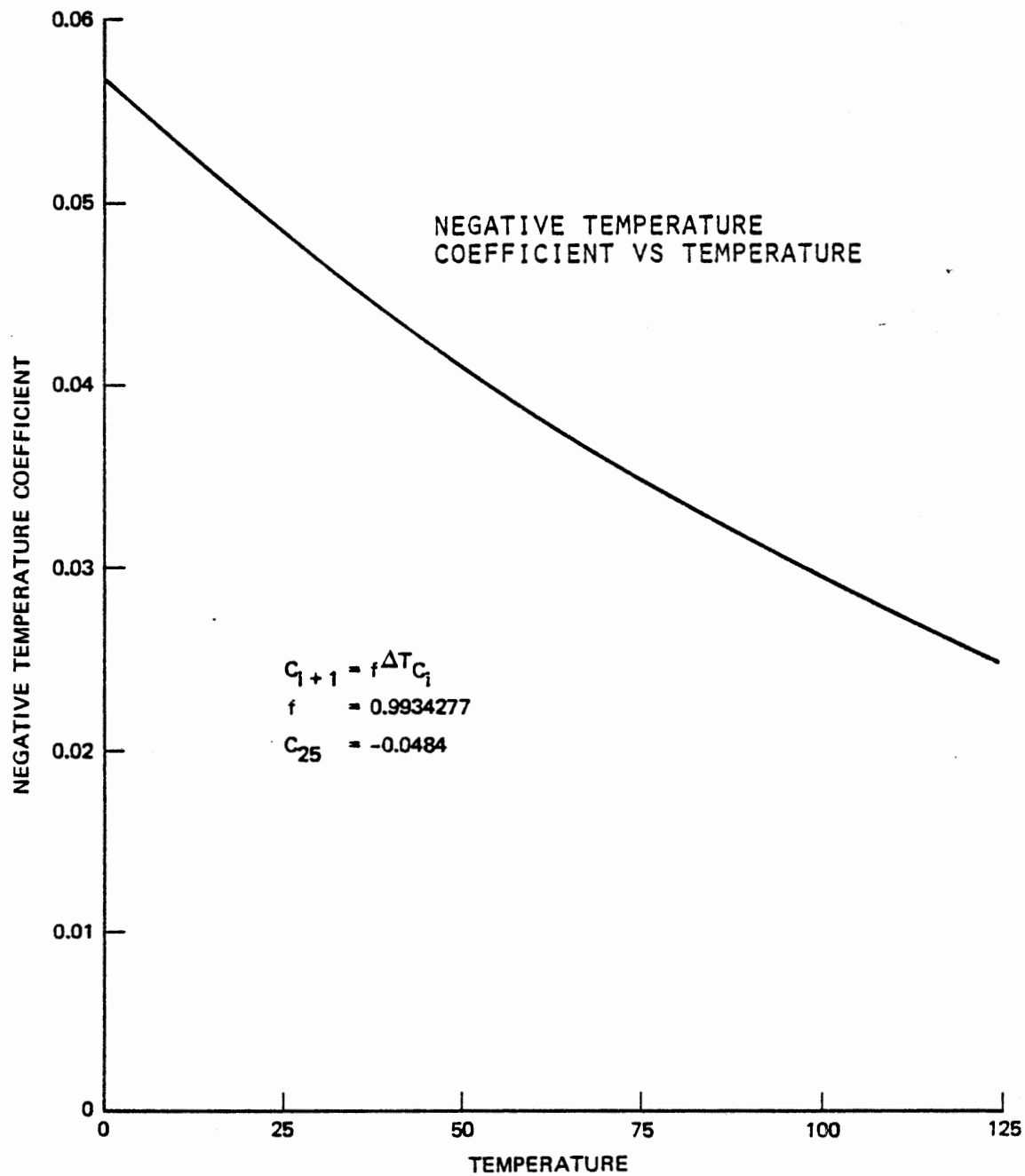
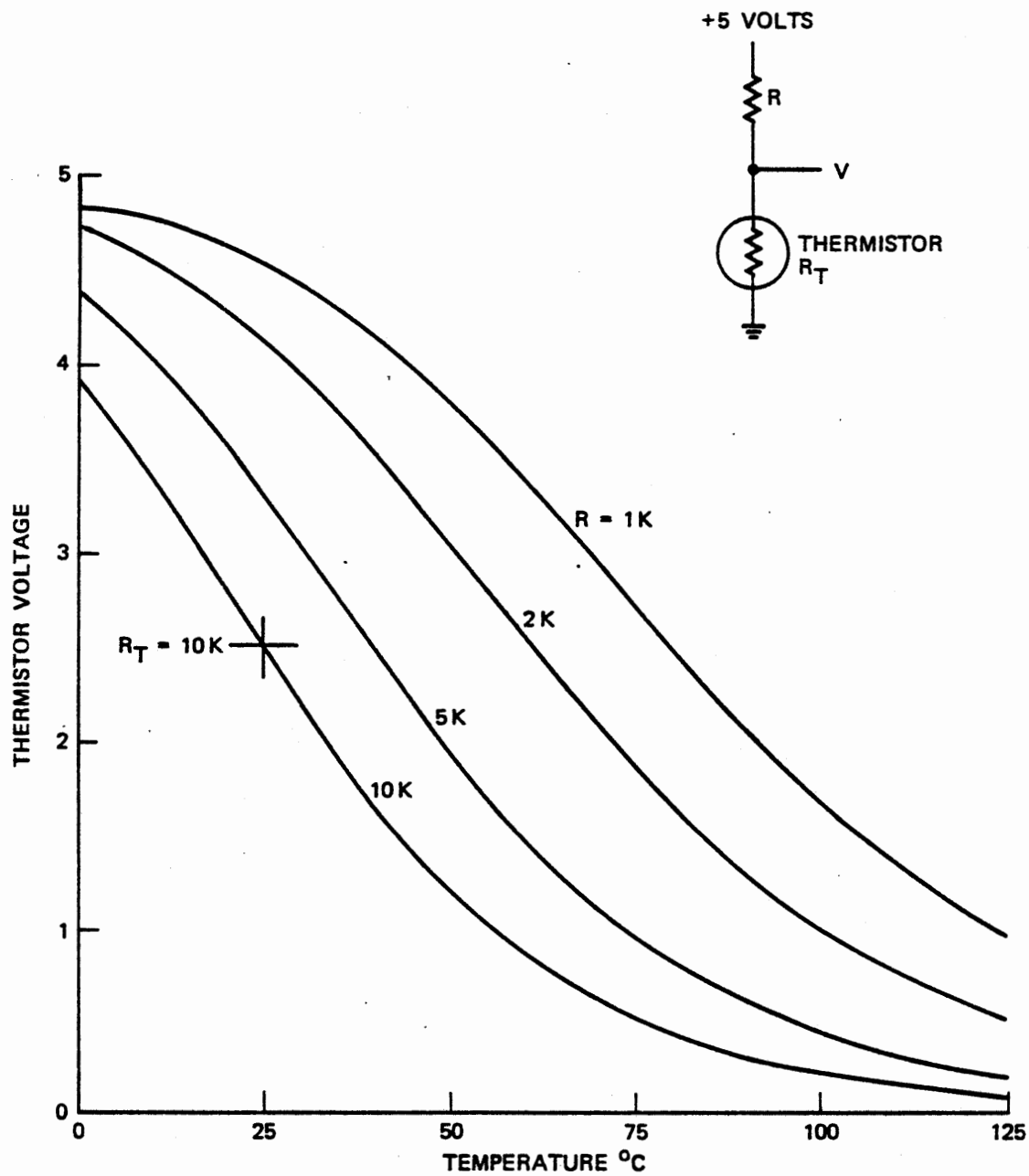


FIGURE 5-32b



THERMISTOR CONNECTION AND VOLTAGE PLOT
 FIGURE 5-33

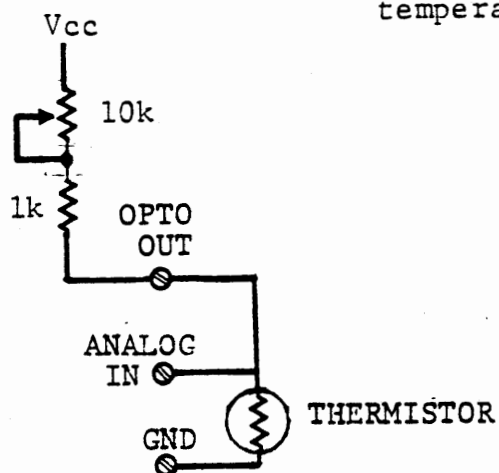
5.6.2 Thermistor Operation

When the thermistor is connected in a circuit such as shown in Figure 5-33, it gives a voltage which is close to linear over modest temperature ranges. The data can be linearized, and scaled from voltage to temperature, by a table lookup with linear interpolation. For the experiments in this section we will use the 10K ohm resistance data, but for high temperatures you might choose a lower resistance.

Note that at room temperature and below, the voltage will be beyond the 2.55 volt range of the D/A converter. We will adjust the ANALOG IN pot to divide the voltage by 2. This should be your first step. Connect +5 VOLTS to ANALOG IN, and use a digital voltmeter program such as Figure 5-26, 5-28 or 5-30 to display the measured voltage. Adjust the ANALOG IN pot to obtain a measured value of 2.50 volts (FA).

Because of its non-linearity and because it is subject to the uncertainties of semiconductor manufacturing, a thermistor must be calibrated. To do this adequately requires a laboratory thermometer and some means of heating the thermistor. This can be done by encapsulating the thermistor and its leads in epoxy resin and heating it in a water bath. If you do not have these facilities available, it is reasonably satisfactory to measure the resistance at a known room temperature and scale the manufacturer's data appropriately. The following procedure does the necessary scaling by a pot adjustment, assuming that you will use the fitted curve data.

Adjust OPTO SENSE pot
to obtain expected
voltage at room
temperature.



Temperature		Expected Voltage	A/D Input with 2:1 Attenuation
$^{\circ}\text{F}$	$^{\circ}\text{C}$		
65	18-33	2.910	92
66	18.89	2.876	90
67	19.44	2.842	8E
68	20.00	2.808	8C
69	20.56	2.773	8B
70	21.11	2.739	89
71	21.67	2.705	87
72	22.22	2.671	86
73	22.78	2.637	84
74	23.33	2.603	82
75	23.89	2.568	80
76	24.44	2.534	7E
77	25.00	2.500	7D

Expected Voltage At Room Temperature

FIGURE 5-34

5.6.3 Thermistor Input adjustment

Connect the thermistor as shown in Figure 5-34. Find the room temperature. (A household thermometer is sufficiently accurate for this.) Find the expected voltage from:

$$V_t = 2.50 + 0.0615 (25-T) \quad (\text{Celsius})$$

or
$$V_t = 2.50 + 0.0342 (77-T) \quad (\text{Fahrenheit})$$

or use the table of Figure 5-34. Still using a digital voltmeter program, adjust the OPTO SENSE pot to obtain the expected voltage. This procedure sets the pot to match the thermistor resistance at 25°C, thereby removing the principal uncontrolled variable of the thermistor. You can now heat or cool the thermistor and observe the voltage changing. If you have a thermometer, you can calibrate the thermistor, taking a series of voltage and temperature measurements.

5.6.4 Table Lookup and Interpolation

To convert a measured voltage to a temperature requires a table lookup and possibly some form of interpolation. Four approaches are available:

5.6.4.1 Method A

Store a complete table of temperature versus voltage. This is not unreasonable, since the 8 bit A/D input only requires 256 table entries. The temperature can be stored in binary and converted to decimal for display, or with two byte entries, it can be stored in decimal. This approach minimizes program complexity, is very fast, but is extravagant of memory.

5.6.4.2 Method B

Store a partial table, listing voltage and temperature at appropriate intervals. Find two (adjacent) points in the table, above and below the measured voltage, and do a linear interpolation between them. This requires the least storage, but requires multiplication and division for the interpolation, and since that would probably be done in binary, it also needs binary to decimal conversion for the display.

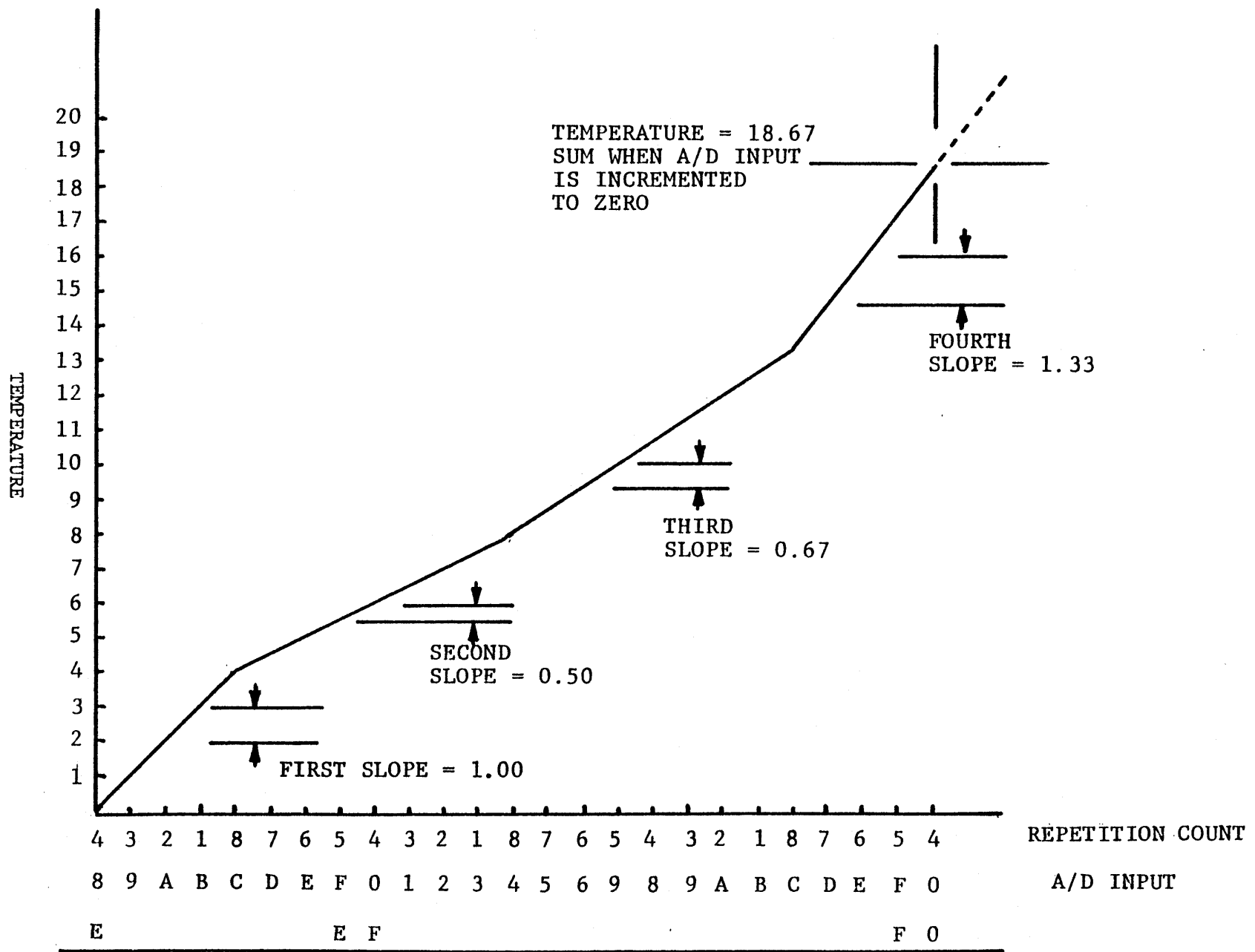
5.6.4.3 Method C

Store a table of voltage, temperature, and slope at appropriate intervals. Find the lowest table entry whose voltage is greater (hence temperature lower) than the measured value and multiply the slope by the difference between tabulated voltage and measured

voltage. Add this to the tabulated temperature. This avoids division, and can reasonably be done in decimal to avoid binary to decimal conversion. The multiplication can readily be done by successive addition, since the multiplier (the voltage difference) will always be a small number.

5.6.4.4 Method D

Store a table of slopes with ranges over which each slope applies, and perform a numerical integration from the start of the table to the measured voltage. The stored slopes need greater precision than with an interpolation, since errors accumulate, but the storage requirement is still less than any method except that of 5.6.4.2. The program is the simplest by far. This method is used in the following exercise.



TEMPERATURE CONVERSION BY INTEGRATION

FIG 5-35

5.6.5 Voltage to Temperature Conversion

Develop a subroutine and a data table to convert voltage to temperature and display both the input and the result. The integration technique discussed in Section 5.6.4.4 is to be used.

5.6.5.1 Data Table

Each table entry includes a slope (in decimal) and a count. The slope is repeatedly summed into the temperature while its count is decremented and the A/D input is incremented. When the A/D input reaches zero, the integration is complete. If the count is decremented to zero before the A/D input reaches zero, the next table entry is accessed and the process continues. The process is portrayed in Figure 5-35, for a hypothetical A/D input of E8. The slopes and temperatures shown are illustrative and not at all realistic. Figure 5-36 lists actual data for an ideal thermistor with the previously specified characteristics. A subroutine for the conversion by integration is defined in Section 5.6.5.2, and shown in Figure 5-37.

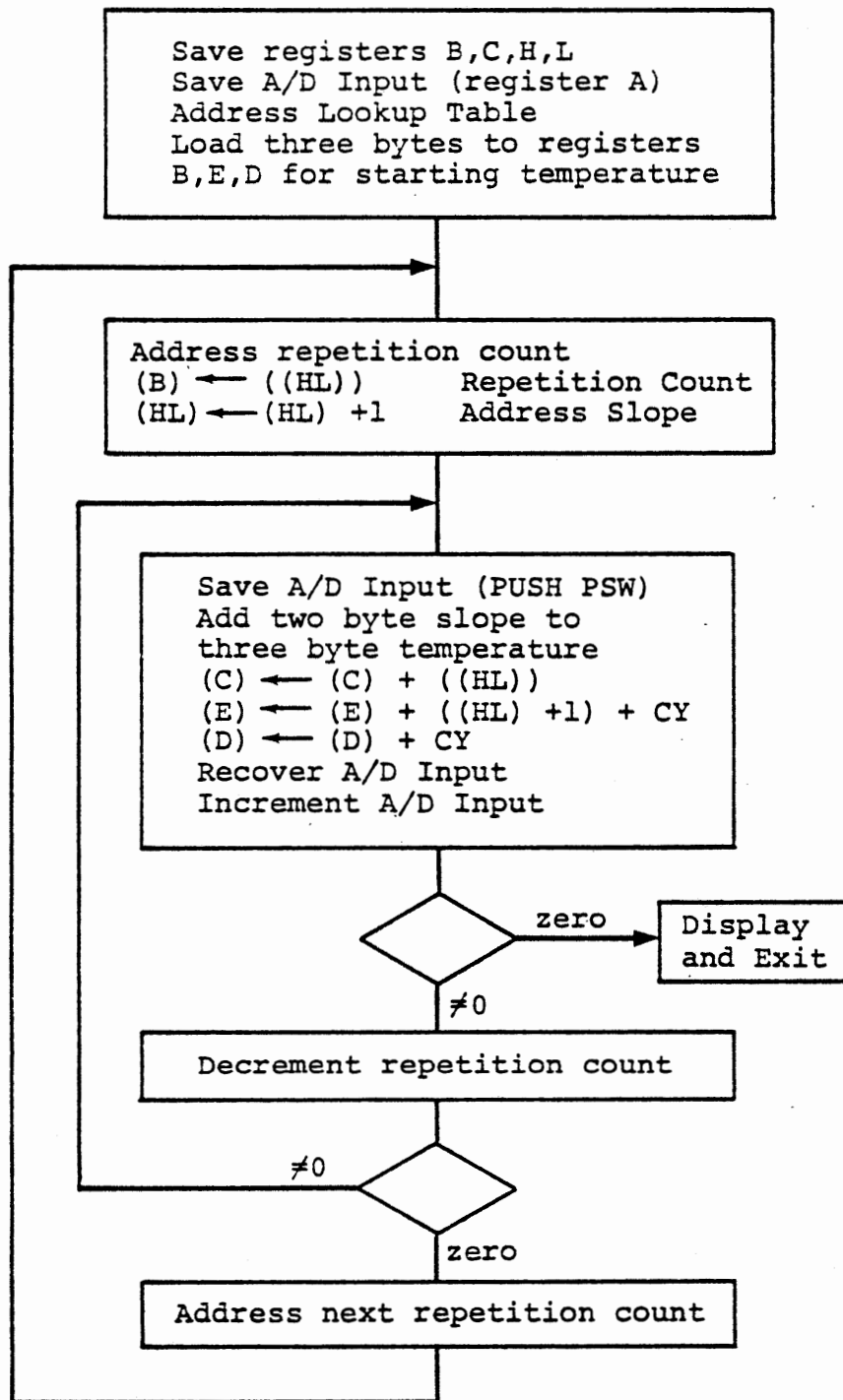
TEST DATA		TABLE DATA	
A/D Input	Temperature	Repetitions (hex)	Slope (decimal)
C4	0.600	4	0.405
C0	2.219	10	0.373
B0	8.188	10	0.341
A0	13.644	20	0.324
80	24.012	10	0.335
70	29.372	10	0.355
60	35.052	10	0.394
50	41.356	10	0.458
40	48.684	10	0.532
38	52.940	8	0.606
30	57.788	4	0.681
2C	60.512	4	0.743
28	63.484	4	0.820
24	66.764	4	0.918
20	70.436	4	1.046
1C	74.620	4	1.214
18	79.476	4	1.452
14	85.284	4	1.803
10	92.496	2	2.190
0E	96.876	2	2.524
0C	101.924	2	3.089
0A	108.102	1	3.573
09	111.675	1	4.065
08	115.740	1	4.697
07	120.437	1	5.537
06	125.974	1	6.700

THERMISTOR CALIBRATION DATA

FIGURE 5-36

Our calibration data do not extend to 2.55 volts (FF). The first meaningful point occurs at 0.60 ° C at 3.92 volts (C4). The integration procedure to be used demands that data be provided for all possible values, so we will start the process with a linear integration from -23.701 ° C at a slope of 0.405 for 64 repetitions. This generates 0.600 ° C at 3.92 volts and gives the correct slope from there to 2.219 ° C at 3.84 volts. Results down to slightly negative temperatures will be approximately correct. The first table entry is the starting temperature (in hundreds complement form) and the next entry provides the 0.405 ° C per 20 mv slope with 40H repetitions.

8310	99	
11	62	-23.701
12	97	
13	40	64 repetitions
14	05	0.405 ° C/20 mv
15	04	
16	10	16 repetitions
17	73	0.373 ° C/20 mv
18	03	
19	10	16 repetitions
1A	41	0.341 ° C/20 mv
1B	03	
etcetera		



TEMPERATURE LOOKUP BY INTEGRATION

Figure 5-37

5.6.5.2 Subroutine Temp

Enter with, (A) = measured voltage
 Return with (A) = measured voltage
 (DE) = temperature (decimal degrees and tenths)
 Registers B,C,H,L preserved.

Display A/D input in hexadecimal, temperature in decimal either as three bytes (xxx.xxx) or as two bytes rounded (xxx.x).

Data table, located at 8310-836F,

8310-02	
8313	Repetition count for first slope
8314-15	Slope, decimal, as x.xxx
8316	Next repetition count
8317-18	Next slope
etcetera	

Note that the flow diagram does not detail the display function, which is left to the student.

5.6.5.3 Test for TEMP

A simple test program is given in Figure 5-38a. Key in a hex value representing a voltage and observe the result displayed by TEMP. Try values from Figure 5-36 to be sure that the corresponding temperature is displayed.

TEST PROGRAM FOR TEMPERATURE LOOKUP 5-124

A D D R CODE										
CODING SHEET	8	200	F3	DI						
		1	CD	CALL	ENTBY					
		2	36							
		3	03							
		4	FB	EI						
		5	7D	MOV	A2L					
		6	CD	CALL	TEMP					
		7	B0							
		8	82							
		9	C3	JMP	8200					
MICROCOMPUTER TRAINING SYSTEM	A	00								
	B	82								
	C									
	D									
	E									
	F									
	8	0								
		1								
		2								
		3								
		4								
		5								
		6								
		7								
		8								
INTEGRATED COMPUTER SYSTEMS	A									
	B									
	C									
	D									
	E									
	F									
	8	0								
		1								
		2								
		3								
		4								
		5								

Figure 5-38a

TEMPERATURE LOOKUP AND DISPLAY

A D D R

CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	2B0	E5	PUSH	H			
	1	C5	PUSH	B			
	2	F5	PUSH	PSW			
	3	21	LXI	H, 8310			
	4	10					
	5	83					
	6	46	MOV	B, M	} Copy starting value into B, E, D		
	7	23	INX	H			
	8	5E	MOV	E, M			
	9	23	INX	H			
	A	56	MOV	D, M			
82B	B	23	INX	H	} Loop - address count (C) ← repetitions		
	C	4E	MOV	C, M			
	D	23	INX	H		} Address slope	
82B	E	F5	PUSH	PSW	} Loop - save A/D		
	F	7E	MOV	A, M			
82C	0	80	ADD	B	} Add low byte of slope to low byte of temperature		
	1	27	DAA				
	2	47	MOV	B, A			
	3	23	INX	H	} Address high byte of slope and add to second byte of temperature		
	4	7E	MOV	A, M			
	5	8B	ADC	E			
	6	27	DAA				
	7	5F	MOV	E, A			
	8	2B	DCX	H	} Address low byte		
	9	3E	MVI	A, 00			
	A	00			} Add carry to high byte of temperature		
	B	8A	ADC	D			
	C	27	DAA				
	D	57	MOV	D, A			
	E	F1	POP	PSW	} (A) ← A/D input		
	F	3C	INR	A		} Increment input	
8	0						
	1						
	2						
	3						
	4						
	5						
	6						
	7						
	8						

Figure 5-38b

TEMPERATURE LOOKUP AND DISPLAY 5-126

	A	D	D	R	CODE															
CODING SHEET	8	2	D	0	CA	JZ		82	DB										Exit when A/D	
				1	DB														input is incremented	
				2	82														to zero	
				3	0D	DCR	C												Decrement repetitions	
				4	C2	JNZ	82	BE											Loop to add slope	
				5	BE														until all repetitions	
				6	82														of this slope done	
				7	23	INX	H												Address high byte	
				8	C3	JMP	82	BB											Loop to address	
				9	BB														next slope	
MICROCOMPUTER TRAINING SYSTEM	A			82																
		82	D	B	EB	XCHG													Exit and display	
			C	F	I	POP	PSW												(ST) ← low byte	
			D	4	F	MOV	C, A												of temperature and	
			E	C	5	PUSH	B												original A/D input	
			F	C	D	CALL	DWORD												Display high two	
		8	2	E	0	D	I												byte of temperature	
MICROCOMPUTER TRAINING SYSTEM				1	02														at left	
				2	E3	XTHL														
				3	11	LXI	D, 83FF													
				4	FF															
				5	83															
				6	C	D	CALL	DWD2												Display voltage at
				7	D	4													right, and low	
				8	0	2													byte of temperature	
				9	E	B	XCHG													
	INTEGRATED COMPUTER SYSTEMS	A			2	B	DCX	H												(HL) ← address
				B	6	ORA	M												whole degrees and	
				C	7	7	MOV	M, A											enter decimal point	
				D	7	B	MOV	A, E											(A) ← voltage	
				E	D	1	POP	D											(DE) ← temperature	
				F	C	1	POP	B											Restore other	
		8	2	F	0	E	1	POP	H										registers	
				1	C	9	RET													
				2																
				3																
			4																	
			5																	
			6																	
			7																	
			8																	

Figure 5-38c

		A	D	D	R	CODE															
CODING SHEET	8				0																
	8 3 3	1			0 4															2F-2C	
		2			8 1																0.681
		3			0 6																
		4			0 4																2B-28
		5			4 3																0.743
		6			0 7																
		7			0 4																27-24
		8			2 0																0.820
		9			0 8																
		A			0 4																23-20
		B			1 8																0.918
		C			0 9																
		D			0 4																1F-1C
		E			4 6																1.046
		F			1 0																
MICROCOMPUTER TRAINING SYSTEM	8 3 4	0			0 4															1B-18	
		1			1 4																1.214
		2			1 2																
		3			0 4																17-14
		4			5 2																1.452
		5			1 4																
		6			0 4																13-10
		7			0 3																1.803
		8			1 8																
		9			0 2																0F-0E
		A			9 0																2.190
		B			2 1																
		C			0 2																0D-0C
		D			2 4																2.524
		E			2 5																
		F			0 2																0B-0A
INTEGRATED COMPUTER SYSTEMS	8 3 5	0			8 9															3.089	
		1			3 0																
		2																			
		3																			
		4																			
		5																			
		6																			
		7																			
	8																				

Figure 5-38e

		A	D	D	R	CODE																
CODING SHEET	8	0																				
		1																				
		835	2	0	1															09		
			3	7	3																3.573	
			4	3	5																	
			5	0	1																08	
			6	6	5																4.065	
			7	4	0																	
			8	0	1																07	
			9	9	7																4.697	
MICROCOMPUTER TRAINING SYSTEM		A	4	6																		
		B	0	1																	08	
		C	3	7																	5.537	
		D	5	5																		
		E	0	1																	07	
		F	0	0																	6.700	
		836	0	6	7																	
			1	F	F																06	125.974°
			2	0	0																upper limit for	
			3	0	0																thermistor	
INTEGRATED COMPUTER SYSTEMS		4																				
		5																				
		6																				
		7																				
		8																				
		0																				
		1																				
		2																				
		3																				
		4																				
	5																					
	6																					
	7																					
	8																					

Figure 5-38f

5.6.6 Thermometer Program

Exercise

Develop a program to read the thermistor voltage and convert the measurement to decimal degrees by table lookup with interpolation.

In many systems it is more appropriate to take measurements at regular intervals than as rapidly as possible. This is particularly true with temperatures which typically change slowly and where rate of change may be of interest. In this program a timed interrupt will decrement a time counter and at one second intervals, it will increment a seconds counter. At each interrupt (20 milliseconds) it will reset the A/D converter and enable the A/D interrupt. Thus, a measurement of temperature will be made every 20 milliseconds, and a timer will be available to the main program.

RST 6 services the A/D interrupt. It will read the input from port 1B and call FILTR, the subroutine of Section 5.5.2, to obtain a filtered value for the input voltage. Since a measurement is wanted at 20 millisecond intervals, RST 6 service disables the A/D interrupt.

The main program loop compares the interval counter with an interval obtained by keyboard input. When the count has reached the desired interval, it restarts the counter, loads the current estimate of voltage and calls TEMP (the subroutine of Section 5.6.5.2) to display the temperature. It also tests the keyboard and calls ENTBY if a key is pressed to enter a new interval.

The subroutine developed in Section 5.3.5 will convert the measured voltage to temperature by table lookup and interpolation. Both temperature and voltage are displayed. The low byte of temperature is not significant, since the absolute accuracy is not better than half a degree. So the display function should be modified to display only the two higher bytes. Rounding of the three byte result is easily achieved by making the initial value -23.651 instead of -23.701 (976.349 in hundreds complement).

Memory assignment for the program are:

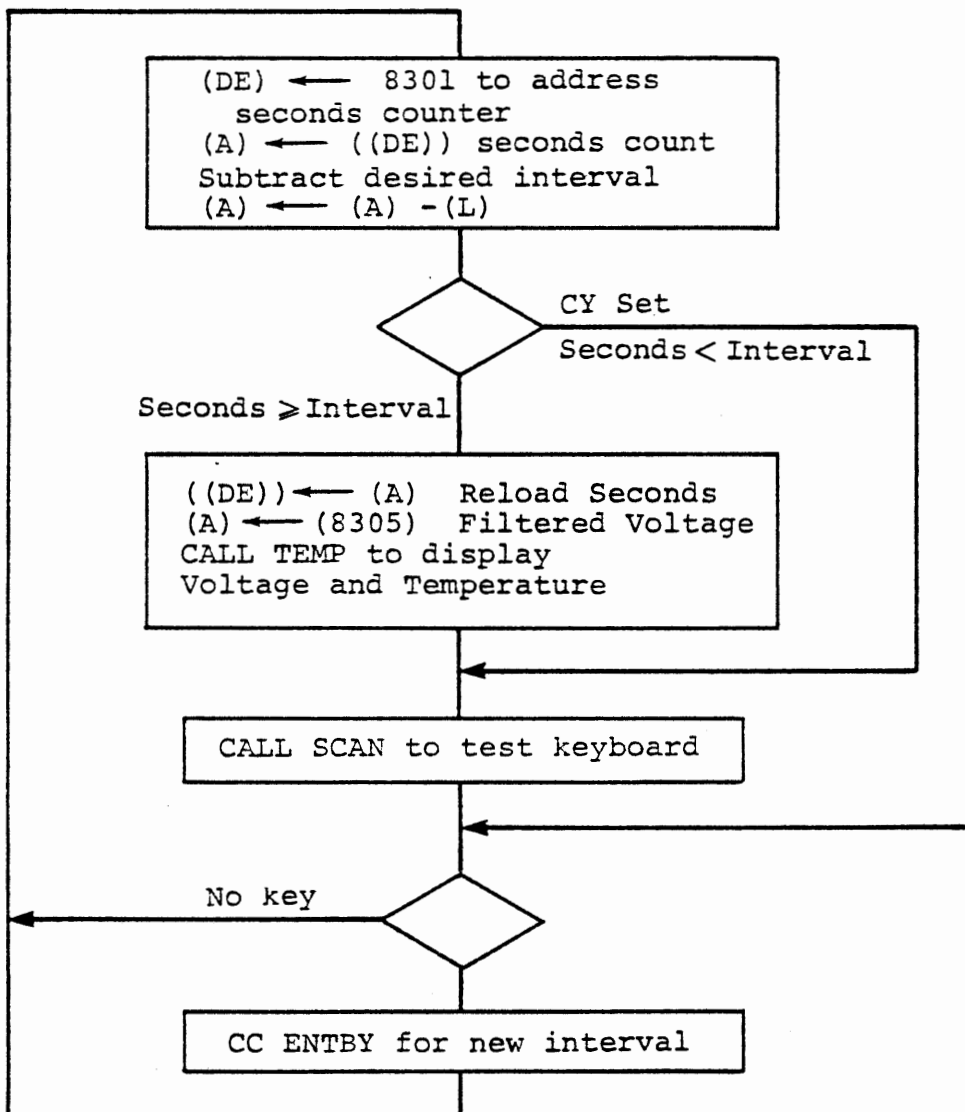
8200 - 8227	Main - Initialize
8228 - 8257	Interrupt Service
8258 - 826F	Main Loop
8270 - 82AF	Subroutine FILTR
82B0 - 82FF	Subroutine TEMP
8300	One second counter
8301	Seconds counter
8302	n for FILTR
8303-4	$2^n E_i$ for FILTR
8305	E_i
8310 - 836F	Table for TEMP

Note that 8300 - 8305 must be initialized at program loading, along with the table for TEMP, even though they contain variables.

MAIN - INITIALIZE

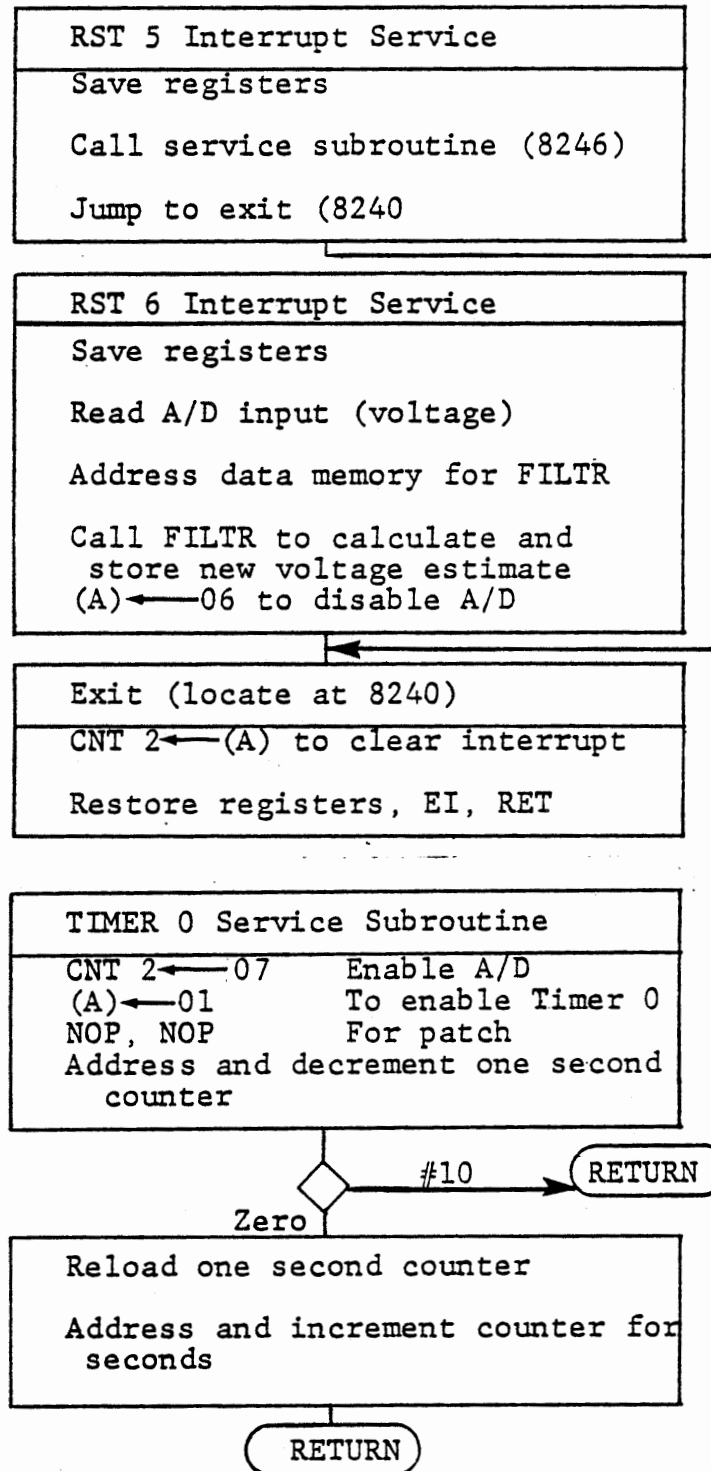
Program Ports - Port 1B In
 Program Timer 0 and load
 for 20 millisecond interrupt
 Program Timer 2 and load
 for $\div 16$ for A/D clock
 Set Port 1C0 for automatic A/D
 RST5 to enable interrupt
 Set CY and jump into main
 loop at CC ENTBY

MAIN LOOP



THERMOMETER - MAIN

Figure 5-39a



THERMOMETER - INTERRUPT SERVICE
 FIGURE 5-39b

THERMOMETER - INITIALIZE

5-134

A D D R.		CODE					
CODING SHEET	8 20 0	3E	MVI	A,	82		Program Ports
							Port 1B In for A/D
		1	82				
		2	D3	OUT	CNT1		
		3	07				
		4	3E	MVI	A,	92	
		5	92				
		6	D3	OUT	CNT2		
MICROCOMPUTER TRAINING SYSTEM		7	0F				
		8	3E	MVI	A,	24	Program Timer 0
		9	24				High byte
		A	D3	OUT	TIMCT		Mode 2
		B	17				Binary
		C	3E	MVI	A,	94	Program Timer 2
		D	94				Low byte
		E	D3	OUT	TIMCT		Mode 2
MICROCOMPUTER TRAINING SYSTEM		F	17				Binary
	8 21 0	3E	MVI	A,	A0		Load Timer 0
		1	A0				for 20 millisecond
		2	D3	OUT	TIM0		interrupt
		3	14				
		4	3E	MVI	A,	20	Load Timer 2
		5	20				for 16 μ second
		6	D3	OUT	TIM2		clock to D/A
INTEGRATED COMPUTER SYSTEMS		7	16				
		8	3E	MVI	A,	01	Set Port 1C0 high
		9	01				for automatic
		A	D3	OUT	CNT1		A/D conversion
		B	07				
		C	EF	RSTS			Enable Timer 0
		D	37	STC			
		E	C3	JMP	826A		Jump to CC ENTBY
INTEGRATED COMPUTER SYSTEMS		F	6A				in main loop
	8 22 0	82					to accept time
		1					interval
		2					
		3					
		4					
		5					
		6					
	7						
	8						

Figure 5-40a

THERMOMETER - INTERRUPT SERVICE

		A	D	D	R	CODE				
CODING SHEET	8	0								
		1								
		2								
		3								
		4								
		5								
		6								
		7								
MICROCOMPUTER TRAINING SYSTEM	822	8	FS			PUSH	PSW		RSTS - Timer 0	
		9	ES			PUSH	H			
		A	CD			CALL	8246		Service Timer 0	
		B	46							
		C	82							
		D	C3			JMP	EXIT		Exit	
		E	40							
		F	82							
	MICROCOMPUTER TRAINING SYSTEM	823	0	FS			PUSH	PSW		RST6 - A/D
			1	ES			PUSH	H		
		2	DB			IN	PORT1B		Read voltage	
		3	05							
		4	21			LXI	H, 8302		Data address. for FILTR	
		5	02							
		6	83							
		7	CD			CALL	FILTR		Calculate and store new voltage estimate	
		8	70							
		9	82							
INTEGRATED COMPUTER SYSTEMS	A	3E				MVI	A, 06		To disable A/D interrupt	
	B	06								
	C	C3				JMP	EXIT			
	D	40								
	E	82								
	F	00								
	8	0								
		1								
	2									
	3									
	4									
	5									
	6									
	7									
	8									

Figure 5-40b

INTERRUPT SERVICE - EXIT, TIMER 0

	A	D	D	R	CODE						
CODING SHEET	8	2	4	0	D3		OUT	CNT2			EXIT
					0F						
					E1		POP	H			
					F1		POP	PSW			
					FB		EI				
MICROCOMPUTER TRAINING SYSTEM					C9		RET				
	8	2	4	6	3E		MVI	A, 07			TIMER 0 SERVICE
					07						
					D3		OUT	CNT2			Reset and enable
					0F						A/D comparator
					3E		MVI	A, 01			To reenable
					01						Timer 0 interrupt
					00		NOP				Room for patch
					00		NOP				
					21		LXI	H, 8300			Address and
					00						decrement one
		8	2	5	0	83					second counter
						35		DCR	M		
						C8		RNZ			
						36		MVI	M, 32		Reload one second
					32						
					23		INX	H		Address and	
					34		INR	M		increment seconds	
					C9		RET				
INTEGRATED COMPUTER SYSTEMS											

Figure 5-40c

THERMOMETER - MAIN LOOP

	A	D	D	R	CODE							
CODING SHEET	8				0							
					1							
					2							
					3							
					4							
					5							
					6							
					7							
MICROCOMPUTER TRAINING SYSTEM	825	8			11		LXI	D	8301		Address seconds counter	
		9			01							
		A			83							
		B			1A		LDAX	D			(A) ← seconds count	
		C			95		SUB	L			Compare interval	
		D			DA		JC		8267		If count less than interval go to test keyboard	
		E			67							
		F			82							
		826	0			12		STAX	D		Restart seconds count	
			1			3A		LDA		8305	(A) ← Filtered input voltage stored by FILTR	
INTEGRATED COMPUTER SYSTEMS		2			05							
		3			83							
		4			CD		CALL		TEMP		Convert to temperature and display	
		5			B0							
		6			82							
		826	7			CD		CALL		SCAN	Test keyboard	
			8			57						
			9			02						
		826	A			DC		CC		ENTBY	If key pressed Accept time interval (one byte, seconds) for temperature update.	
			B			36						
		C			03							
		D			C3		JMP		8258			
		E			58							
		F			82							
INTEGRATED COMPUTER SYSTEMS	8				0							
					1		REQUIRES		SUBROUTINES			
					2		FILTR	AT	8270 - 82AF			
					3		TEMP	AT	82B0 - 82FF			
					4		INITIAL DATA		8300 - 8305			
					5		TEMP TABLE		8310 - 836F			
					6							
					7							
				8								

Figure 5-40d

SUBROUTINE FILTER

A	D	D	R	CODE					
8	27	0		DS	PUSH	D		Save (DE)	
		1		CS	PUSH	B		Save (BC)	
		2		46	MOV	B, M		(B) ← M	
		3		48	MOV	C, B		(C) ← M	
		4		23	INX	H		Address $2^m E_i - 1$	
		5		ES	PUSH	H		Save address	
		6		5E	MOV	E, M		}	
		7		23	INX	H			(DE) ← $2^m E_i - 1$
		8		56	MOV	D, M		}	
		9		6B	MOV	L, E			(HL) ← $2^m E_i - 1$
		A		62	MOV	H, D			
8	27	B		29	DAD	H		}	
		C		0D	DCR	C			
		D		C2	JNZ	827B			(HL) ← $2^{2^m} E_i - 1$
		E		7B					
		F		82					
8	28	0		4F	MOV	C, A		(C) ← V_i	
		1		7D	MOV	A, L		}	
		2		93	SUB	E			(DE) ← (HL) - (DE)
		3		5F	MOV	E, A		}	
		4		7C	MOV	A, H			= $2^m(2^m - 1) E_i - 1$
		5		9A	SBB	D		}	
		6		57	MOV	D, A			
		7		69	MOV	L, C		}	
		8		26	MVI	H, 00			(HL) ← V_i
		9		00					
		A		CD	CALL	SHIFTN		Divide by 2^m	
		B		A0				(DE) ← $(2^m - 1) E_i - 1$	
		C		82					
		D		EB	XCHG			}	
		E		19	DAD	D			(DE) ← $V_i + (2^m - 1) E_i - 1$
		F		EB	XCHG			= $2^m E_i$	
8		0			CONTINUED AT 8290				
		1							
		2			ENTER (A) = V_i (new value)				
		3			((HL)) = M (filter constant)				
		4			((HL) + 1) = $2^m E_i - 1$				
		5			((HL) + 2)				
		6			RETURN				
		7			((HL) + 3) = (A) = (H) = E_i				
		8			(L) = V_i				

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

Figure 5-40e

FILTR continued and SHFTN

	A	D	D	R	CODE												
CODING SHEET	8	2	9	0	E3		X	T	H	L					(H) ← Address $2^m E_i$		
					73		M	O	V		M	,	E				
					23		I	N	X		H				} Store $2^m E_i$		
					72		M	O	V		M	,	D				
					23		I	N	X		H				Address E_i		
					1B		D	C	X		D				To roundup only $1/2$		
					CD		C	A	L	L		S	H	F	T	N	Fractional part $> 1/2$
					A0												
					82												
					77		M	O	V		M	,	A			Store E_i	
MICROCOMPUTER TRAINING SYSTEM	A				E1		P	O	P		H				(L) ← V_i		
	B				67		M	O	V		H	,	A		(H) ← E_i		
	C				C1		P	O	P		B				Restore BCDE		
	D				D1		P	O	P		D						
	E				C9		R	E	T						Exit		
	F				00		N	O	P								
	8	2	A	0	48		M	O	V		C	,	B		SHFTN (C) ← m		
					AF		X	R	A		A				Loop - clear carry		
					7A		M	O	V		A	,	D				
					1F		R	A	R						Shift (DE) right		
				57		M	O	V		D	,	A		to divide by 2			
				7B		M	O	V		A	,	E					
				1F		R	A	R									
				5F		M	O	V		E	,	A					
				0D		D	C	R		C				Loop m times			
				C2		J	N	Z		82A1				to divide by 2^m			
INTEGRATED COMPUTER SYSTEMS	A				A1												
	B				82												
	C				D0		R	N	C						Exit if LSB = 0		
	D				13		I	N	X		D				Else roundup		
	E				7B		M	O	V		A	,	E		(A) ← less		
	F				C9		R	E	T						significant byte		
	8				0												
					1												
					2												
					3												
				4													
				5													
				6													
				7													
				8													

Figure 5-40f

TEMPERATURE LOOKUP AND DISPLAY 5-140

	A	D	D	R	CODE										
CODING SHEET	8	2	B	0	E5		P	U	S	H	H				
				1	C5		P	U	S	H	B				
				2	F5		P	U	S	H	P	S	W		
				3	21		L	X	I	H	,	8	3	1	0
				4	10										
				5	83										
				6	46		M	O	V	B	,	M			
				7	23		I	N	X	H					
				8	5E		M	O	V	E	,	M			
				9	23		I	N	X	H					
MICROCOMPUTER TRAINING SYSTEM	A			56		M	O	V	D	,	M				
	8	2	B	B	23		I	N	X	H					
				C	4E		M	O	V	C	,	M			
				D	23		I	N	X	H					
	8	2	B	E	F5		P	U	S	H	P	S	W		
				F	7E		M	O	V	A	,	M			
	8	2	C	0	80		A	D	D	B					
				1	27		D	A	A						
				2	47		M	O	V	B	,	A			
				3	23		I	N	X	H					
INTEGRATED COMPUTER SYSTEMS				4	7E		M	O	V	A	,	M			
				5	8B		A	D	C	E					
				6	27		D	A	A						
				7	5F		M	O	V	E	,	A			
				8	2B		D	C	X	H					
				9	3E		M	V	I	A	,	0	0		
				A	00										
				B	8A		A	D	C	D					
				C	27		D	A	A						
				D	57		M	O	V	D	,	A			
			E	F1		P	O	P	P	S	W				
			F	3C		I	N	R	A						
	8		0												
			1												
			2												
			3												
			4												
			5												
			6												
			7												
			8												

} Copy starting value into B, E, D

} Loop - address count (C) ← repetitions Address slope

} Loop - save A/D Add low byte of slope to low byte of temperature

} Address high byte of slope and add to second byte of temperature

} Address low byte Add carry to high byte of temperature

} (A) ← A/D input Increment input

Figure 5-40g

TEMPERATURE WITH TWO BYTE DISPLAY cont'd 5-141

A D D R		CODE								
CODING SHEET	8	2D	0	CA		JZ		82DB	Exit when A/D	
			1	DB					input is incremented	
			2	82					to zero	
			3	0D		DCR	C		Decrement repetitions	
			4	C2		JNZ		82BE	Loop to add slope	
			5	BE					until all repetitions	
			6	82					of this slope done	
			7	23		INX	H		Address high byte	
			8	C3		JMP		82BB	Loop to address	
			9	BB					next slope	
MICROCOMPUTER TRAINING SYSTEM	A	82								
		82D	B	EB		XCHG			Exit and display	
			C	CD		CALL		DWORD	Display temperature	
			D	DI						
			E	02						
			F	F1		POP	PSW		(A) ← voltage	
		8	2E	0	CD		CALL		DBYTE	Display
				1	95					(C) ← voltage
				2	02					(A) ← 80
				3	EB		XCHG			(DE) ← temperature
MICROCOMPUTER SYSTEMS			4	2E		MVI	L, FA		Address whole	
			5	FA					degrees	
			6	B6		ORA	M		Insert decimal	
			7	77		MOV	M, A		point	
			8	79		MOV	A, C		(A) ← voltage	
			9	C1		POP	B			
			A	E1		POP	H			
			B	C9		RET				
			C							
			D							
INTEGRATED COMPUTER SYSTEMS			E							
			F							
		8		0						
				1						
				2						
				3						
				4						
				5						

Figure 5-40h

THERMOMETER - PRELOAD VARIABLE DATA

5-142

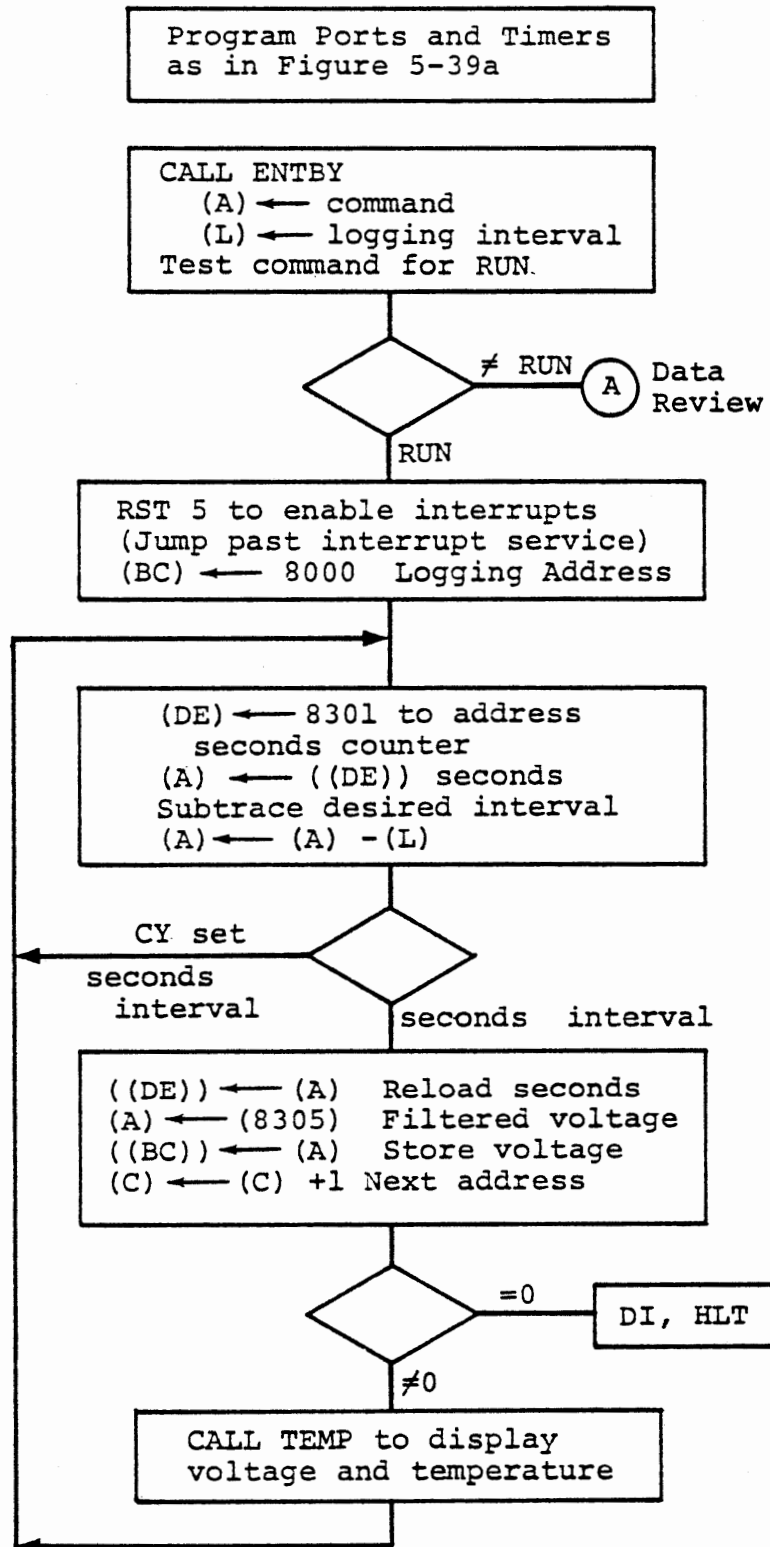
	A	D	D	R	CODE															
CODING SHEET	8	3	0	0	3 2														One second count	
					0 0														Seconds count	
					0 4														n for FILTER	
					0 0														} 2 nd EI	
					0 0															
						0 0														EI
MICROCOMPUTER TRAINING SYSTEM	8	3	1	0		TEMPERATURE TABLE														
						SEE FIGURE 5-38 d, e, f														
INTEGRATED COMPUTER SYSTEMS	8																			

Figure 5-40i

5.6.7 Data Logging

OPTIONAL EXERCISE

Modify the thermometer program to make a data logger. This will record in memory a series of measurements for later review. Your MTS must be fitted with 1K bytes of memory for this exercise. The design of subroutines FILTR and TEMP makes the revision very simple. Before CALL TEMP in the main loop, increment a data address and copy the filtered voltage to that location. Figure 5-41 shows the data logging program. To fit the program into the same space, we abandon the keyboard test while running, and simply call ENTBY once as part of initialization to obtain an interval. Register pair BC is available for the data logging address. Note the virtue of having subroutines save registers.



LOGGING THERMOMETER - MAIN

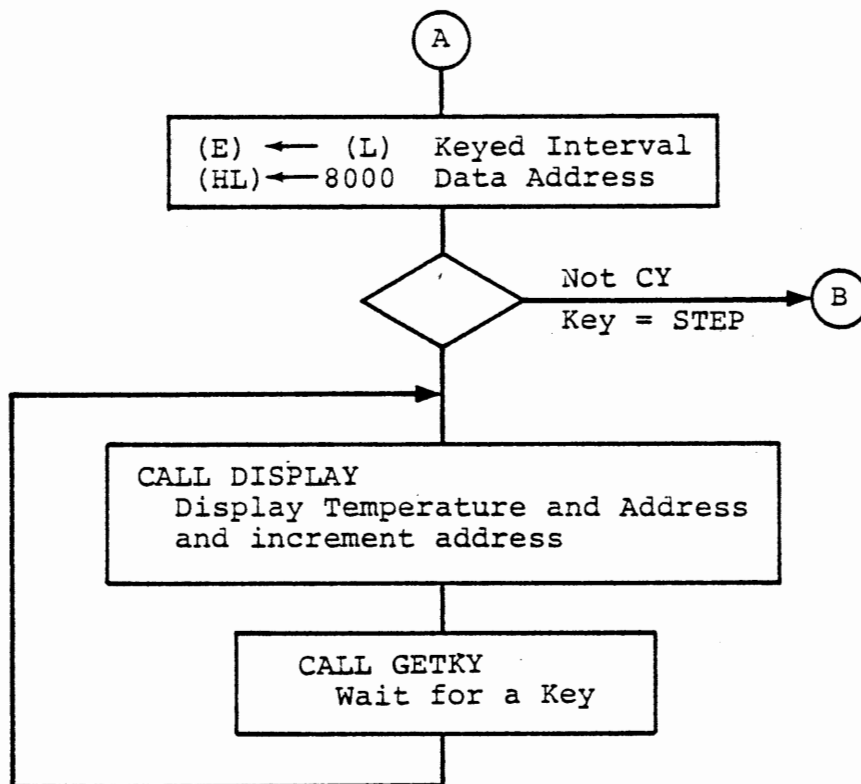
Figure 5-41

After CALL ENTBY in the initialization module, we will test the command for any of three keys:

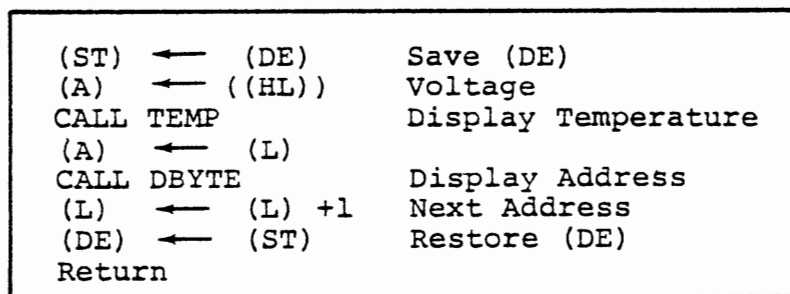
RUN	Start data logging, at intervals given (in seconds)
NEXT	Review the stored data, showing the temperature at each point in succession when NEXT is pressed
STEP	Replay the stored data as an output to the D/A converter, using the time interval entered with the command.

NEXT and STEP jump to the modules shown in Figures 5-42 and 5-43. Replay is interesting because it allows several hours of data to be displayed on a scope or voltmeter in a much shorter time. For instance, if the interval entered for data logging was 3C (decimal 60), RUN, the recording interval is one minute, allowing four hours of data to be recorded. Then a replay of the logged data with an interval of 01 will play the data back in four minutes. Greater speedup can be obtained by altering the program constants used for loading timer 0 and the "one second" counter. The table of Figure 5-44 shows the constants needed for various recording and playback intervals.

Temperature Review



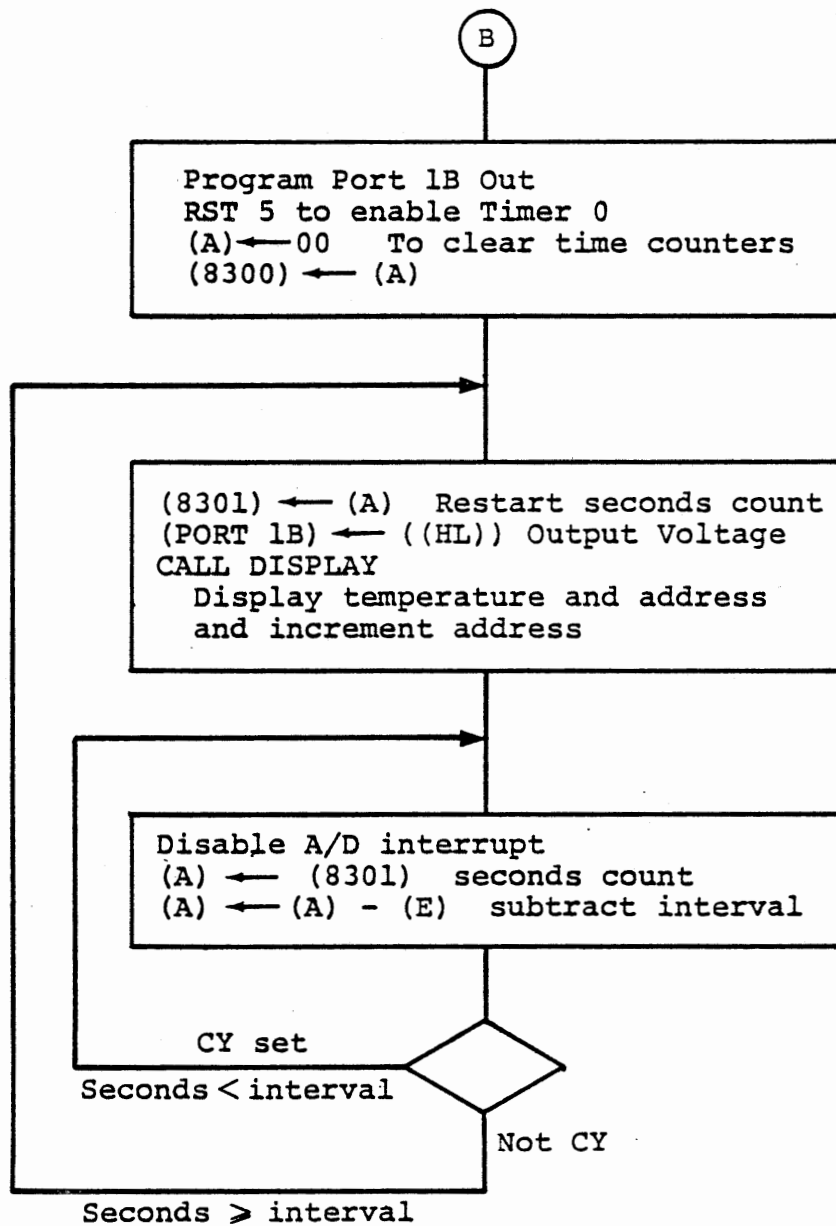
Display Subroutine



LOGGING THERMOMETER - REVIEW DATA

Figure 5-42

Temperature Replay



LOGGING THERMOMETER - REPLAY

Figure 5-43

Timer 0 Preload (high byte)	Interrupt Time	"One Second" Preload	Interval Count Time	Interval Entered	Recording Time	Max Run Time (approx)
08	1 ms	01	1 ms	01	1 ms	.25 sec
A0	20 ms	32	1 sec	01	1 sec	4 min
				0A	10 sec	40 min
				1E	30 sec	12 hrs
				3C	1 min	4 hrs
				78	2 min	8 hrs
F0	20 ms	C8	6 sec	0A	1 min	4 hrs
				3C	6 min	24 hrs
				64	10 min	40 hrs
				96	15 min	64 hrs
F0	30 ms	F0	7.2 sec	FA	30 min	5 days

LOGGING THERMOMETER - TIMING CONSTANTS

FIGURE 5-44

LOGGING THERMOMETER - INITIALIZE

5-149

	A	D	D	R	CODE							
CODING SHEET	8	2	0		3E	MVI	A,	82			Program 8255#1	
					82						Port 1B In	
					D3	OUT		CNT1				
					07							
					3E	MVI	A,	92			Program 8255#2	
					92							
					D3	OUT		CNT2				
					0F							
					3E	MVI	A,	24			Program Timer 0	
					24						High byte	
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT		TIMCT			Mode 2, Binary	
	B				17							
	C				3E	MVI	A,	94			Program Timer 2	
	D				94						Low byte	
	E				D3	OUT		TIMCT			Mode 2, Binary	
	F				17							
	MICROCOMPUTER TRAINING SYSTEM	8	2	1	0	3E	MVI	A,	A0			Load Timer 0
						A0						for 20 millisecond
						D3	OUT		TIMO			interrupt
						14						
					3E	MVI	A,	20			Load Timer 2	
					20						for 1.6 μsecond	
					D3	OUT		TIM2			clock to A/D	
					16							
					3E	MVI	A,	01			Set Port 1C0	
					01						for automatic A/D	
INTEGRATED COMPUTER SYSTEMS	A				D3	OUT		CNT1				
	B				07							
	C				CD	CALL		ENTBY				
	D				36						(L) ← Interval	
	E				03						(A) ← Command	
	F				FE	CPI		RUN				
	8	2	2	0	14							
					C2	JNZ		8100			Jump to review	
					00						if not RUN.	
					81							
				EF	RST5							
				C3	JMP		8255					
				58								
				82								

Figure 5-45a

LOGGING THERMOMETER - RUN LOOP 5-150

		A	D	D	R	CODE					
CODING SHEET	8	0									
		1									
		2									
		3									
		4									
		5									
		6									
		7									
MICROCOMPUTER TRAINING SYSTEM	825	8	01			LXI	B	8000		Initial data	
		9	00							storage address	
	A		80								
	825	B	11			LXI	D	8301		Address seconds	
		C	01								
		D	83								
	825	E	1A			LDAX	D			(A) ← seconds	
		F	95			SUB	L			Subtract interval	
	826	0	DA			JC		825E		Loop until	
		1	5E							seconds ≥ interval	
		2	82								
	INTEGRATED COMPUTER SYSTEMS		3	12			STAX	D			Restart seconds
		4	3A			LDA		8305		(A) ← filtered	
		5	05							voltage	
		6	83								
		7	02			STAX	B			Store voltage	
		8	0C			INR	C			Next address	
		9	CD			CALL		TEMP		Display voltage	
		A	BO							and temperature	
		B	82								
		C	C3			JMP		825B		Loop	
		D	5B								
		E	82								
	F										
	8	0			REQUIRES SUBROUTINES						
		1			FILTER		8270-82AF				
		2			TEMP		82B0-82FF				
		3			INITIAL DATA		8300-8305				
		4			TEMP TABLE		8310-				
		5									
		6									
		7									
		8									

Figure 45b

TEMPERATURE REVIEW WITH NXT KEY

		A	D	D	R	CODE						
CODING SHEET	8	1	0	0		EB		XCHG				(E) ← interval if any
			1			21		LXI	H,	8000		(HL) ← data address
			2			00						
			3			80						
			4			D2		JNC	8120			If command STEP
			5			20						jump to replay
			6			81						
		810	7			CD		CALL	8110			Display temperature
			8			10						and address
			9			81						
MICROCOMPUTER TRAINING SYSTEM		A				CD		CALL	GETKY			Wait for a key
			B			3D						
			C			02						
			D			C3		JMP	8107			Loop
			E			07						
			F			81						
		811	0			DS		PUSH	D			Display Subroutine
			1			7E		MOV	A,M			(A) ← voltage
			2			CD		CALL	TEMP			Display temperature
			3			B0						
INTEGRATED COMPUTER SYSTEMS						82						
			5			7D		MOV	A,L			Display address
			6			CD		CALL	DBYTE			
			7			95						
			8			02						
			9			2C		INR	L			Next address
			A			D1		POP	D			
			B			C9		RET				
			C									
			D									
		E										
		F										
	8	0										
		1										
		2										
		3										
		4										
		5										
		6										
		7										
		8										

Figure 5-45c

TEMPERATURE REPLAY

A D D R CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	120	3E	MVI	A, 80	Program Port 1B Out
	1	80			for D/A conversion
	2	D3	OUT	CNT1	
	3	07			
	4	EF	RST 5		Enable timer 0
	5	AF	XRA	A	interrupt
	6	32	STA	8300	Clear one second
	7	00			counter
	8	83			
812	9	32	STA	8301	Clear seconds
A		01			counter
B		83			
C		7E	MOV	A, M	(A) ← voltage
D		D3	OUT	PORT 1B	Output to
E		05			D/A converter
F		CD	CALL	8110	Display temperature
813	0	10	CNT		and address and
	1	81			increment address
813	2	3E	MVI	A, 06	Disable A/D
	3	06			interrupt
	4	D3	OUT	CNT2	
	5	0F			
	6	3A	LDA	8301	
	7	01			
	8	83			
	9	93	SUB	E	
A		DA	JC	8132	
B		32			
C		81			
D		C3	JMP	8129	
E		29			
F		81			
8	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				

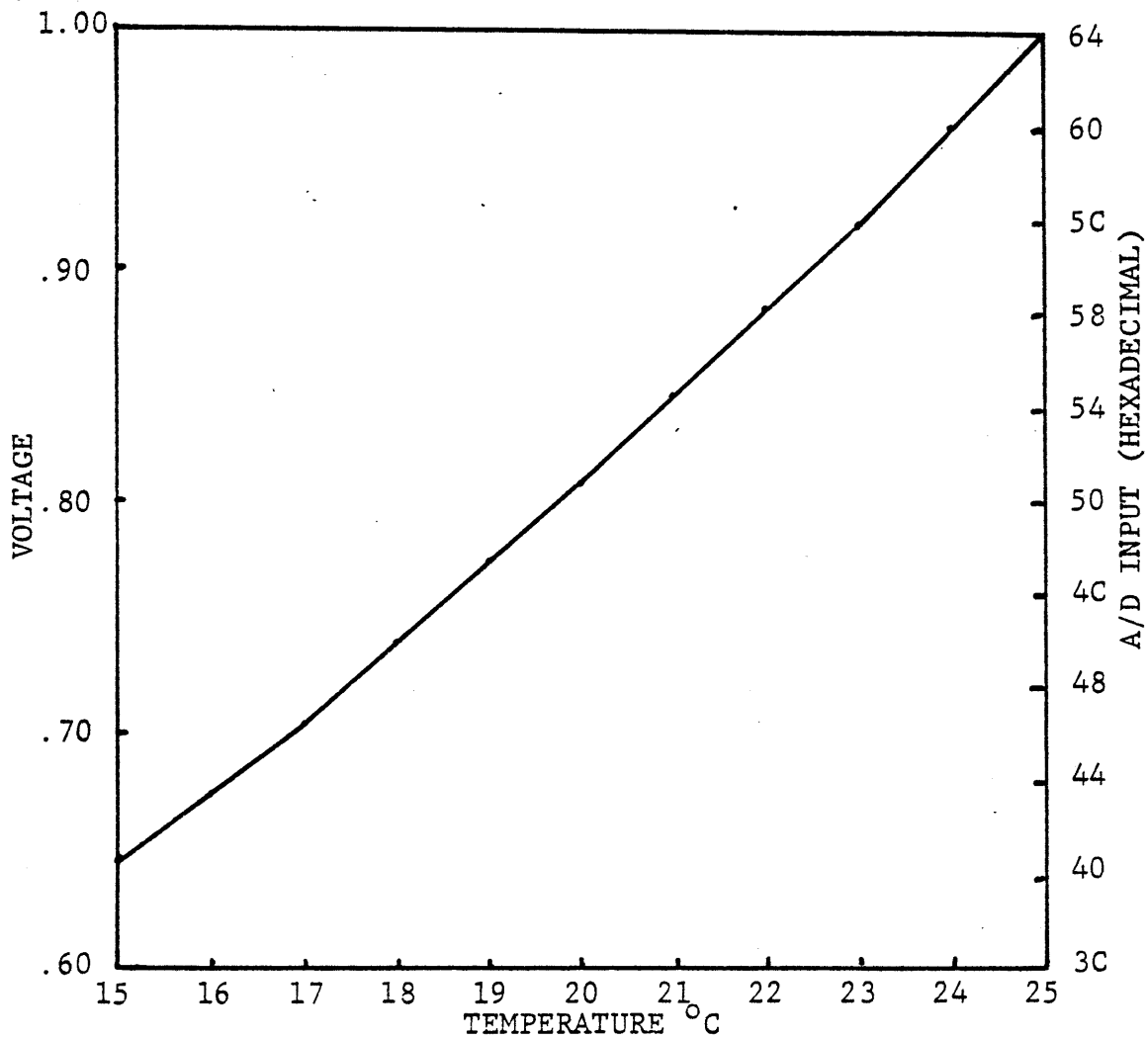
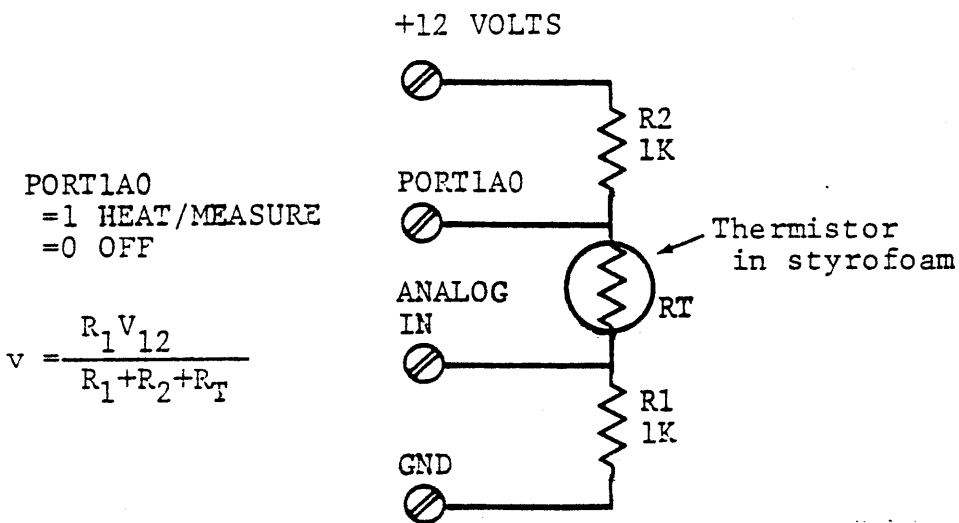
Figure 5-45d

5.6.8 Thermistor Self Heating

When a temperature measuring device is carrying current, as the thermistor does, it generates heat internally. If the thermal resistance between the sensor and its environment is very low, as it will be in a liquid, this means that the sensor affects the experiment. If the thermal resistance is high, as in still air, the measured temperature will be higher than the real temperature.

This effect is usually negligible. In the circuit we have been using, the maximum self heating is about 0.6 milliwatt, occurring near 25° C. Thus, the self heating would be less than 0.1 degree, not detectable with our A/D converter. For an experiment, the effect can be increased in two ways. Supply the thermistor from +12 volts with smaller series resistance to increase the heat generated, and bury the thermistor in a piece of styrofoam to decrease its dissipation. The connection of Figure 5-46 accomplishes this, and also allows switching the power to the thermistor on and off. We will show that the self heating error can be eliminated by applying power only during brief measurement intervals.

Since the circuit is changed, our previous calibration data are not valid for this experiment. Figure 5-46 shows a plot of voltage vs temperature for this connection, near room temperature. The input is in the neighborhood of one volt, so the ANALOG IN pot should be adjusted to no attenuation for maximum sensitivity.



THERMISTOR CONNECTION AND CALIBRATION FOR SELF HEATING EXPERIMENT

FIGURE 5-46

Note that with this connection, the slope is inverted from that of the normal connection, so the temperature conversion of the previous exercises must be modified. Interrupt service must be modified to set port 1A0 high at RST 5 to enable the measurement. To detect self heating, leave port 1A0 high, but to demonstrate its elimination, set port 1A0 low at RST 6.

In the interrupt service routine given in Figure 5-40b and 5-40c room was left for two patches to control power to the thermistor. In the subroutine that services Timer 0, we had:

```

MVI    A,01           To reenale Timer 0
NOP
NOP

```

Replace the NOP instructions with:

```

OUT    PORT1A        Turn on Thermistor Power

```

Now self heating should be measurable. To eliminate self heating we will turn thermistor power off after reading and processing the input. After the call to FILTR insert:

```

MVI    A,00           Turn off thermistor power
OUT    PORT1A
MVI    A,06           To disable A/D

```

Now power will be applied to the thermistor only while the A/D conversion is being performed. The MVI A,00 can be changed to MVI A,01 to again keep power on. The data log should demonstrate the

difference. The revised program is shown in Figure 5-47. One instruction in TEMP is changed from INR A to DCR A, and a very short calibration table is entered.

THERMISTOR SELF HEATING - INTERRUPT SERVICE

		A	D	D	R	CODE																	
CODING SHEET	8		0																				
			1																				
			2																				
			3																				
			4																				
			5																				
			6																				
			7																				
MICROCOMPUTER TRAINING SYSTEM	822	8	FS			PUSH	PSW													TIMER 0			
		9	ES			PUSH	H																
		A	CD			CALL	8246														Call for service		
		B	46																		of Timer 0		
		C	82																				
		D	C3			JMP	8240															Jump to exit	
		E	40																			with (A)=01	
		F	82																			to renewable Timer 0	
		823	0	FS			PUSH	PSW														A/D CONVERTER	
			1	ES			PUSH	H															
INTEGRATED COMPUTER SYSTEMS		2	DB			IN	PORT 1 B														Read A/D Input		
		3	05																				
		4	21			LXI	H, 8302															Data address	
		5	02																			for FILTR	
		6	83																				
		7	CD			CALL	FILTR															Calculate and store	
		8	70																			new voltage estimate	
		9	82																				
		A	3E	*		MVI	A, 00															To stop heating	
		B	00	*																		01 to leave heat on	
	C	D3	*		OUT	PORT 1 A																	
	D	04	*																				
	E	3E	*		MVI	A, 06																To disable and	
	F	06	*																			reset A/D	
	8	0																					
		1			*	CHANGED	FROM															5-406	
		2																					
		3																					
		4																					
		5																					
		6																					
		7																					
		8																					

Figure 5-47a

INTERRUPT SERVICE - EXIT, TIMER 0

		A	D	D	R	CODE			
CODING SHEET	8	24	0	D3		OUT	CNTZ	EXIT	
			1	0F					
			2	E1		POP	H		
			3	F1		POP	PSW		
			4	FB		EI			
			5	C9		RET			
MICROCOMPUTER TRAINING SYSTEM	824	6	3E		MVI	A, 07		TIMER 0 SERVICE	
		7	07						
		8	D3		OUT	CNTZ		Reset and enable	
		9	0F					A/D converter	
		A	3E		MVI	A, 01		Apply power to	
		B	01					thermistor	
		C	D3	*	OUT	PORTIA			
		D	04	*					
		E	21		LXI	H, 8300		Address and	
		F	00					decrement	
INTEGRATED COMPUTER SYSTEMS	825	0	83					one second counter	
		1	35		DCR	M		Unless zero	
		2	C8		RNZ			exit with (A)=01	
		3	36		MVI	M, 32		Reload one	
		4	32					second counter	
		5	23		INX	H		Address and	
		6	34		INR	M		increment seconds	
		7	C9		RET			Exit with (A)=01	
	8						to reenable Timer 0		
	9								
	A			*	CHANGED	FROM	5-40c		
	B								
	C								
	D								
	E								
	F								
	8	0							
	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								

Figure 5-47b

TEMPERATURE TABLE FOR FIGURE 5-46

	A	D	D	R	CODE															
CODING SHEET	8	3	1	0	08															
					1	72														
					2	99														
					3	00														
					4	78														
					5	02														
					6															
					7															
				8																
MICROCOMPUTER TRAINING SYSTEM																				
INTEGRATED COMPUTER SYSTEMS																				

Starting Temperature
 -2.792 as
 100's complement
 One slope for full
 voltage range
 0.278 °C/20mV

CHANGE IN TEMP
 FOR POSITIVE SLOPE
 DCR A

8ZCF 3D

Figure 5-47c

5.6.9 Other Temperature Logging Experiments

The thermistor is encapsulated so that it can be used in water, making several interesting experiments possible. Plot temperature versus time as you bring a pot of water from room temperature to boiling. Determine the temperature difference from a very gentle boil to a full boil. Determine the effect of a lid on the pot.

There is an old wives' tale that hot water will freeze more rapidly than cold water. Test it.

5.6.10 Abbreviated Temperature Lookup

For most measurement and control purposes, the extensive temperature table used up to here is not necessary, since it gives a precision greater than the absolute accuracy of the thermistor. In the program of Figure 5-48, a much shorter table is provided, fitting in the memory space 82EC to 82FF. Each slope is used for 32 additions, so the repetition count is not needed. This table gives the same results within ± 0.1 °C over the range of 10 °C to 40 °C and ± 1.0 °C from 0° C to 90° C.

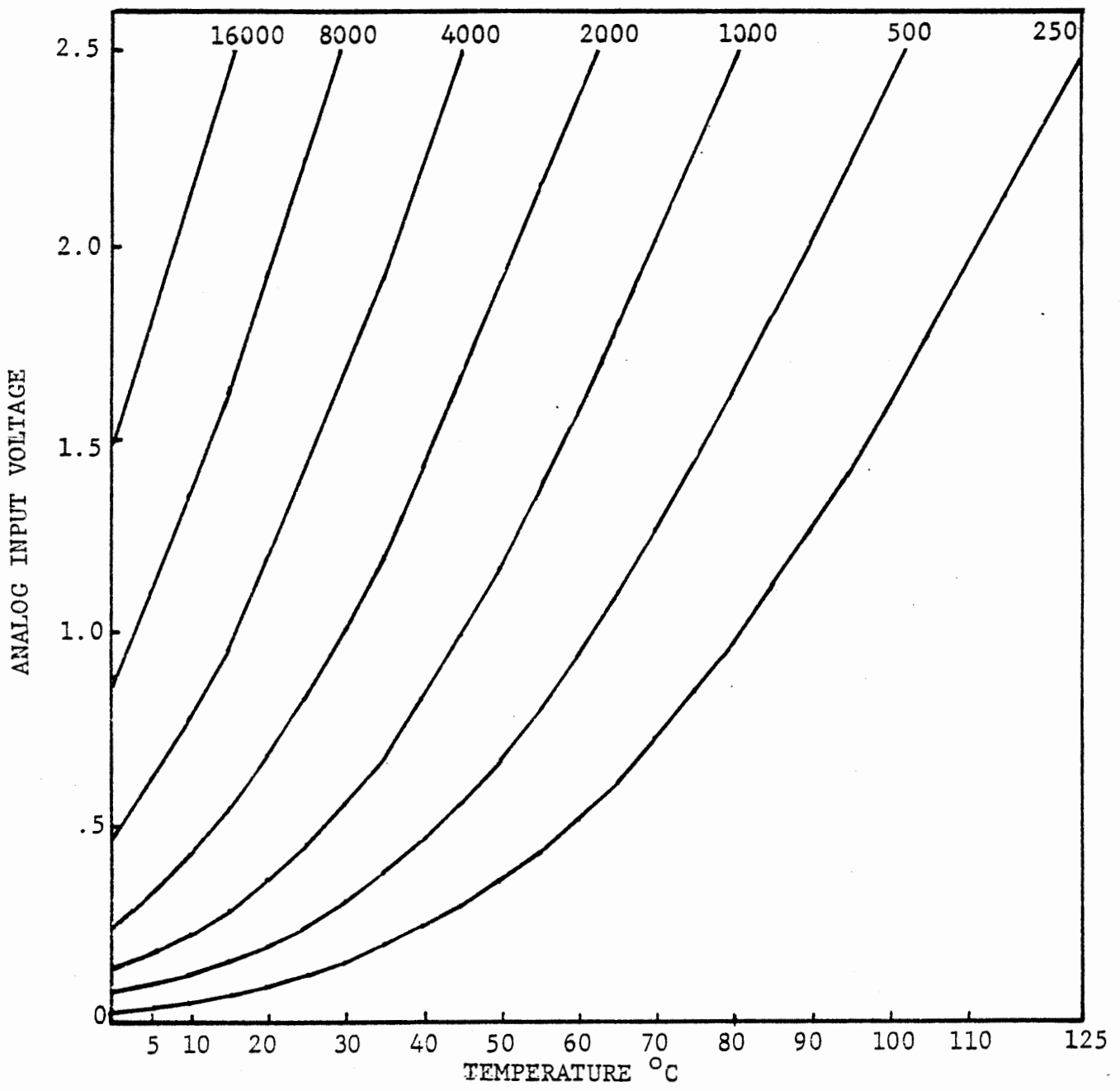
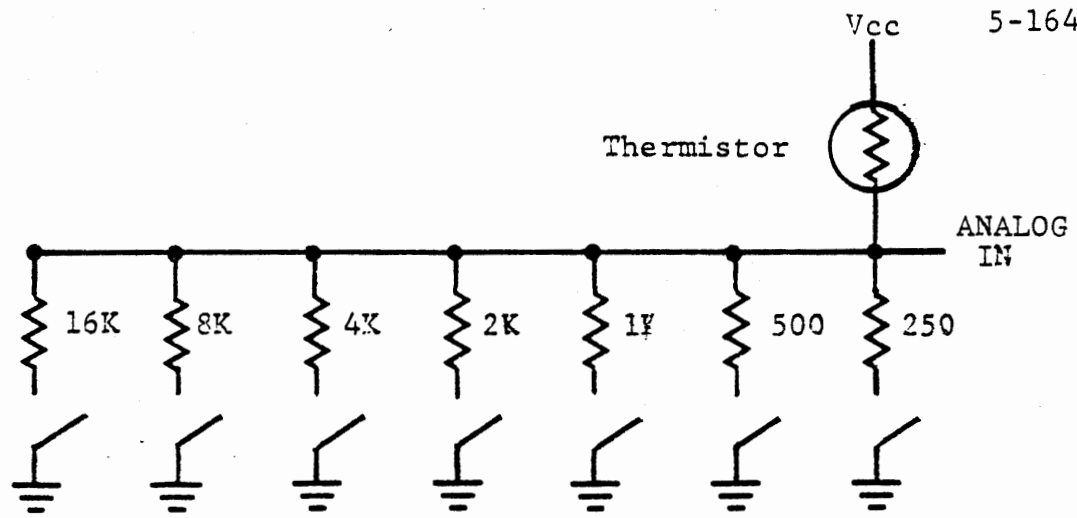
ABTMP - Abbreviated Temperature Lookup

A	D	D	R	CODE							
8	2B	0		ES		PUSH	H				Save registers
		1		CS		PUSH	B				
		2		FS		PUSH	PSW				Save voltage
		3		21		LXI	H, 82EC				Address initial
		4		EC	*						temperature in
		5		82	*						table
		6		46		MOV	B, M				} Copy starting value into B, E, D
		7		23		INX	H				
		8		5E		MOV	E, M				
		9		23		INX	H				
		A		56		MOV	D, M				
	82B	B		23		INX	H				Loop - address slope
		C		0E	*	MVI	C, 20				Set counter
		D		20	*						
	82B	E		FS		PUSH	PSW				Loop - Save A/D
		F		7E		MOV	A, M				} Add slope to temperature
8	2C	0		80		ADD	B				
		1		27		DAA					
		2		47		MOV	B, A				
		3		23		INX	H				
		4		7E		MOV	A, M				
		5		8B		ADC	E				
		6		27		DAA					
		7		5F		MOV	E, A				
		8		2B		DCX	H				
		9		3E		MVI	A, 00				
		A		00							
		B		8A		ADC	D				
		C		27		DAA					
		D		57		MOV	D, A				
		E		F1		POP	PSW				(A) ← A/D Input
		F		3C		INR	A				Increment Input
8		0									
		1									
		2			*	CHANGED	FROM	FIGURE	5-40	g	
		3									
		4									
		5									
		6									
		7									
		8									

Figure 5-48a

		A	D	D	R	CODE					
CODING SHEET	8 2 D	0	CA	JZ	8 2 D B	Exit when A/D					
		1	DB			Input is incremented					
		2	8 2			to zero					
		3	0 D	DCR	C	Decrement repetitions					
		4	C 2	JNZ	8 2 B E	Loop to add slope					
		5	B E			until 32 repetitions					
		6	8 2			of this slope done					
		7	2 3	INX	H	Address high byte					
		8	C 3	JMP	8 2 B B	Loop to address					
		9	B B			next slope					
MICROCOMPUTER TRAINING SYSTEM	A	8 2									
	B	E B	XCHG			Exit and display					
	C	C D	CALL	DWORD		Display temperature					
	D	D I									
	E	0 2									
	F	F I	POP	PSW		(A) ← voltage					
	8 2 E	0	C D	CALL	DBYTE	Display Voltage					
		1	9 5			(C) ← voltage					
		2	0 2			(A) ← 80					
		3	E B	XCHG		(DE) ← temperature					
INTEGRATED COMPUTER SYSTEMS		4	2 E	MVI	L, FA	Address whole					
		5	FA			degrees					
		6	B 6	ORA	M	Insert decimal					
		7	7 7	MOV	M, A	point					
		8	7 9	MOV	A, C	(A) ← voltage					
		9	C I	POP	B						
		A	E I	POP	H						
		B	C 9	RET							
		8 2 E	C	9 9		} Starting Temperature					
		D	6 2			} - 023.701 °C					
	E	9 7			} as 100's complement						
	F	0 5			FF - E0						
	8 2 F	0	0 4		0.405 °C / 20mV						
	1										
	2										
	3										
	4										
	5										
	6										
	7										
	8										

Figure 5-48b



THERMISTOR RESISTOR MATCHING
FIGURE 5-49

5.6.11 Thermistor Resistance Matching

OPTIONAL EXERCISE

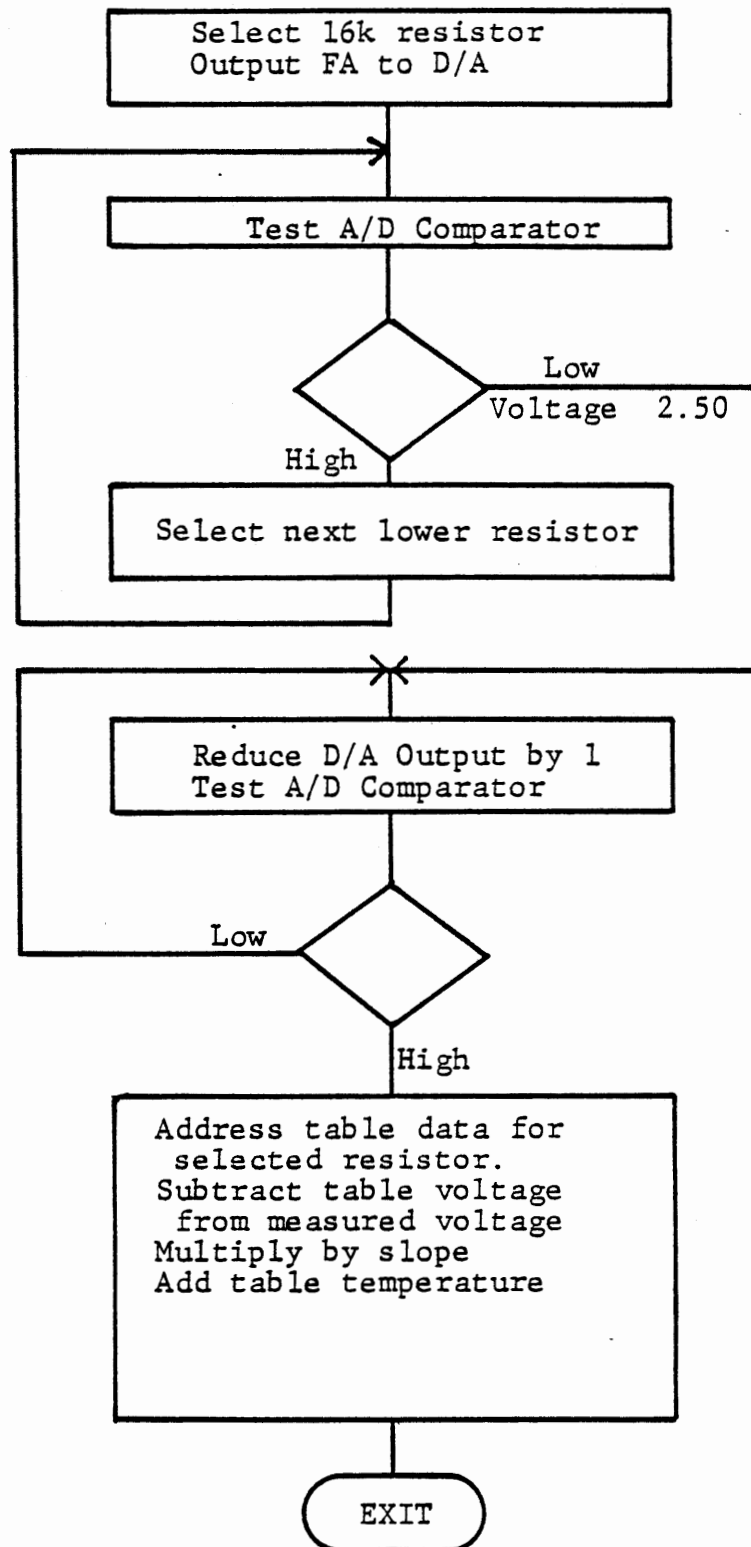
The chief difficulties in using the thermistor come from the non-linearity and high slope (degrees/volt) at higher temperatures. These problems can be avoided if the series resistance is well matched to the thermistor resistance at the temperature being measured. By switching discrete outputs the computer can accomplish this matching automatically. Figure 5-49 shows a circuit in which the thermistor is connected between Vcc and a resistor ladder which can be switched to provide different series resistance. Calibration curves for each resistance are shown. The advantage is that only a linear portion of each calibration curve is used, and a moderate slope (temperature/voltage) is retained at all temperatures.

Switching of the network can be done by the Port 1A outputs. These introduce an offset voltage because their outputs are not pulled perfectly to ground but only to about + 0.4 volts. To use this scheme effectively each curve should be calibrated independently, with two temperatures for each.

The processing algorithm finds the highest single resistor which brings the input voltage on-scale (less than 2.56 volts). Thus at 20 ° C the 16K resistor produces an off-scale voltage, but the 8K resistor produces 1.92 volts. A table stores, for each resistor, the lowest temperature for which that resistor will be used, the corresponding voltage, and the slope. The measured temperature is the

low temperature plus the slope times the voltage difference. Figure 5-50 shows the program flow.

The resistors required for this experiment are not supplied with the course.



THERMISTOR RESISTOR MATCHING FLOW
FIGURE 5-50

THIS PAGE INTENTIONALLY LEFT BLANK

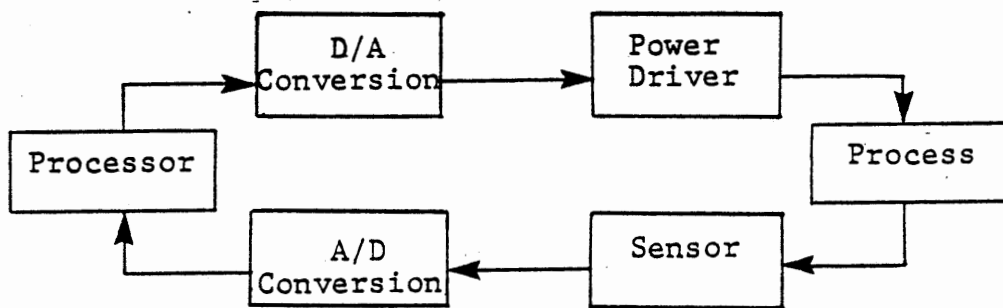
MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 6

CLOSED LOOP CONTROL

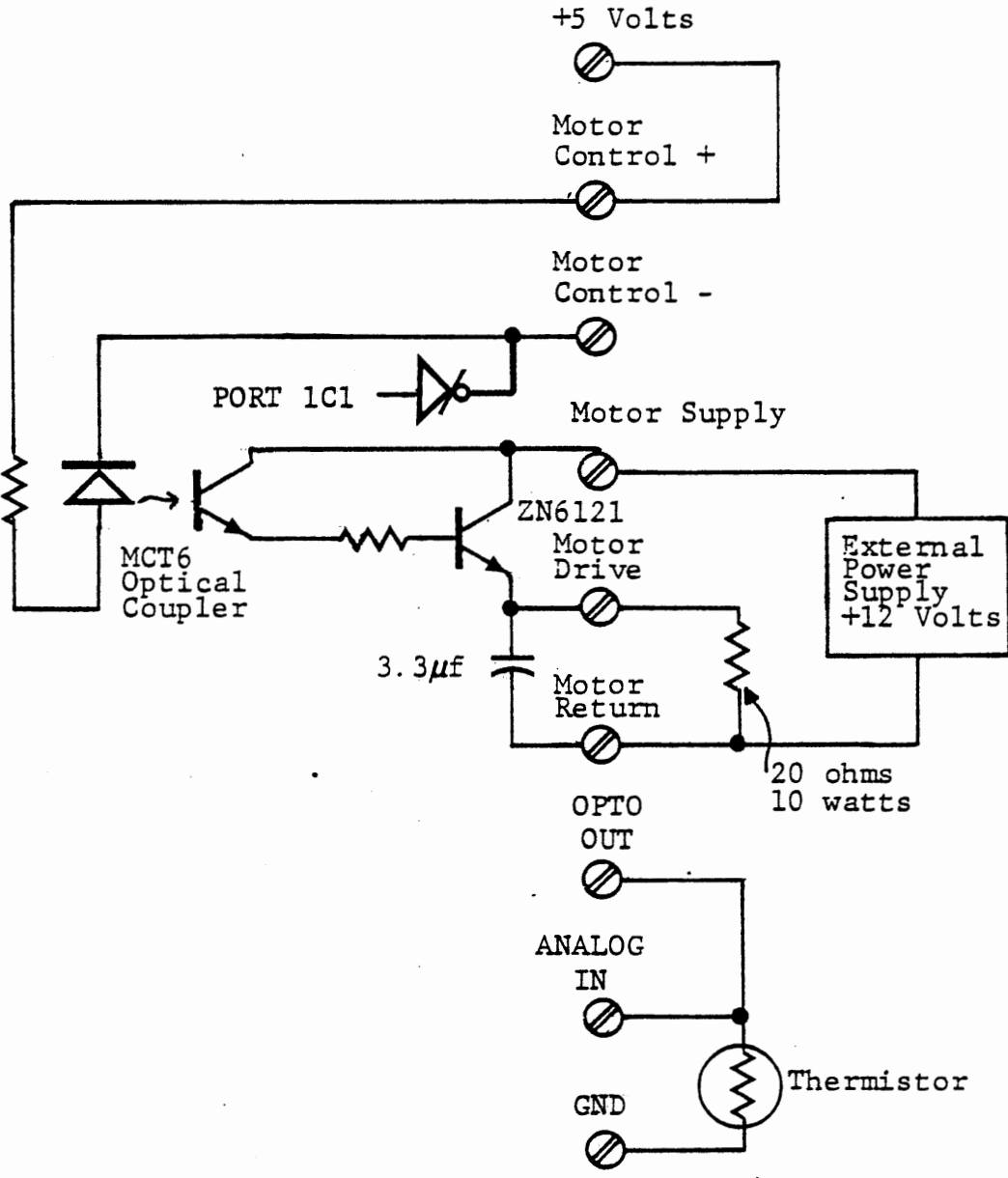
6. CLOSED LOOP CONTROL

As we have seen, a computer can generate control signals and receive sensed inputs. When the input is used to determine the control output, we have "closed the loop".



In this chapter we will deal with three closed loop problems: on-off (thermostat) control, proportional voltage or temperature control, and speed control.

Voltage and temperature control are essentially similar problems and will use the same forms of analog to digital input and digital to analog output. The automatic A/D input function of the Ferranti 425 will be used to sense the condition of the external "process", and output switching or pulse width modulation will be used for control. In speed control we will determine the speed by a frequency measurement, and we will try both PWM and the Ferranti D/A conversion for control.



CONNECTIONS FOR THERMOSTAT EXERCISE

FIGURE 6-1

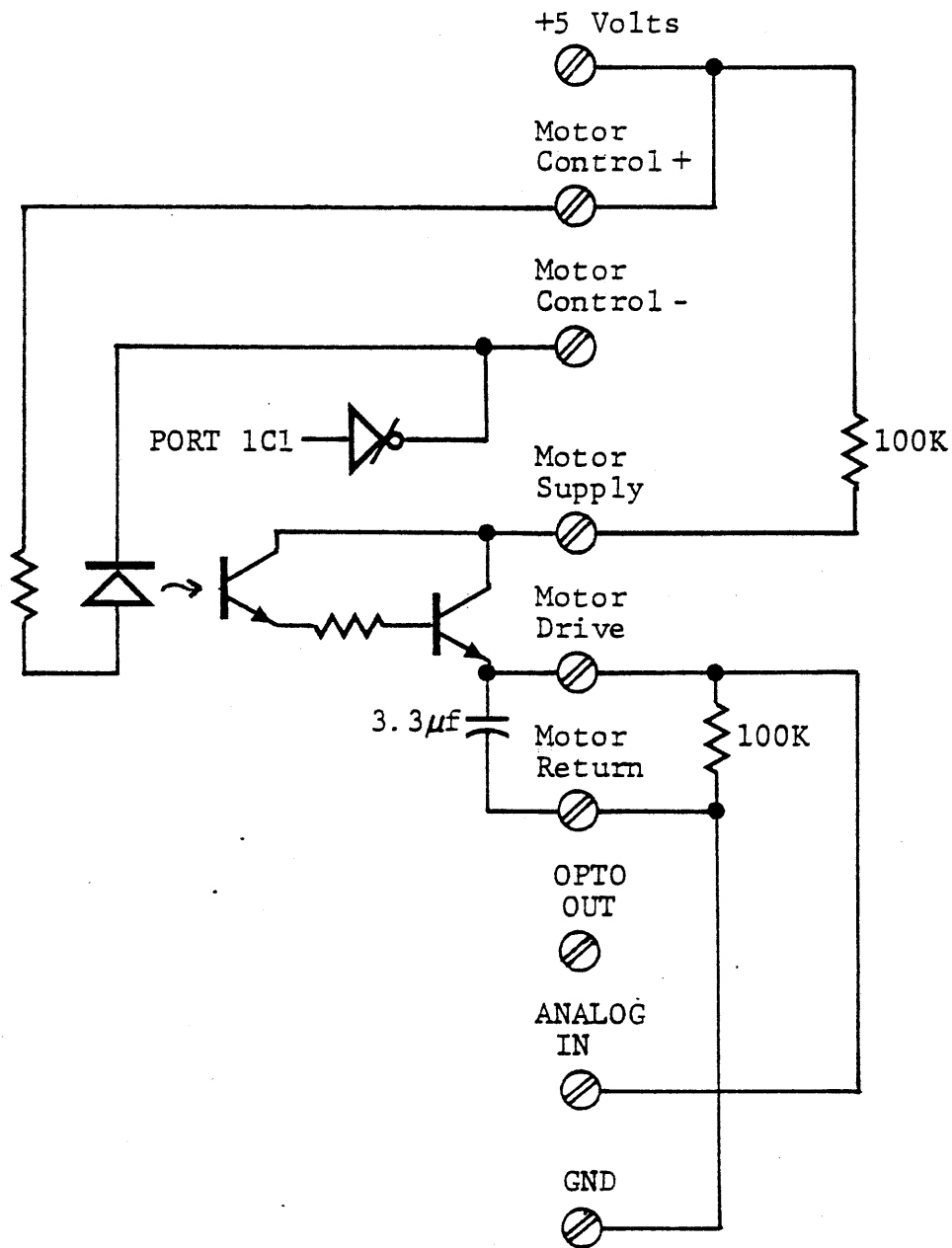
6.1 On-Off Control

The simplest closed loop control system is exemplified by the familiar household thermostat. The temperature is measured, and if it is too low (i.e., below some preset value) the furnace is turned on. When the temperature reaches the desired value the furnace is turned off.

An on-off control system usually has a "dead band" in which the present condition of the output is not changed - if the furnace is on it stays on until an upper temperature limit is reached; if it is off it stays off until a low limit is reached.

Without the dead band the output will switch on and off too frequently, resulting in inefficient operation (and annoying noise in the household situation). In a simple thermostat, the dead band is provided by mechanical hysteresis in the bimetal switch. In more sophisticated systems the upper and lower limits can be programmed independently. It is also possible to let system time constants provide the dead band, as we shall see in the first exercise.

The interface board includes a Fairchild 2N6121 power transistor driven by a Monsanto MCT6 optical coupler. This circuit can be used directly to heat a power resistor, as shown in Figure 6-1. The thermistor must be coupled to the heater as closely as possible; they can be wrapped together with tape, since the heating available is not very great.

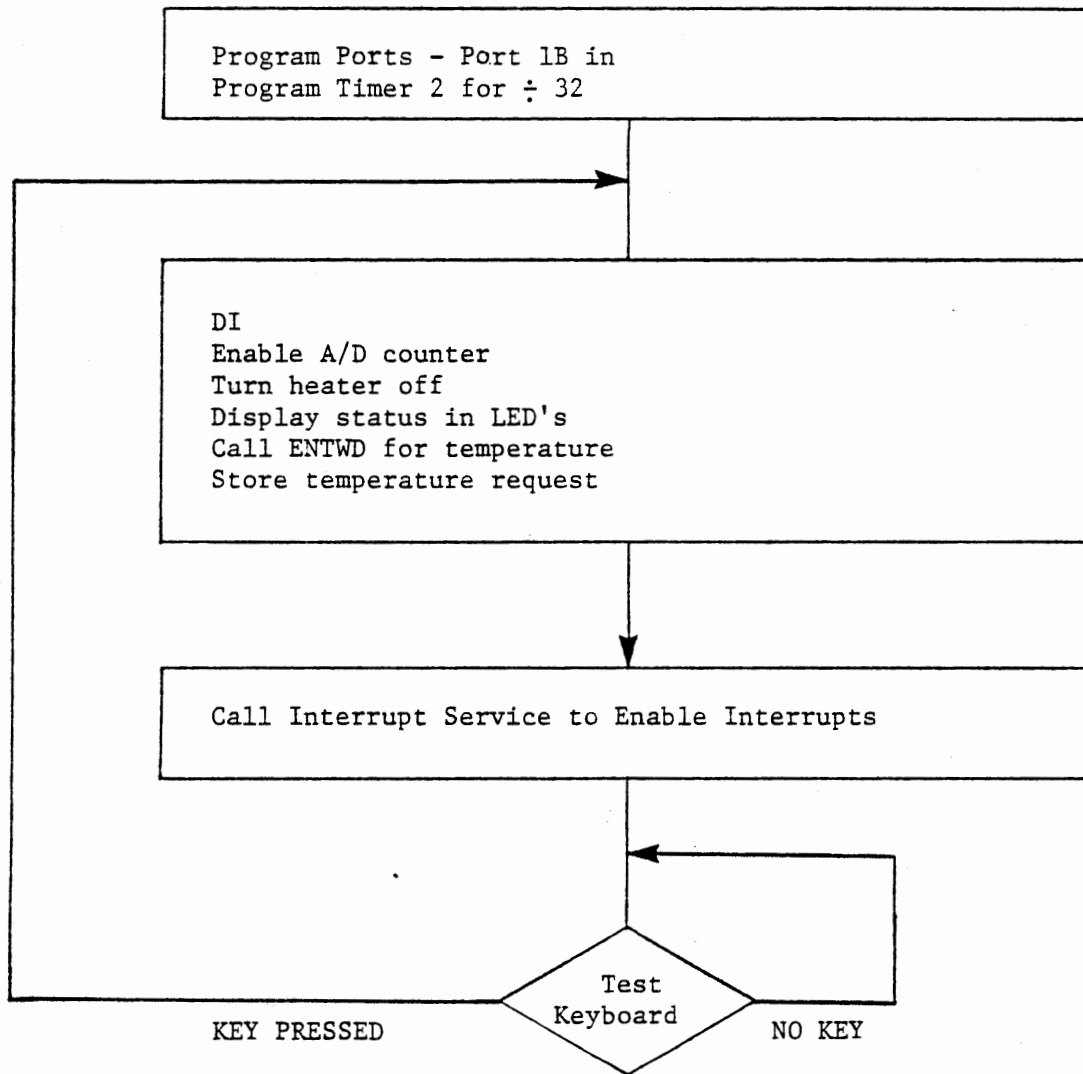


CONNECTIONS FOR ON-OFF VOLTAGE CONTROL

FIGURE 6-2

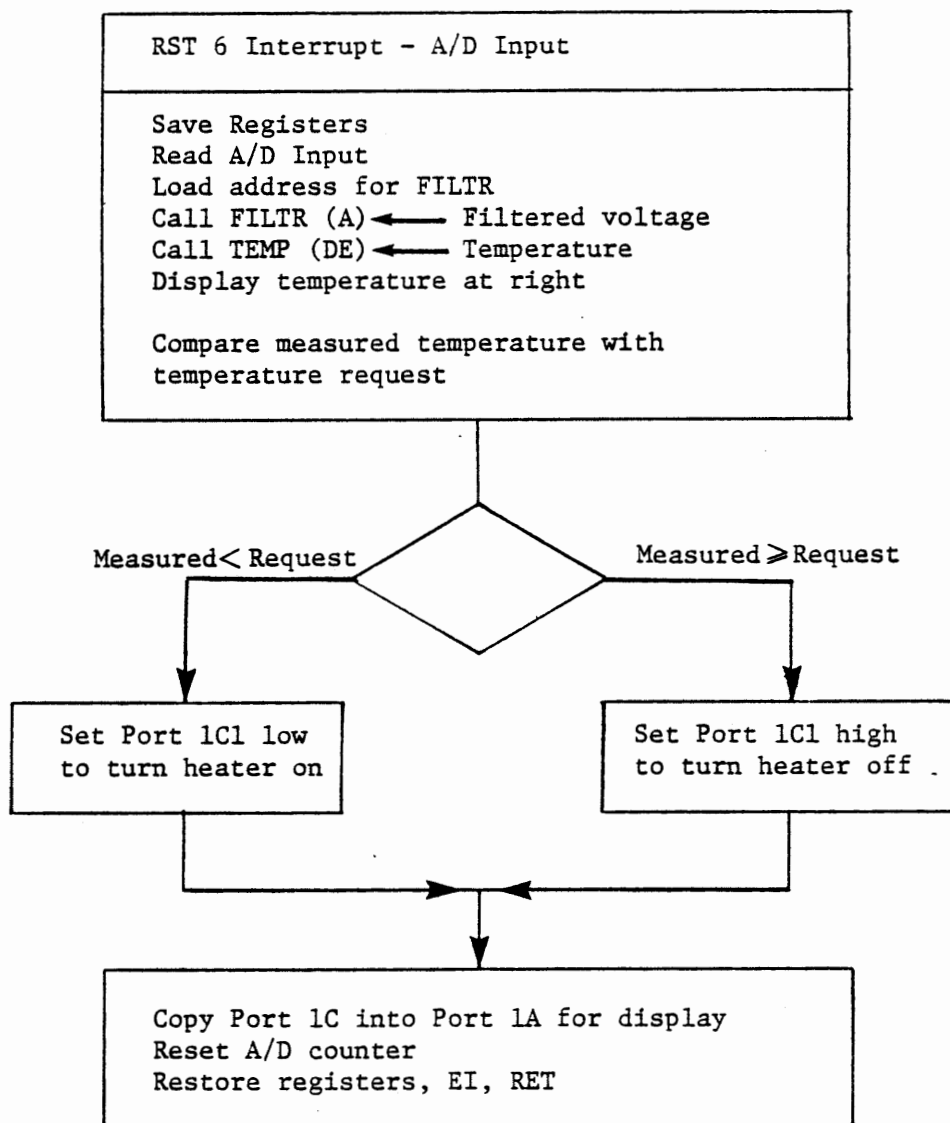
Alternately, the power transistor can drive a relay controlling power to an ac heater, with the thermistor in water that is being heated. (Note: Exercise 6.1.1 should not be done with a relay, because the fast switching would damage the relay contacts).

The experiments can be performed without heating anything, but simply simulating heat by the charge on a capacitor and measuring its voltage, as shown in Figure 6-2.



Initialization
On-Off Control - No Deadboard

FIGURE 6-3a



Interrupt Service

ON-OFF CONTROL - NO DEADBOARD

FIGURE 6-3b

6.1.1 On-Off Control Without Deadband

EXERCISE

The on-off control experiment will use subroutines TEMP and FILTR, developed in Chapter 5. The program of Figure 6-3 accepts a temperature request (by keyboard entry) and attempts to heat the load to that temperature as measured by the thermistor. Whenever the temperature is less than that requested it turns the heater on by setting Port 1C1 low; when the temperature is greater it sets Port 1C1 high to turn the heater off. The temperature is calculated and displayed by subroutine TEMP. The measurement and control functions are contained entirely in the RST6 interrupt service, with the main program performing initialization and then calling ENTWD for a temperature request. The desired temperature must be entered as degrees and tenths, to permit comparison with the temperature returned by TEMP. For instance, enter 315 to request a temperature of 31.5 oC.

Clearly this program does nothing very exciting. Its main purpose is to demonstrate the effect of having no deadband. Observe the high frequency at which it switches the power on and off when the desired temperature is reached. This is no problem to the power transistor, nor to an SCR, but would damage the contacts of a relay. Moreover, many heating devices are less efficient when being switched on and off repeatedly, and some may be damaged.

THERMOSTAT WITHOUT DEADBAND

	A	D	D	R	CODE					
CODING SHEET	8	2	0		3E	MVI	A,	82		Program Ports
					82					Port B In
					D3	OUT		CNT1		
					07					
					3E	MVI	A,	92		
					92					
					D3	OUT		CNT2		
					0F					
					3E	MVI	A,	94		Program Timer 2
					94					Low byte
MICROCOMPUTER TRAINING SYSTEM	A				D3	OUT		TIMCT		Mode 2
	B				17					Binary
	C				3E	MVI	A,	20		Load Timer 2
	D				20					for ÷ 32
	E				D3	OUT		TIM2		
	F				16					
	8	2	1			F3	DI			No interrupts while
						3E	MVI	A,	03	in ENTWD
						03				PICO high for A/D
						D3	OUT		PORTIC	PICI high to
INTEGRATED COMPUTER SYSTEMS					06					turn power off
					D3	OUT		PORTIA		Display in LED's
					04					
					CD	CALL		ENTWD		Enter temperature
					46					limit in degrees
					03					and tenths -
					A	22	SHLD		8306	eg 273 for 27.3°C
					B	06				Store temperature
					C	83				limit
					D	F7	RST6			Enable A/D Interrupt
	8	2	1		E	DB	IN		PORTOA	Test keyboard
					F	00				(FF if no key)
	8	2	2			3C	INR	A		
					1	CA	JZ		821E	Loop until
					2	1E				Key pressed
					3	82				
					4	C3	JMP		8210	Go to DI and
					5	10				turn power off
					6	82				and accept data
					7					
					8					FIGURE 6-4a

THERMOSTAT - RST6 INTERRUPT

		A	D	D	R	CODE					
CODING SHEET	8	2	3	0		F5	PUSH	PSW			
				1		E5	PUSH	H			
				2		D5	PUSH	D			
				3		C5	PUSH	B			
				4		21	LXI	H, 8300		Data address	
				5		00				for FILTR	
				6		83					
				7		DB	IN	PORT1B		Read A/D	
				8		05					
				9		CD	CALL	FILTR		(A) ← filtered	
MICROCOMPUTER TRAINING SYSTEM		A				70				voltage	
						82					
				C		CD	CALL	TEMP		Calculate and	
				D		B0				display temperature	
				E		82				(DE) ← Temperature	
				F		2A	LHLD	8306		(HL) ← Requested	
	INTEGRATED COMPUTER SYSTEMS	8	2	4	0		06				Temperature
					1		83				
					2		7D	MOV	A, L		Compare temperature
					3		93	SUB	E		to request
				4		7C	MOV	A, H		Sets CY if	
				5		9A	SBB	D		temperature high	
				6		3E	MVI	A, 01		} (A) ← 03 if high ← 02 if low	
				7		01					
				8		17	RAL				
				9		D3	OUT	CNT1		Set Port 1C1 low	
			A		07				(power on) if		
			B		DB	IN	PORT1C		temperature low		
			C		06						
			D		D3	OUT	PORT1A		Read and display		
			E		04				power control		
			F		00	NOP					
	8			0							
				1							
				2							
				3							
				4							
				5							
				6							
				7							
				8						FIGURE 6-46	

THERMOSTAT - EXIT FROM INTERRUPT

	A	D	D	R	CODE						
CODING SHEET	8	25	0	3E	MVI	A	07			Enable A/D and	
										reset counter	
						OUT	CNT2				
						POP	B				
						POP	D				
						POP	H				
						POP	PSW				
						EI					
					RET						
	A										
	B										
	C										
	D										
	E										
	F										
MICROCOMPUTER TRAINING SYSTEM	8	0			SUBROUTINES REQUIRED						
					FILTR	8270-AF					
					TEMP	82B0-EF					
					DATA REQUIRED						
					TEMP	TABLE 8310-836F					
					8300	04	} Preload for FILTR				
					8301	00					
					8302	00					
					8303	00					
				8306		} Temperature Request stored by MAIN					
				8307							
	A										
	B										
	C										
	D										
	E										
	F										
INTEGRATED COMPUTER SYSTEMS	8	0									

FIGURE 6-4C

SUBROUTINE FILTR

A D D R CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	27	0	DS	PUSH	D			Save (DE)
	1		CS	PUSH	B			Save (BC)
	2		46	MOV	B, M			(B) ← m
	3		48	MOV	C, B			(C) ← m
	4		23	INX	H			Address $2^m E_i - 1$
	5		ES	PUSH	H			Save address
	6		5E	MOV	E, M			} (DE) ← $2^m E_i - 1$
	7		23	INX	H			
	8		56	MOV	D, M			} (HL) ← $2^m E_i - 1$
	9		6B	MOV	L, E			
	A		62	MOV	H, D			
827	B		29	DAD	H			} (HL) ← $2^{2m} E_i - 1$
	C		0D	DCR	C			
	D		C2	JNZ	827B			
	E		7B					
	F		82					
828	0		4F	MOV	C, A			(C) ← V_i
	1		7D	MOV	A, L			} (DE) ← (HL) - (DE) = $2^m(2^m - 1) E_i - 1$
	2		93	SUB	E			
	3		5F	MOV	E, A			
	4		7C	MOV	A, H			
	5		9A	SBB	D			
	6		57	MOV	D, A			
	7		69	MOV	L, C			} (HL) ← V_i
	8		26	MVI	H, 00			
	9		00					
	A		CD	CALL	SHFTN			Divide by 2^m
	B		A0					(DE) ← $(2^m - 1) E_i - 1$
	C		82					
	D		EB	XCHG				} (DE) ← $V_i + (2^m - 1) E_i - 1$ = $2^m E_i$
	E		19	DAD	D			
	F		EB	XCHG				
8	0			CONTINUED				AT 8290
	1							
	2			ENTER	(A) = V_i			(new value)
	3			(HL)	= m			(filter constant)
	4			(HL) + 1	?			$2^m E_i - 1$
	5			(HL) + 2	}			
	6			RETURN				
	7			(HL) + 3	= (A) = (H) = E_i			
	8				(L) = V_i			

A	D	D	R	CODE						
8	29	0	E3		XTHL					(HL) ← Address $2^m E_i$
		1	73		MOV	M	,	E		} Store $2^m E_i$
		2	23		INX	H				
		3	72		MOV	M	,	D		
		4	23		INX	H				Address E_i
		5	1B		DCX	D				To roundup only if
		6	CD		CALL	SHFTN				fractional part $> 1/2$
		7	A0							
		8	82							
		9	77		MOV	M	,	A		Store E_i
A			E1		POP	H				(L) ← V_i
B			67		MOV	H	,	A		(H) ← E_i
C			C1		POP	B				Restore BCDE
D			D1		POP	D				
E			C9		RET					Exit
F			00		NOP					
8	2A	0	48		MOV	C	,	B		SHFTN (C) ← m
		1	AF		XRA	A				Loop - clear carry
		2	7A		MOV	A	,	D		} Shift (DE) right to divide by 2
		3	1F		RAR					
		4	57		MOV	D	,	A		
		5	7B		MOV	A	,	E		
		6	1F		RAR					
		7	5F		MOV	E	,	A		
		8	0D		DCR	C				Loop m times
		9	C2		JNZ	82A1				to divide by 2^m
A			A1							
B			82							
C			D0		RNC					Exit if LSB = 0
D			13		INX	D				Else roundup
E			7B		MOV	A	,	E		(A) ← less
F			C9		RET					significant byte
8		0								
		1								
		2								
		3								
		4								
		5								
		6								
		7								
		8								

Figure 6 - 4e

TEMPERATURE LOOKUP AND DISPLAY

	A	D	D	R	CODE						
CODING SHEET	8	2	B	0	E5	PUSH	H				
				1	C5	PUSH	B				
				2	F5	PUSH	PSW				
				3	21	LXI	H, 8310				
				4	10						
				5	83						
				6	46	MOV	B, M	}	Copy starting value into B, E, D		
				7	23	INX	H				
				8	5E	MOV	E, M				
				9	23	INX	H				
			A	56	MOV	D, M					
MICROCOMPUTER TRAINING SYSTEM	8	2	B	B	23	INX	H	Loop - address count (C) ← repetitions Address 'slope'			
				C	4E	MOV	C, M				
				D	23	INX	H				
				8	2	B	E	F5	PUSH	PSW	Loop - save A/D
				F	7E	MOV	A, M	}	Add low byte of slope to low byte of temperature.		
	8	2	C	0	80	ADD	B				
				1	27	DAA					
				2	47	MOV	B, A				
				3	23	INX	H				
				4	7E	MOV	A, M	}	Address high byte of slope and add to second byte of temperature		
			5	8B	ADC	E					
			6	27	DAA						
			7	5F	MOV	E, A					
			8	2B	DCX	H					
			9	3E	MVI	A, 00	}	Address low byte Add carry to high byte of temperature			
			A	00							
			B	8A	ADC	D					
			C	27	DAA						
			D	57	MOV	D, A					
INTEGRATED COMPUTER SYSTEMS				E	F1	POP	PSW	(A) ← A/D input Increment input			
				F	3C	INR	A				
	8			0							
				1							
				2							
				3							
				4							
				5							
			6								
			7								
			8								

Figure 6 - 4f

TEMPERATURE WITH TWO BYTE DISPLAY cont'd

A D D R		CODE				
CODING SHEET	8 2 D 0	CA	JZ	8 2 DB		Exit when A/D
		DB				input is incremented
		8 2				to zero
		0 D	DCR	C		Decrement repetitions
		C 2	JNZ	8 2 BE		Loop to add slope
		BE				until all repetitions
		8 2				of this slope done
		2 3	INX	H		Address high byte
		C 3	JMP	8 2 BB		Loop to address
		BB				next slope
MICROCOMPUTER TRAINING SYSTEM	A	8 2				
	8 2 D B	EB	XCHG			Exit and display
	C	CD	CALL	DWORD		Display Temperature
	D	DI				
	E	0 2				
	F	F 1	POP	PSW		(A) ← voltage
	8 2 E 0	CD	CALL	DBYTE		Display
		9 5				(C) ← voltage
		0 2				(A) ← 80
		EB	XCHG			(DE) ← temperature
INTEGRATED COMPUTER SYSTEMS		2 E	MVI	L, FA		Address whole
		FA				degrees
		B 6	ORA	M		Insert decimal
		7 7	MOV	M, A		point
		7 9	MOV	A, C		(A) ← voltage
		C 1	POP	B		
	A	E 1	POP	H		
	B	C 9	RET			
	C					
	D					
E						
F						
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

Figure 6 - 4g

TABLE OF REPETITIONS AND SLOPES

	A	D	D	R	CODE														
CODING SHEET	8	3	1	0	99													} Starting temperature	
					1	62												} -0.23.701 °C	
					2	97												} as 100's complement	
					3	40													FF - C0
					4	05													} 0.405 °C/20mv
					5	04													}
					6	10													BF - B0
					7	73													} 0.373 °C/20mv
					8	03													}
					9	10													AF - A0
MICROCOMPUTER TRAINING SYSTEM	A				41													} 0.341 °C/20mv	
	B				03													}	
	C				20													9F - 80	
	D				24													} 0.324 °C/20mv	
	E				03													}	
	F				10													7F - 70	
	8	3	2	0	35													0.335 °C/20mv	
					1	03													
					2	10													6F - 60
					3	55													0.355
INTEGRATED COMPUTER SYSTEMS					4	03													
					5	10													5F - 50
					6	94													0.394
					7	03													
					8	10													4F - 40
					9	58													0.458
					A	04													
					B	08													3F - 38
					C	32													0.532
					D	05													
				E	08													37 - 30	
				F	06													0.606	
	8	3	3	0	06														
				1															
				2															
				3															
				4															
				5															
				6															
				7															
				8															

Figure 6 - 4h

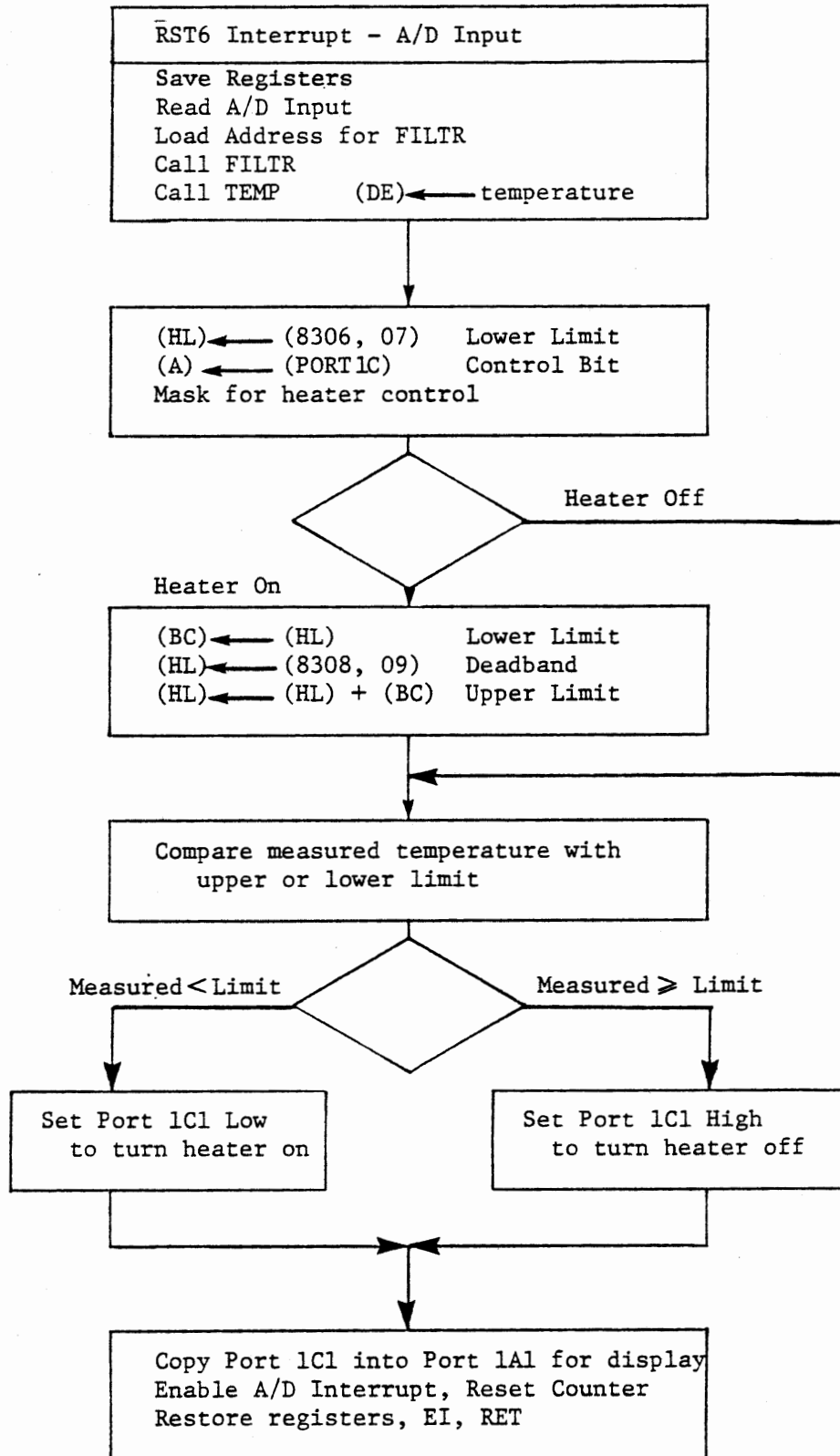
	A	D	O	R	CODE															
CODING SHEET	8				0															
	8	3	3	1	04														2F-2C	
				2	81															0.681
				3	06															
				4	04															2B-28
				5	43															0.743
				6	07															
				7	04															27-24
				8	20															0.820
				9	08															
MICROCOMPUTER TRAINING SYSTEM				A	04															23-20
				B	18															0.918
				C	09															
				D	04															1F-1C
				E	46															1.046
				F	10															
		8	3	4	0	04														1B-18
				1	14															1.214
				2	12															
				3	04															17-14
INTEGRATED COMPUTER SYSTEMS				4	52															1.452
				5	14															
				6	04															13-10
				7	03															1.803
				8	18															
				9	02															0F-0E
				A	90															2.190
				B	21															
				C	02															0D-0C
				D	24															2.524
				E	25															
				F	02															0B-0A
		8	3	5	0	89														3.089
				1	30															
				2																
				3																
				4																
				5																

Figure 6 - 4i

THIS PAGE INTENTIONALLY LEFT BLANK

		A	D	D	R	CODE								
CODING SHEET	8	0												
	1													
	835	2	0	1							09			
		3	7	3							3.573			
		4	3	5										
		5	0	1							08			
		6	6	5							4.065			
		7	4	0										
		8	0	1							07			
		9	9	7							4.697			
MICROCOMPUTER TRAINING SYSTEM	A	4	6											
	B	0	1							08				
	C	3	7							5.537				
	D	5	5											
	E	0	1							07				
	F	0	0							6.700				
	836	0	6	7										
		1	F	F							06	125.974°		
		2	0	0							upper limit for			
		3	0	0							thermistor			
INTEGRATED COMPUTER SYSTEMS		4												
		5												
		6												
		7												
		8												
		9												
		A												
		B												
		C												
		D												
		E												
		F												
	8	0												
		1												
		2												
		3												
	4													
	5													
	6													
	7													
	8													

Figure 6 - 4j



THERMOSTAT WITH DEADBAND - RST6

FIGURE 6-5

.1.2 On-Off Control With Deadband

EXERCISE

Modify the program of the preceding Section (6.1.1) to provide a deadband. You can do this either by entering upper and lower temperature limits or by entering a value for the deadband which is always added to the lower temperature limit to obtain the upper limit. (The latter approach is used in the given solution.)

The main program is identical to that of figure 6-4a except that instead of storing data entered through ENTWD it calls a subroutine (STORE, at 82FO) which stores a deadband value at 8308,09 if the STEP key was used, or a lower temperature limit at 8306, 07 if the RUN key was used. NEXT stores the data entered as a lower temperature limit and also enters a default deadband of 2 degrees.

Figure 6-5 shows interrupt service for the deadband thermostat. The present state of the control is tested to determine whether power is on or off. If it is off, the temperature is compared with the lower limit. If the power is on, the deadband is added to the lower limit before the comparison is made. Now power is turned on if the temperature is less than the selected limit.

THERMOSTAT WITH DEADBAND - MAIN

6 - 22

A D D R		CODE				
CODING SHEET	8	200	3E	MVI	A, 82	Initialization
		1	82			same as
		2	D3	OUT	CNT1	figure 6-4a
		3	07			
		4	3E	MVI	A, 92	
		5	92			
		6	D3	OUT	CNT2	
		7	0F			
		8	3E	MVI	A, 94	
		9	94			
MICROCOMPUTER TRAINING SYSTEM	A	D3	OUT	TIMCT		
	B	17				
	C	3E	MVI	A, 20		
	D	20				
	E	D3	OUT	TIMZ		
	F	16				
	8	210	F3	DI		Data entry same
		1	3E	MVI	A, 03	as figure 6-4a
		2	03			
		3	D3	OUT	PORTIC	
INTEGRATED COMPUTER SYSTEMS		4	06			
		5	D3	OUT	PORTIA	
		6	04			
		7	CD	CALL	ENTWD	
		8	46			
		9	03			
	A	CD	CALL	STORE		Store temperature
	B	F0				limit at 8306,07
	C	82				or deadband at 08,09
	D	F7	RST6			Enable A/D Interrupt
E	DB	IN	PORTOA		Test keyboard	
F	00					
8	220	3C	INR	A		
	1	CA	JZ	821E	Loop until	
	2	1E			key pressed	
	3	82				
	4	C3	JMP	8210	Go to DI and	
	5	10			turn power off	
	6	82			and accept data	
	7					
	8				FIGURE 6-6a	

THERMOSTAT - RSTG INTERRUPT SERVICE

	A	D	D	R	CODE						
CODING SHEET	8	2	3	0	FS	PUSH	PSW				
				1	ES	PUSH	H				
				2	DS	PUSH	D				
				3	CS	PUSH	B				
				4	21	LXI	H, 8300	Address for	FILTR		
				5	00						
				6	83						
				7	DB	IN	PORTIB	Read	A/D		
				8	05						
				9	CD	CALL	FILTR	(A) ← filtered			
MICROCOMPUTER TRAINING SYSTEM	A			70				A/D voltage			
	B			82							
	C			CD	CALL	TEMP	Calculate	temperature			
	D			B0			Display				
	E			82			(DE) ←	temperature			
	F			2A	LHLD	8306	(HL) ←	low limit			
	8	2	4	0	06			for	temperature		
				1	83						
				2	DB	IN	PORTIC	(A) ←	controls		
				3	06						
INTEGRATED COMPUTER SYSTEMS				4	E6	ANI	02	Test	for	heater	
				5	02			on	or	off	
				6	C2	JNZ	8250	Jump	if	off	
				7	50			(Port	IC1	high)	
				8	82						
				9	4D	MOV	C, L	Heater	On		
	A			44	MOV	B, H	(BC) ←	temp	request		
	B			2A	LHLD	8308	(HL) ←	deadband			
	C			08							
	D			83							
			E	09	DAD	B	(HL) ←	upper	limit		
			F	00	NOP						
INTEGRATED COMPUTER SYSTEMS	8			0							
				1							
				2							
				3							
				4							
				5							
				6							
				7							
			8								

FIGURE 6-6 b

THERMOSTAT - RSTG CONTINUED

A D D R		CODE					
CODING SHEET	8 25	0	7D	MOV	A ₂ L		Compare temperature
		1	93	SUB	E		with limit
		2	7C	MOV	A ₂ H		Sets CY if
		3	9A	SBB	D		temperature high
		4	3E	MVI	A ₂ 01		} (A) ← 03 if high ← 02 if low
		5	01				
		6	17	RAL			
		7	D3	OUT	CNT1		Set Port1C1 low
		8	07				(power on) if
		9	DB	IN	PORT1C		Temperature low
MICROCOMPUTER TRAINING SYSTEM	A		06				
	B		D3	OUT	PORT1A		Display power control
	C		04				
	D		3E	MVI	A ₂ 07		Reenable and
	E		07				reset A/D
	F		D3	OUT	CNT2		
	8 26	0	0F				
INTEGRATED COMPUTER SYSTEMS		1	C1	POP	B		
		2	D1	POP	D		
		3	E1	POP	H		
		4	F1	POP	PSW		
		5	FB	EI			
		6	C9	RET			
		7					
		8			SUBROUTINES REQUIRED		
		9			FILTR	8270-AF	
		A			TEMP	82B0-EF	
	B			DATA REQUIRED			
	C			TEMP	TABLE 8310-836F		
	D						
	E			8300	04	} Preload for FILTR	
	F			8301	00		
8	0			8302	00		
	1			8303	00		
	2						
	3			8306	07	Temperature Limit	
	4			8308	09	Deadband	
	5					Stored by STORE	
	6					(82F0-82FF)	
	7						
	8					FIGURE 6-6c	

SUBROUTINE STORE

		A	D	D	R	CODE						
CODING SHEET	8	2	F	0	FE		CPI	RUN			Test Command	
					14							
					2	DA	JC	82FC			If STEP jump to store deadband	
					3	FC						
					4	82						
					5	22	SHLD	8306			Else store lower temperature	
					6	06						
					7	83						
					8	C8	RZ				Exit if RUN	
					9	21	LXI	H, 0020			If NEXT enter default deadband of 2.0°C	
MICROCOMPUTER TRAINING SYSTEM	A				20							
	B				00							
	C				22	SHLD	8308			STEP (or NEXT)		
	D				08					Store deadband		
	E				83							
	F				C9	RET						
	INTEGRATED COMPUTER SYSTEMS	8				0						
						1						
						2						
						3		ENTER WITH				
					4		(HL) = TEMPERATURE °C					
					5		(LOWER LIMIT)					
					6		OR DEADBAND °C					
					7		(A) = COMMAND					
					8		COMMANDS ARE					
					9		RUN - STORE TEMP ONLY					
				A		STEP STORE DEADBAND						
				B		NEXT STORE TEMPERATURE						
				C		AND SET DEFAULT						
				D		DEADBAND = 2.0°C						
				E								
				F								
	8				0							
					1							
					2							
					3							
					4							
					5							
					6							
					7							
					8						FIGURE 6-6d	

6.1.3 Thermostat with Alarm Limits

OPTIONAL EXERCISE:

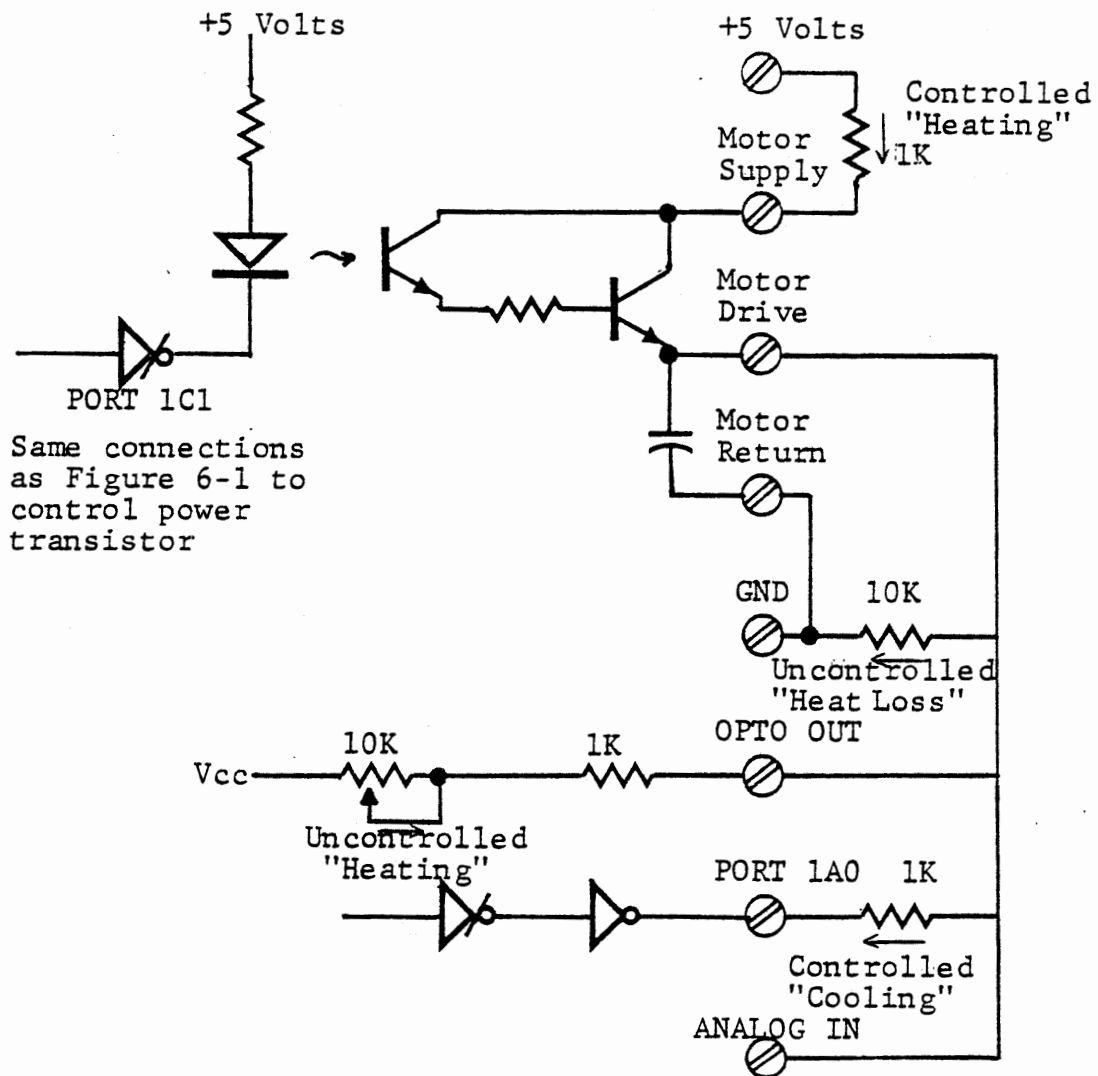
A very common requirement in process control systems is to test a temperature for certain limits and give an alarm if the limits are exceeded. This is likely to be used in conjunction with a temperature control, either thermostatic or proportional, so there may be as many as five limits:

- | | |
|--------------|---|
| Highest | - Danger to equipment or personnel.
Probably indicates an equipment failure. |
| Next Highest | - Process is out of control. May result
in degraded product. |
| Normal High | - Heater should be turned off to maintain
normal process temperature. |
| Normal Low | - Heater should be turned on to maintain
normal process temperature. |
| Lowest | - Process is out of control. May result
in degraded product. |

In a batch process, of course, the initial and final temperatures are likely to be lower than the lowest control temperature, so the alarm corresponding to that temperature should not be given until after the normal low temperature has been reached, nor after the process has been turned off.

There may also be time limits imposed on the process. In Section 4.2.9 an optional exercise was suggested in which a heater was controlled according to a schedule of times, with an upper limit for off-time and a lower limit for on-time. We will impose such limits here. In addition, we will place an upper limit for on-time, because if the heater stays on too long it may indicate a failure in the temperature sensor such that the highest temperature limit could be reached without being detected.

Develop your own flow charts and program for this problem. Select limits that can be reached by the heater you are using, and force the alarm conditions to occur by connecting MOTOR CONTROL - to ground so that the heater will stay on.



Same connections as Figure 6-1 to control power transistor

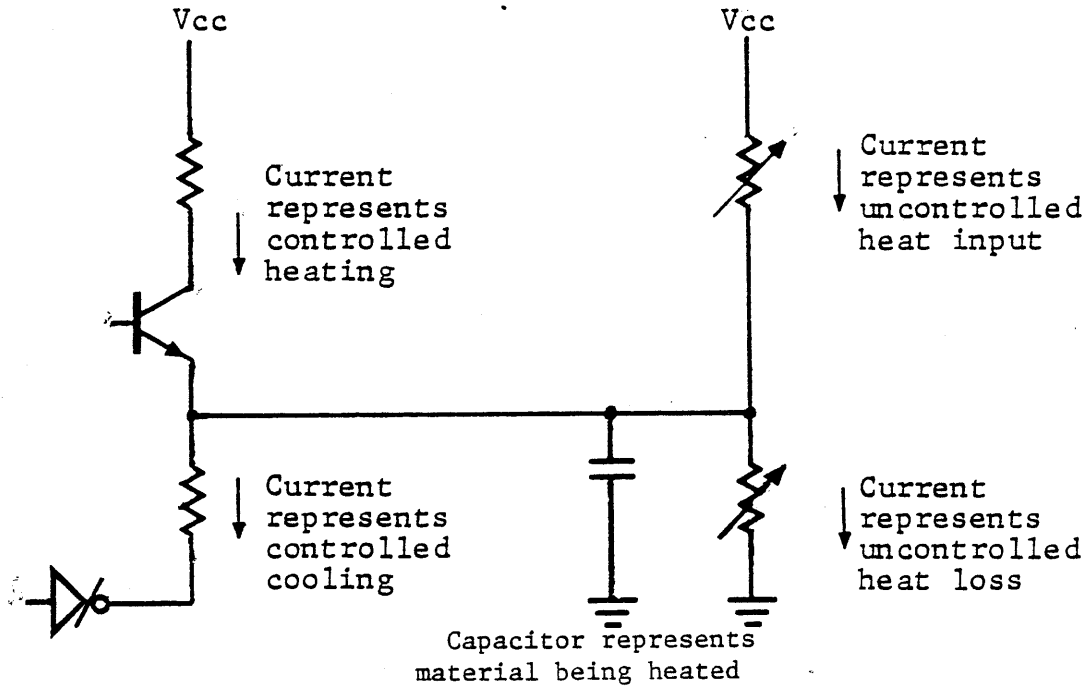
CIRCUIT CONNECTIONS FOR SIMULATION

FIGURE 6-7

1.4 Two-Way Control

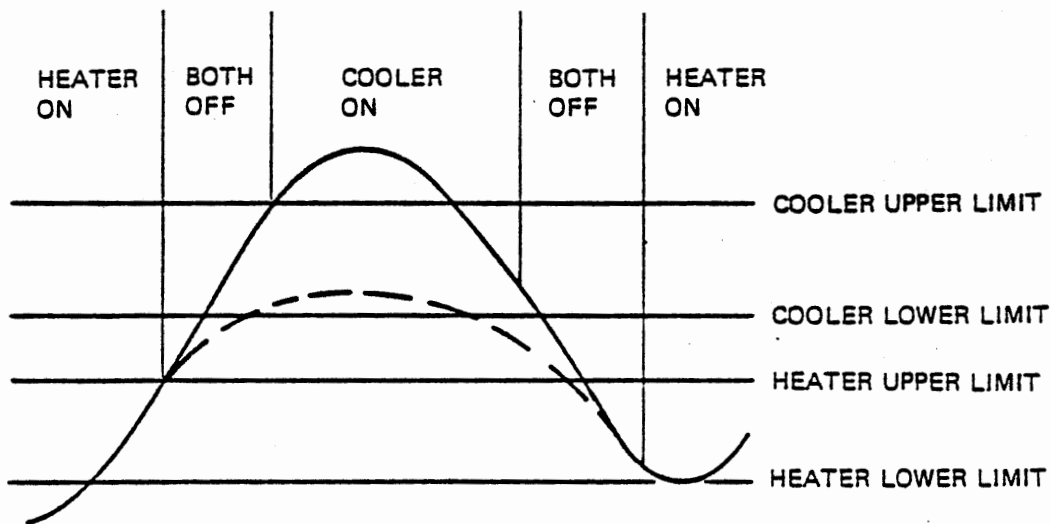
OPTIONAL EXERCISE:

In the simple thermostat problem it was assumed that only heating would be needed to maintain a required temperature; incidental heat loss would provide for cooling. In many temperature control problems that is not the case - both heating and cooling are required. The building with both furnace and air conditioner is the obvious example, but chemical processes also may require both. We cannot readily provide a realistic exercise with a heater and cooler, but the functions are easily simulated by charging and discharging a capacitor. Figure 6-8 below shows a circuit diagram for such a simulation, and Figure 6-7 shows connections to the interface board.



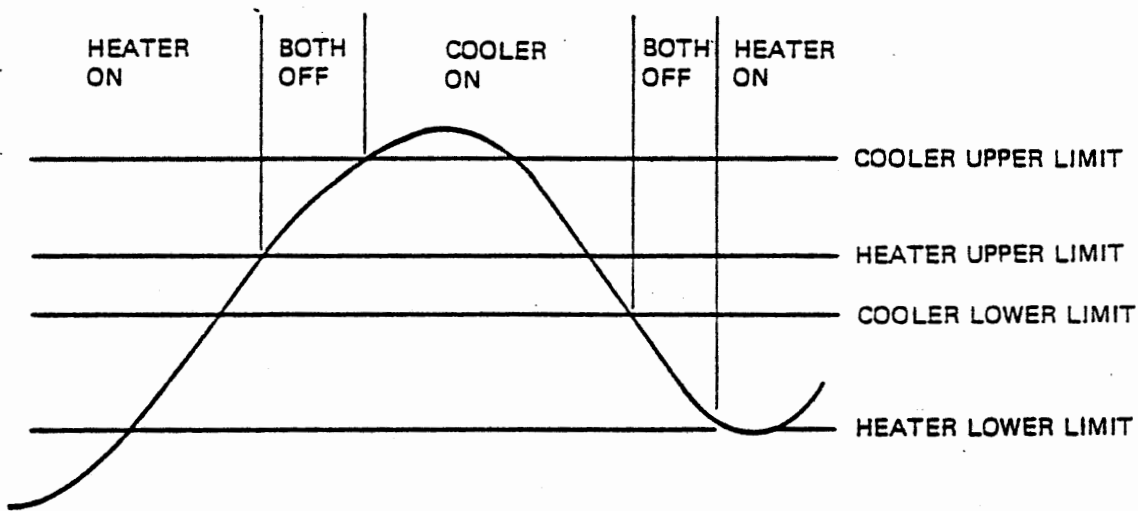
HEATING AND COOLING SIMULATION

FIGURE 6-8



Normal Inner Limits

Figure 6-9a



Integral Inner Limits

Figure 6-9b

Heating and Cooling Limits

Figure 6-9

We use the SENSE pot to simulate a variable heat input that cannot be controlled by the program and a 10K resistor to simulate heat loss. (An external 10K pot can be used instead, but is not necessary). The power transistor and Port 1A0, each with a series 1K resistor, represent the controlled heating and cooling.

Figure 6-9 depicts the control process. In Figure 6-9a a normal situation is shown. As the process temperature rises with the heater on, it crosses the heater lower limit and reaches the heater upper limit, where the heater is turned off. If either thermal inertia or uncontrolled heating causes the temperature to reach the cooler upper limit the cooler is turned on to return the process toward the desired temperature, and it runs until the temperature drops to the cooler lower limit. If the temperature does not reach the cooler upper limit the cooler remains off (dashed line). Obviously in the case of temperature control of a building (rather than a chemical process) there are many days when only the heater is turned on or only the cooler is turned on.

In process control however, it is often the case that the cooler involves greater thermal inertia than the heater. Heating can usually begin very quickly after a control signal and also stop very quickly. Cooling is likely to involve significant delay, since cold water must be pumped through pipes that are full of water heated by the process, and when the cooler is turned off the remaining cold water in the pipes will still absorb heat. In such a case the heater upper limit may be above the cooler lower limit, as shown in Figure 6-9b. In an

extreme case, both cooler limits might be between the two heater limits. This implies that sometimes both the heater and cooler would be in operation at the same time. The conclusion is that independent upper and lower limits should be provided for both the heater and the cooler. (These may of course be expressed as lower limit and deadband).

Clearly the control algorithm is essentially the same as for the case of a heater (or cooler) only, but must be processed twice for each temperature measurement. Once again we leave program development to the student.

.2 PROPORTIONAL Vs. INTEGRAL CONTROL

On - off control is unsuitable for many purposes. Imagine steering a car with a three position switch allowing only left turn, straight ahead, or right turn. You might make your way along a sufficiently broad highway, but the turn radius suitable for high speed driving would make parking very difficult. Proportional control is a method of applying a varying control force depending on the magnitude of the adjustment required.

In steering a car along a straight road you may be able to take your hands off the steering wheel for several seconds, and continue in a straight line. Soon, however, the car will drift off the track and a gentle touch on the wheel will be needed to return to the straight line. You have observed an "error signal" and applied a "control force" to correct the error. When the error becomes zero you again relax the control force.

If a bump in the road caused the car to swing sharply you would apply a more vigorous control force through the steering wheel. This is the essence of proportional control: a larger error results in a stronger restoring force.

When you reach an intersection and want to turn, you have changed the "setpoint". Suddenly a large error signal appears, because now the intended direction (i.e. setpoint) is pointed 90 degrees away from your car's current direction. You apply a large control force to correct for this large error signal, until once again the error signal

reaches zero. Thus the control force is in direct relation or proportional to the error signal.

A proportional control system can be described by:

$$\text{Control Force} = \text{Gain} (\text{Desired Value} - \text{Measured Value}).$$

The difference between the desired value and the measured value is called the error signal. It may be temperature, distance, angle, speed, voltage, or any of a host of other continuous variables. The control force might be electric current or gas flow to a heater; voltage, current or frequency to an electric motor; hydraulic pressure, etcetera. Note that gain is usually not a dimensionless ratio in this abstract form. For instance if a measurement in degrees is to give a control force in pounds, gain would have the dimensions "pounds per degree". In many cases with computer control both the measured value and the control force may appear as voltage analogs of the real variables.

In some processes it is not enough to provide only a correcting force. It may be necessary to provide some driving force all the time. For instance, we might operate a heater at some steady current, but increase or decrease that current in response to a temperature measurement. Then the control force would be:

$$(a) \quad F = G (E) + S$$

where F = control force

G = gain

E = error signal as above

S = steady state force

Now if the system is well understood and conditions are constant, corrective changes are made in the control force only when some disturbance causes an error signal.

If you were driving a camper or truck on a windy highway you would not take your hands off the steering wheel. Some force is needed all the time to keep the truck going in a straight line. This force must be increased or decreased momentarily to compensate for gusts or bumps. It is not a constant force, however, but must be adjusted if the wind force changes. Systems of this kind, that require a continuous control force to be adjusted for both momentary and long term changes, are more common than those where the control force is usually zero or constant.

The first kind of control system, with no steady state force, is called "proportional control" or "pure proportional control". It is well suited to processes that are subject to temporary disturbances or changes in setpoint, but will otherwise do what is intended by themselves. A simple autopilot in an airplane or boat detects any error between the compass and the setpoint and applies a control force to reduce the error to zero. Until a wind gust or a wave disturbs it, the vessel will go in a straight line.

When some control force is required all the time, as on our windswept highway, pure proportional control is not satisfactory because it will

déliver a control force only when there is an error signal - as our camper goes off the road. To provide the continuously adjustable control, we would use "integral control". This system is named from its control equation, which is described below. Integral control can provide a steady state control force that will maintain a zero error signal when there are no disturbances. It will correct for disturbances such as the wind gusts, and it will adjust the steady state force in response to changing conditions.

Pure integral control also has a weakness. It is inclined to overcorrect for momentary disturbances because it cannot immediately distinguish between (for instance) a gust of wind and a change in the wind force. When both momentary and long term changes in the conditions will occur it is best to use a combination of proportional and integral control.

In the remainder of this chapter we will develop a "proportional plus integral" control system, working up by steps from an open loop system of pulse width modulation, to a pure integral control system, and finally to the combined system. Much of the description, the program, and the experiments will be devoted to observing the behavior of the system. First we will develop the control equation for an integral control system to see why it is so named.

In pure integral control we adjust the control force whenever an error signal occurs, and maintain that new control force until another error signal is measured.

$$(b) \quad F = G \cdot E(1) + G \cdot E(2) + G \cdot E(3) \text{ ----}$$

As long as repeated measurements indicate a positive error (desired value greater than measured value) the control force will be increased. Eventually it will be enough to make the error negative, and the force will be reduced. After a time, if the control system is stable, the force will reach just the right value to maintain a zero error under the existing conditions. The sum of a series of measurements of a variable is the integral of that variable, so if equation (b) is expressed as:

$$F(i) = G \cdot E(i) + F(i-1)$$

or

$$(c) \quad F(i) = G \int E$$

the derivation of the name "integral control" is apparent.

In the following experiments we require both analog output for the Control Force and analog input to determine the error signal. Since the experiment board uses the D/A converter for A/D conversion we will use pulse width modulation to generate a voltage on a capacitor for the digital to analog output and measure that voltage with the automatic A/D input. Clearly, in any of these exercises the output voltage or the pulses could be driving some other load such as a

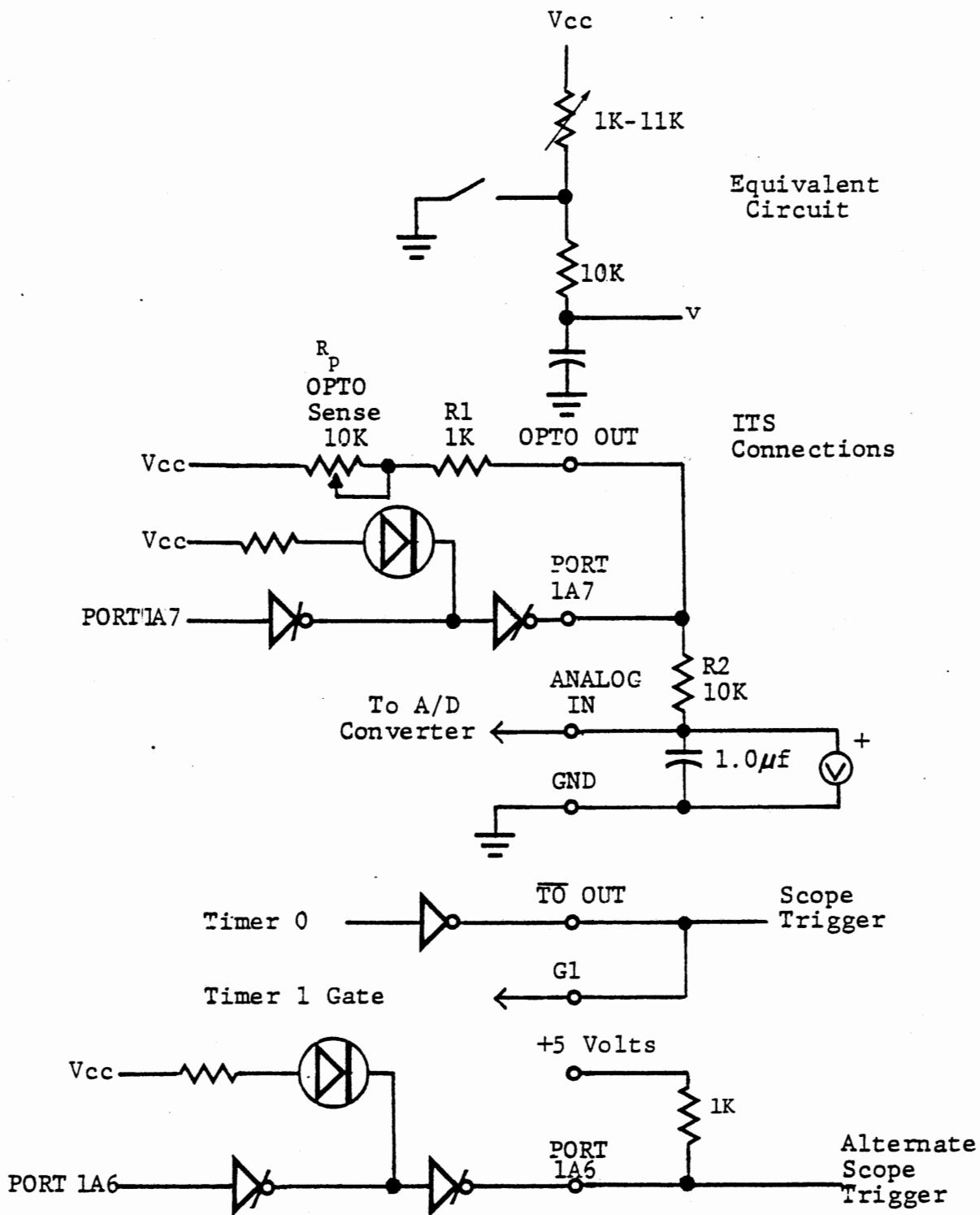
heater or a motor, and the voltage input could be obtained from a thermistor for heat, a linear potentiometer for position, or some other sensor with a voltage output. It is important here that control is exercised so as to force a measurable end result to be equal to a desired value.

In the first exercise we will develop some of the program modules necessary for the Pulse Width Modulation Control System: Initialization, Data Entry via Keyboard, Digital Data Filter, Digital Voltmeter Display, Interrupt Service for PWM and the Main Loop. However, we will not yet develop the modules responsible for Error Signal generation until section 6.2.4. This will allow the opportunity to experiment with an Open Loop PWM Control System since the system cannot perform error detection and correction. We can thus demonstrate how the system output is dependent on external factors (i.e. OPTO SENSE).

In the second exercise (section 6.2.3) we will observe the system's response time characteristics via data logging, and an oscilloscope if available.

In the third exercise (section 6.2.4) we will develop the program modules necessary to close the loop: Error Signal Calculation and Display, Control Force Calculation and Modification. We will also enable additional command keys to select either Open Loop or Closed Loop control. We can thus demonstrate how Closed Loop Control eliminates the system output's dependence on external factors.

Then, in section 6.2.6, we will experiment with various aspects of Closed Loop Control. Access to an inexpensive oscilloscope will be helpful.



CONNECTIONS FOR PWM EXPERIMENT

FIGURE 6-10

2.1 Voltage Control Circuit

The circuit of Figure 6-10 is similar to that used in Section 4.2 for pulse width modulation, except that a capacitor is included to average the PWM signal. It is charged through the OPTO SENSE pot to introduce an external variable not controllable by the program. If this were to drive a load, an external amplifier would be needed, but we will not be concerned with that here. The capacitor voltage is measured both by the A/D converter and the voltmeter.

When Port 1A7 is high (output turned off) the capacitor is charged through the SENSE pot (R_p) the internal resistor R_1 , and the external resistor R_2 . It charges toward the V_{cc} (5 volt) supply, with a time constant of $R_s C$,

$$\text{where} \quad R_s = R_p + R_1 + R_2.$$

When Port 1A7 is set low, the capacitor discharges through R_1 , with a time constant $(R_2)C$. With the resistances shown the charging time constant ranges from 11 to 21 milliseconds, according to the pot setting; the discharge time constant is ten milliseconds. If we operate pulse width modulation with a cycle time of a few hundred microseconds the capacitor voltage will change only slightly during each cycle.

Using a linear approximation, the voltage increases during charging by:

$$(a) \quad \Delta v_c = \frac{t_c}{R_s C} (V_c - v)$$

where t_c = charging time
 $R_s C$ = charging time constant
 V_c = supply voltage (5 volts)
 v = capacitor voltage

When Port 1A7 is low the capacitor discharges by:

$$(b) \quad \Delta v_d = \frac{t_d}{R_2 C} (v - V_g)$$

where t_d = discharging time
 $R_2 C$ = discharging time constant
 V_g = voltage drop across the open collector buffer
 (0.4 volts)

Provided the pulse sequence is maintained for a time greater than several time constants, a steady state will be reached where the charging and discharging are equal in each cycle. Then:

$$(c) \quad \frac{t_c}{R_s C} (V_c - v) = \frac{t_d}{R_2 C} (v - V_g)$$

This equation can be solved for the steady state average voltage:

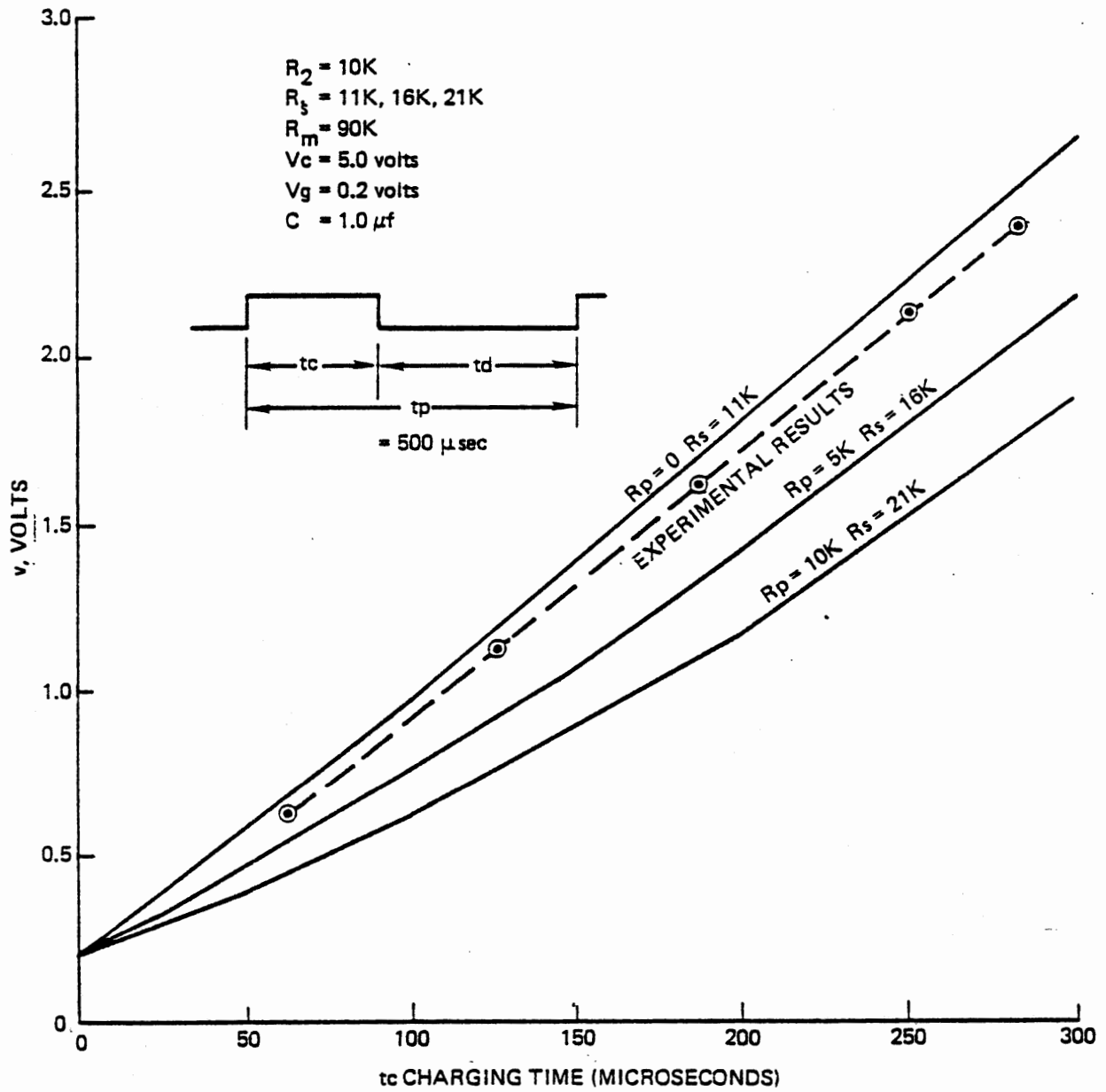
$$(d) \quad v = \frac{\frac{t_c}{R_s C} V_c + \frac{t_d}{R_2 C} V_g}{\frac{t_c}{R_s C} + \frac{t_d}{R_2 C}}$$

The value of the capacitor factors out of equation (d), showing that the steady state average voltage is independent of the capacitor. The equations above neglect the voltmeter, which drains a little current to ground. Accounting for the voltmeter resistance (R_m) leads to equation (e):

$$(e) \quad v = \frac{\frac{t_c V_c}{R_s} + \frac{t_d V_g}{R_2}}{t_c \left(\frac{1}{R_s} + \frac{1}{R_m} \right) + t_d \left(\frac{1}{R_2} + \frac{1}{R_m} \right)}$$

This equation is plotted in Figure 6-11 for a fixed total period of 500 microseconds and varying charging time, for three different values of potentiometer resistance. Note that with zero resistance in the potentiometer the voltage is nearly linear with charging time. The voltage is only moderately sensitive to the pot setting because of the fairly large value (10K) of R_2 , but the pot does cause a substantial departure from linearity. Figure 6-11 also shows experimental results generated with the program to be developed.

$$v = \frac{\frac{tc}{R_s C} V_c + \frac{td}{R_2 C} V_g}{tc \left(\frac{1}{R_s C} + \frac{1}{R_m C} \right) + td \left(\frac{1}{R_2 C} + \frac{1}{R_m C} \right)}$$



PWM Voltage - Fixed Period

Figure 6-11

With constant total period $t_p = t_c + t_d$ we can solve equation (e) for the ratio of charging time to total period required for any desired voltage.

$$(f) \quad \frac{t_c}{t_p} = \frac{v - \frac{R_m}{R_2 + R_m} V_g}{\left(\frac{R_m}{R_2 R_m}\right) \left[V_c \frac{R_2}{R_s} - V_g + \left(1 - \frac{R_2}{R_s}\right) v \right]}$$

If the pot is set to zero resistance we have the following values:

$$\frac{R_m}{R_2 + R_m} = \frac{90K}{10K + 90K} = 0.90$$

$$\frac{R_2}{R_s} = \frac{10K}{10K + 1K} = 0.91$$

$$V_c = 5.0$$

$$V_g = 0.2$$

Then:

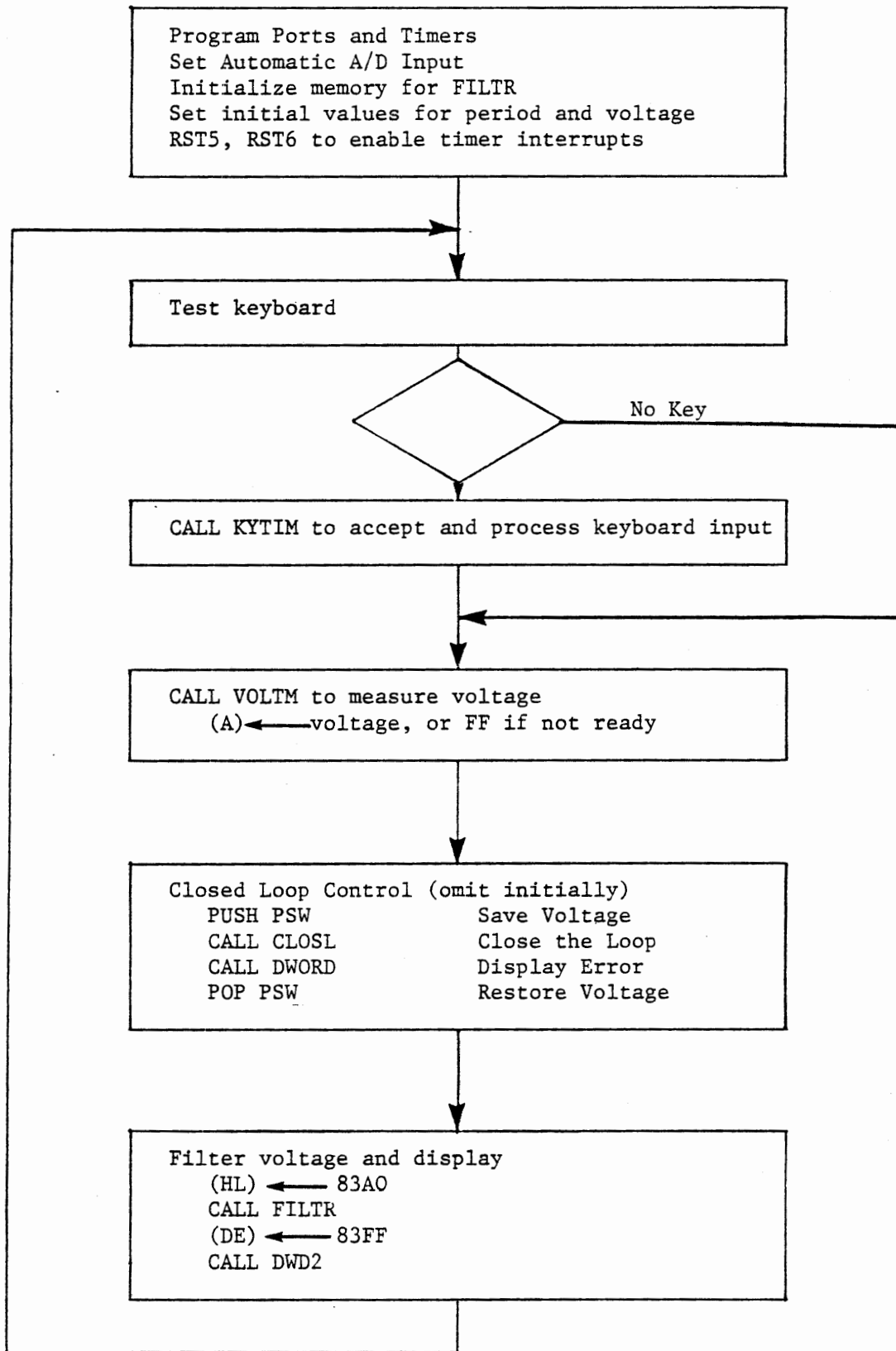
$$(g) \quad \frac{t_c}{t_p} = \frac{v - 0.18}{3.91 + 0.08v}$$

Since the .08v term in the denominator is quite small the relationship is nearly linear when the pot is set to zero, as shown in Figure 6-11. We will use this linear relationship in our pulse width modulation scheme.

THIS PAGE INTENTIONALLY LEFT BLANK

Timer 0 will be loaded with a count for the total period; at each interrupt from this counter Port 1A7 will be set high to start charging. Timer 1 will be loaded with a count for charging time, derived from the desired voltage. Timer 1 will be started when charging starts, and its interrupt will set Port 1A7 low to stop charging.

We will enter a desired voltage in hexadecimal with the least significant bit representing 10 millivolts, just as the A/D converter represents a voltage. Subtract 12 (representing 0.18 volts). This value will be used to determine the charging time in microseconds, so double it to account for two system clocks per microsecond, and load the result to Timer 1. The total period, loaded to Timer 0, should correspond to 3.91 volts or 391 microseconds. This is given by a hexadecimal value of 30E (= decimal 782). This value must be adjusted to compensate for resistor and voltage tolerances and the error caused by neglecting .08 v. The program provides for keyboard input of the total period so that the adjustment can be made while the program is running.



PWM VOLTAGE CONTROL - MAIN LOOP

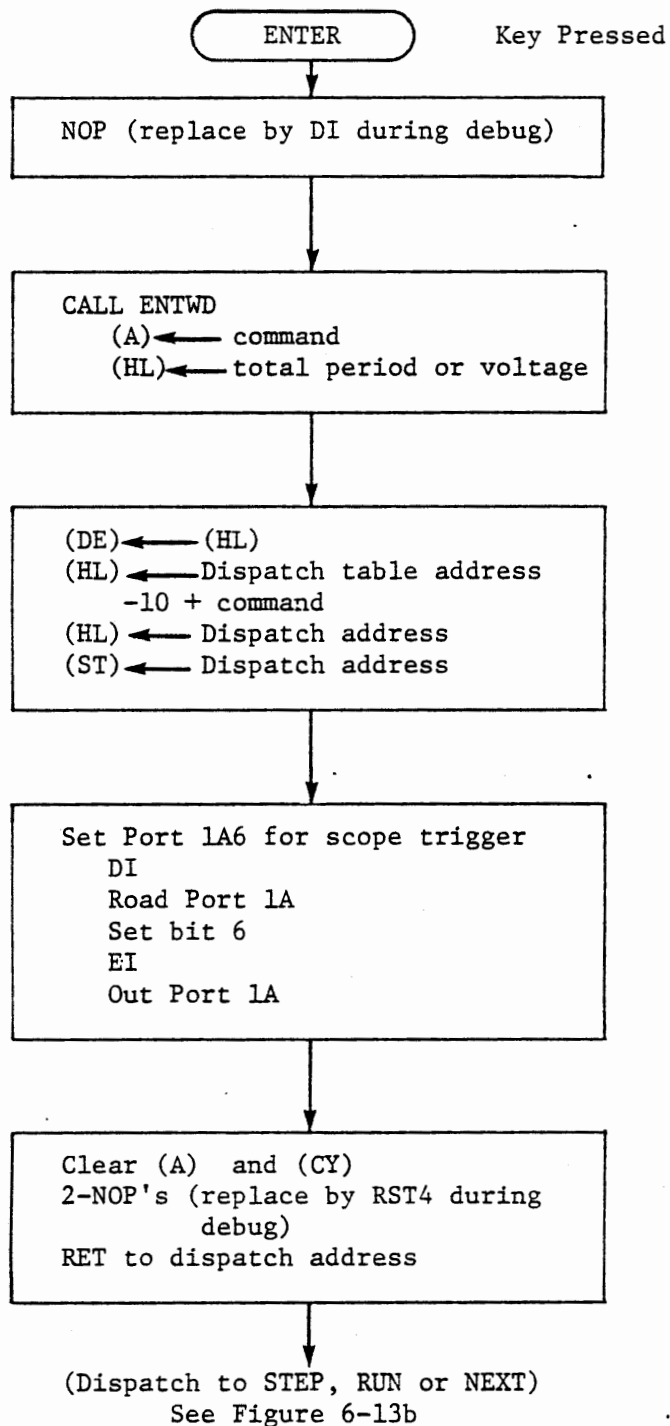
FIGURE 6-12

6.2.2 Voltage Control by PWM

EXERCISE:

In the program of Figure 6-12 we repetitively measure and display the voltage at the analog input, and test the keyboard for data entry. The voltage is determined by pulse width modulation under interrupt control; this is discussed in Section 6.2.2.4. The main loop in Figure 6-12 includes a call to a closed loop control subroutine that will be developed in Section 6.2.4. For developing the open loop program you can enter eight NOP instructions. The following sections discuss the three subroutines KYTIM, VOLTM, and FILTR.

As always, it is recommended that you study the problem and develop your own solution for each program module. This program is lengthy, and barely fits into 512 bytes of memory. Also, a number of revisions will be made as we develop the closed loop system. Therefore it is suggested that rather than coding your own solution you load the program given in Figure 6-17. With various modifications and additions we will use this program through the remainder of Chapter 6.



PWM VOLTAGE - SUBROUTINE KYTIM

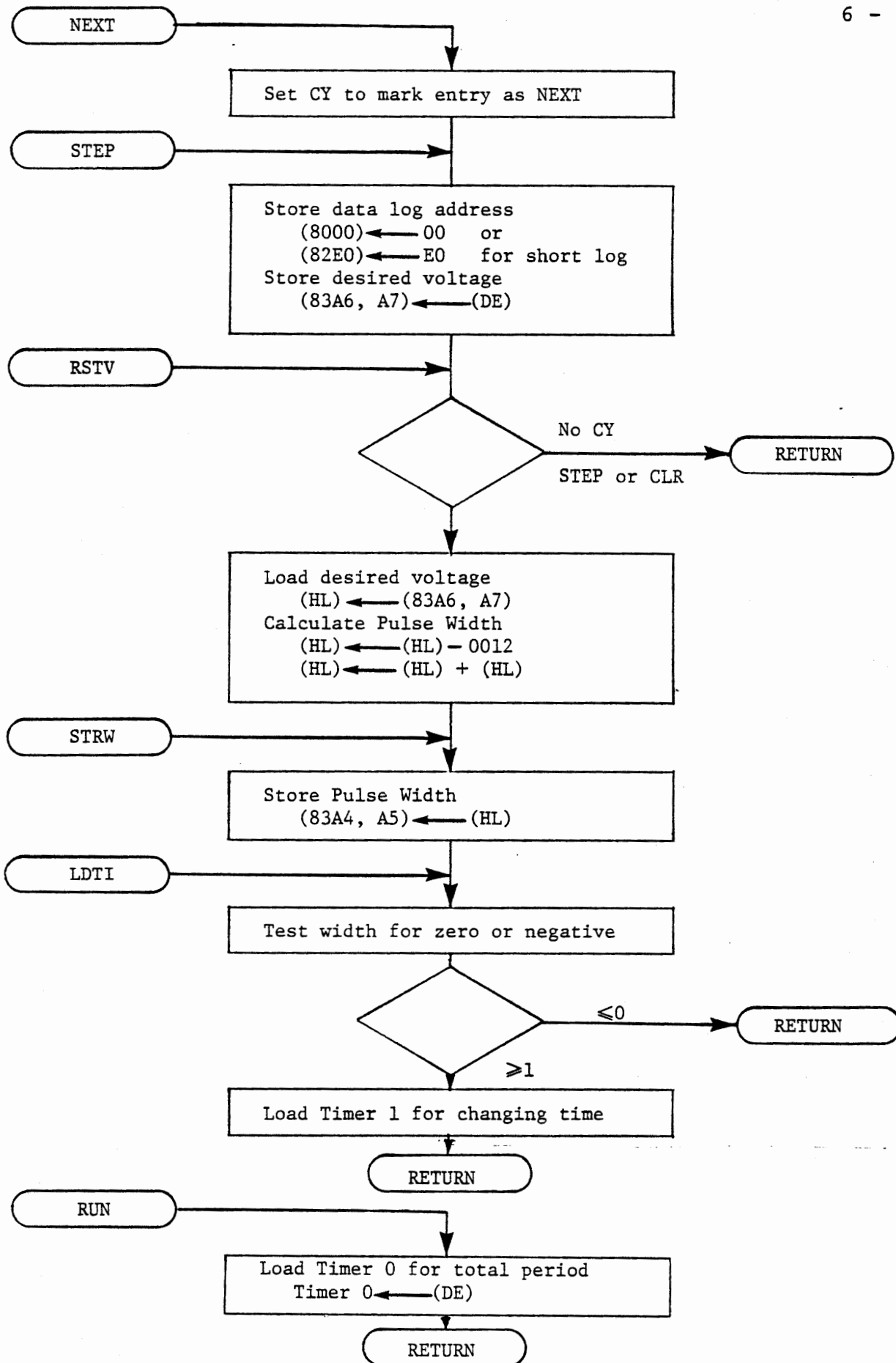
FIGURE 6-13a

.2.2.1 Data Entry Subroutine KYTIM

Subroutine KYTIM (Figure 6-13) is called for data entry when the main loop detects a key depression. KYTIM calls ENTWD (0346) to accept up to four hex keys (returned in HL) and a command (returned in A).

A dispatch table is used for distinguishing among the keys. Although it is not necessary to treat all keys differently at present, we will add some functions later. Immediately after looking up the dispatch address and before jumping to the process, Port 1A6 is switched high to give a scope trigger for use in observing the response time after a new voltage is keyed in. This output is available at a tie block, but must be pulled up through a resistor since it is an open collector buffer. Using any undefined key will generate the scope trigger without changing the pulse width or total period; the ADDR key will be reserved for this function.

Three debugging aids are included in the program design. A NOP before the call to ENTWD allows insertion of DI to eliminate the excessive delay in ENTWD in Step mode. A NOP before the RET instruction that dispatches to the processing module could be replaced by RST4 to call the monitor. Alternately a breakpoint can be placed at the RET instruction or at the first instruction of any key processing module. The EI instruction before OUT PORT1A will allow the monitor to interrupt before the RET is executed. Remember that one instruction after an EI is always executed before any interrupt can occur, and the monitor (STEP) interrupt cannot occur until one multiple memory access instruction (such as OUT) and one following instruction have both been



PWM KYTIM - SET PULSE WIDTHS

FIGURE 6-13b

executed.

The key processing modules are described below. In addition to their functions in response to command keys these can be called as independent subroutines by other program modules. Each such entry is identified either by the name of the key or some other name such as LDT1, "load Timer 1".

If the command is NEXT or STEP the input data represents a desired voltage. Before storing and processing this value, memory location 8000 is cleared for a data logging function to be introduced later. (If your MTS is not equipped with 1024 bytes of memory store E0 into location 82E0).

The desired voltage is then stored at (83A6, A7). For the data logging function it will be important that these steps occur in the sequence indicated.

The STEP key calls for the same processing as NEXT up to this point, but the following steps are omitted by a conditional return if carry is clear, indicating that the command was STEP. The STEP key is useless in the open loop system but is needed for closed loop control. The RSTV ("restore voltage") entry also reaches this conditional return. In a later modification of the program the BRK and CLR keys will jump to this entry after performing their other functions, and the conditional return will be executed for CLR but not for BRK.

Now the required pulse width is calculated. This is done in response to NEXT, and it will be done in response to BRK. Because of this

alternate entry the desired voltage is loaded into (HL) from the location (83A6, A7) where NEXT or STEP has stored it. Subtract 0012 to allow for the zero offset voltage Vz (0.18 volt) and double the result to give two clocks (one microsecond) for each ten millivolts.

The calculated pulse width is stored at (83A4, A5). This function is identified as a subroutine entry, STRW, which will be called in closed loop control. The following process is identified as subroutine entry LDT1, which will be called when we introduce proportional (not integral) control.

Before actually loading Timer 1 we check that a legitimate pulse width has been entered. The system will go out of control if a very long time is entered to Timer 1. This could occur in the closed loop system if a negative width were calculated, or in the open loop system if you request a voltage less than 12. The timer treats 0000 as a maximum delay, so we also test for this value. One way of making the test would be:

PUSH H	Save original value
DCX H	Force 0000 to FFFF
DAD H	High bit to carry
POP H	Restore original value
RC	Exit if zero or negative

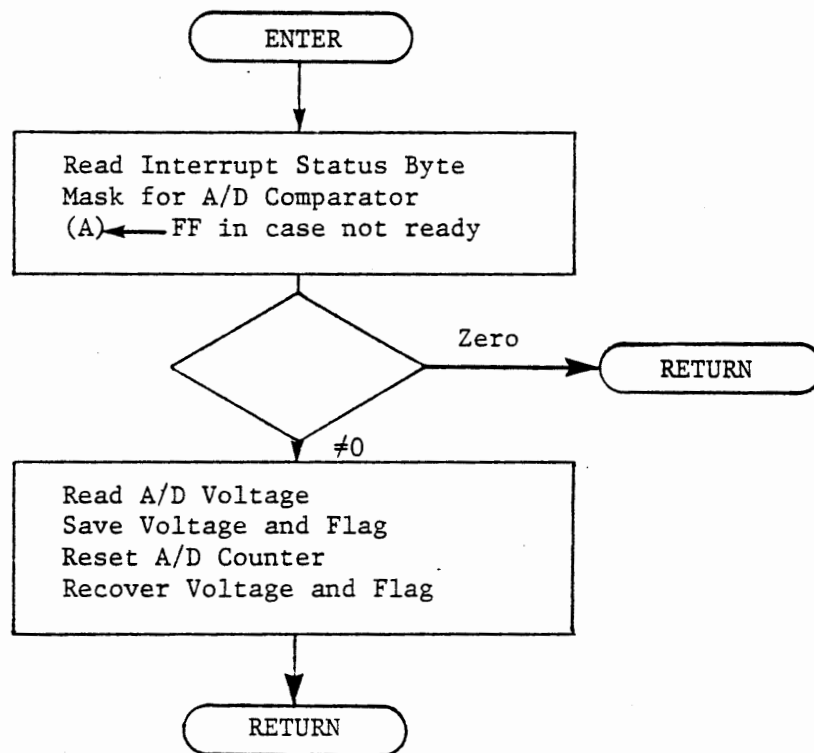
The above method has the virtue of changing only the carry flag.

Another method is:

DCX H	Force 0000 to FFFF
MOV A,H	Set all flags according
ORA A	to content of H.
INX H	Restore original value
RM	Exit if zero or negative.

This method, which the author has used, is faster and avoids using any stack area.

Provided that the test indicates a legitimate pulse width, timer 1 is loaded with the data. This sets the charging pulse width. The RUN key copies the input data from ENTWD to Timer 0, to set the total period.



VOLTMETER SUBROUTINE

FIGURE 6-14

2.2.2 Voltmeter Subroutine VOLTM

The automatic A/D converter is used in this program. This requires the following steps during initialization:

```
Program Port 1B for input
Set Port 1C1 high
Program and load timer 2 to divide the
    system clock by 8
```

Figure 6-14 shows the subroutine.

If the input voltage is greater than 2.55 volts it cannot be measured by the A/D converter because the comparator will always see the input greater than the D/A output. The comparator signal appears in bit 3 of the interrupt status byte. This is tested by `IN PORT2B, ANI 08`; if the bit is low the subroutine returns a value of FF with the zero flag set. If the comparator signal is high the voltage is read by `IN PORT1B`, and will be returned with the zero flag not set. Before return, however, the A/D counter must be reset by disabling the A/D interrupt, to start a new conversion.

Note that this subroutine will also return FF and Zero set if the voltage is less than 2.55, volts but insufficient time for the conversion is allowed between calls. In the program being developed, however, enough time is taken by the main loop and subroutines to ensure a valid conversion if the voltage is less than 2.55 volts.

6.2.2.3 Using FILTR

The voltage generated by pulse width modulation is averaged, or filtered, by the capacitor. It fluctuates by about 50 millivolts, so the voltage measured will vary in the less significant bits. To obtain a valid eight bit measurement we will use subroutine FILTR to provide additional filtering for display of the output.

FILTR is called with the raw (unfiltered) voltage in (A) and a memory address (83A0) in (HL). That memory location must be loaded with 1,2,3 or 4 during initialization, and the following two bytes must be cleared initially.

FILTR returns the input value (the raw voltage) in L, and the filtered voltage in both A and H. For display, load DE with 83FF and call DWD2 (02D4). This will display the data at the right hand side of the display, leaving your last keyed in data displayed at the left.

FILTR and DWD2 take most of the time needed for the A/D conversion. If you should choose not to use them you must provide about a one millisecond delay by other means, such as a call to the monitor subroutine DELAY (0236).

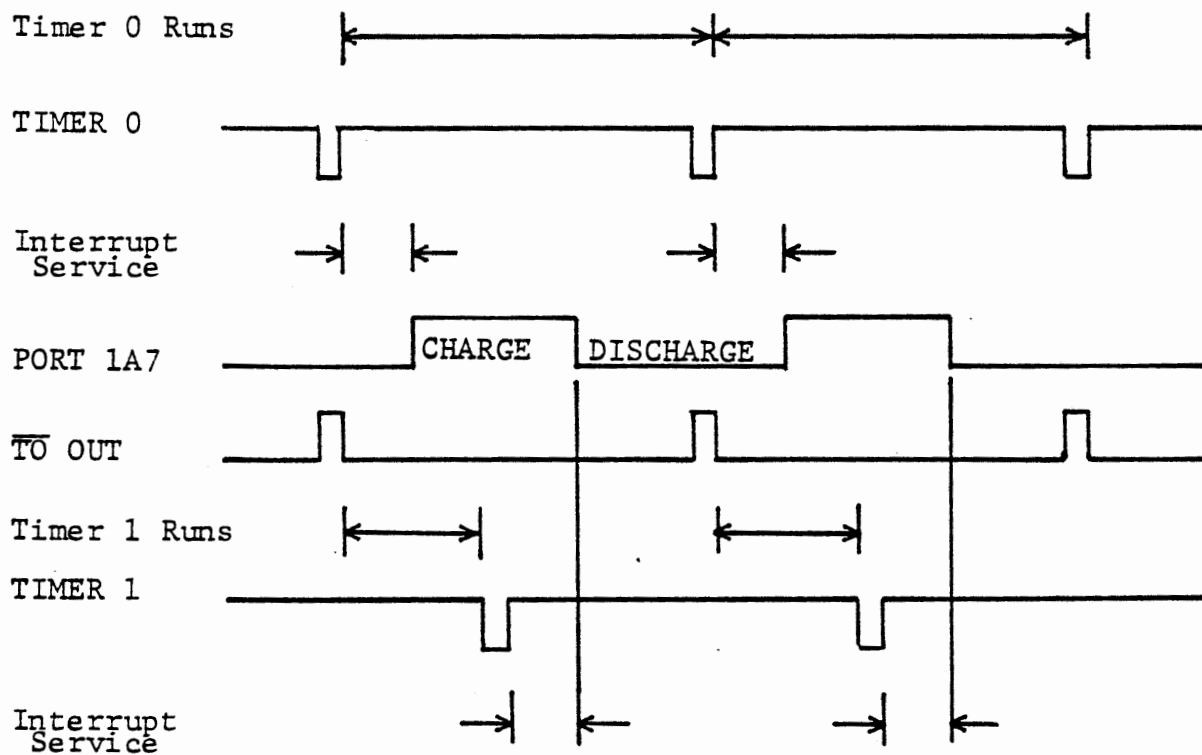
6.2.2.4 Timer Operation

Timer 0 is used to define the total period. It operates in mode 2 so that RST5 interrupts occur at precisely repeated intervals according to the value keyed in with RUN, and this period is repeated until a new value is entered. Interrupt service for RST5 sets Port 1A7 high

to start charging the capacitor. It also sets 1A5 low to clear the scope trigger.

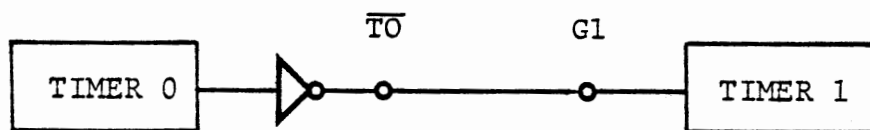
Timer 1 controls the charging time. While it is counting, Port 1A7 will be high so that charging can occur. At its terminal count, RST6 interrupt occurs and Port 1A7 is set low to start discharging the capacitor.

Timer 1 is used in mode 5, "Hardware Triggered Strobe". This mode of the 8253 interval timer was briefly described in Chapter 3 but has not been used in any previous exercise. In this mode the timer can be preloaded with a count value, and when a rising edge signal occurs at its gate input it starts counting. The timer output is normally high; it goes low for one clock period at the zero count and rises again to create an interrupt one clock time later. The timer then waits for another rising edge at its gate input, obtained from Timer 0 output.



Timer 0 in Mode 2

Timer 1 in Mode 5, triggered by Timer 0



PWM TIMER OPERATION

FIGURE 6-15

Figure 6-15 shows the timing relationship of Timer 0, Timer 1, and Port 1A7. Figure 6-16 shows the interrupt service routines. Note that both service routines should take the same length of time from the interrupt to the switching of Port 1A7.

Both mode 2 and mode 5 of the interval timer allow the timer to be loaded at any time. If the timer is counting, the present period is completed before the new time value is loaded. Therefore we can load these timers from the KYTIM subroutine without regard for their present states.

RST 5 Interrupt - Timer 0
Set Port 1A7 high to start charging Set Port 1A6 low to clear scope trigger Reenable Timer 0 Interrupt
Exit

RST6 Interrupt - Timer 1
Set port 1A7 low to start discharging
Reenable Timer 1 Interrupt
Exit

Note Ports 1A0 - 1A5 are not used in the system, so these may be set high or low as convenient.

PWM INTERRUPT SERVICE

FIGURE 6-16

.2.2.5 PWM Memory Allocation

You should develop a detailed flow diagram for the main program and write your own programs for MAIN, KYTIM, VOLTM, and the interrupt service routines. Subroutine FILTR was developed in Section 5.5. Remember to provide its initialization and to load (HL) with the required memory address.

The following memory assignments are used in the given solution:

8200 - 8227	Main - Initialize
8228 - 824F	Interrupt Service
8250 - 826F	Subroutine VOLTM
8270 - 82AF	Subroutine FILTR
82B0 - 82BF	Finish Initialization
82C0 - 82DF	Main Loop
8300 - 835F	Subroutine KYTIM
8360 - 839F	Subroutine CLOSL

Data memory assignments are:

83A0	value of M for FILTR (must be loaded with 1,2,3, or 4)
83A1,A2,A3	Used by FILTR (83A1,A2 must be cleared during initialization).
83A4,A5	Pulse width
83A6,A7	Desired voltage
83A8	Measured voltage

Additional memory assignments used for data logging if 1024 bytes of memory are available:

8000	Low byte of log address
8001 - 80FF	Data log

For a shorter data log with only 512 bytes of memory:

82E0	Low byte of log address
82E1 - 82FF	Data log

6.2.2.6 Debugging

Check that you have provided all of the proper initialization by comparing your program with the solution given in Figure 6-17a. Then step through all of the initialization procedure, including calls to special entries of KYTIM and the RST6 and RST5 programmed calls to interrupt service routines. (This is an added advantage to using those calls for enabling the interrupts, since it allows the monitor to operate through the service routine).

Since FILTR was developed in an earlier exercise it should need no debugging, except for checking that it has been loaded correctly.

To check the voltmeter subroutine (VOLTM) you should omit the RST5 and RST6 in the initialization procedure, so that interrupts will not be enabled. Connect a 10K resistor in parallel with the capacitor (from ANALOG IN to ground) to obtain a voltage within the A/D range. Enter a breakpoint at the start of the voltmeter subroutine, and press RUN. Step through the voltmeter section to test the program flow, also

observing the A register when the interrupt status byte is read and when the A/D input is read.

KYTIM should be checked with RST5 and RST6 still omitted. Enter a DI before CALL ENTWD and RST4 before the RET that jumps to the processing module. Run the program in STEP mode. Press a command key, and after the RST4 command is executed step through the KYTIM process for that command.

When all segments of the program have been checked restore RST5 and RST6, remove the DI and RST4, and operate the program in AUTO mode.

6.2.2.7 Program Operation

Start the program with an initial value of 0400 for period and C8 for voltage (2.00 volts). Turn the OPTO SENSE pot fully to the left for no resistance (highest voltage) and observe the average voltage with the voltmeter and on the display. The A/D input value varies in the less significant bits, because it senses the voltage at random points in the charge, discharge cycle. You can observe any single measurement by pressing an undefined key (e.g., ADDR). While the key is held down the measurements are stopped. Do this repeatedly and observe the range of voltage.

The voltage measured will be less than the requested 2.00 volts because the total period (500 microseconds) is too long. Gradually reduce the total period by keying in values less than 0400, followed by RUN. You should be able to obtain an accurate output of 2.00 volts, or C8 in the hexadecimal display. (If the display and

voltmeter do not agree, adjust the ANALOG IN pot to make the A/D measured voltage agree with the voltmeter). The total period needed will generally be less than the nominal value of 30E, principally because the supply voltage Vc at the terminal blocks will be less than 5.0 volts.

Enter different voltage requests and record the resulting output.

TOTAL PERIOD		RESULT	
VOLTAGE REQUEST			
DECIMAL	HEX	VOLTMETER A/D	
2.40	F0	_____	_____
2.00	C8	_____	_____
1.50	96	_____	_____
1.00	64	_____	_____

Find the lowest voltage request that reduces the output voltage. There is a lower limit to the charging pulse width, set by the time taken by Timer 0 interrupt service. Lower voltages can only be obtained by extending the total period.

Now request 2.00 volts again (C8) and alter the OPTO SENSE pot setting to reduce the output voltage to 1.50 volts (observed as 96 hex). Reduce the total period (entering values with the RUN key) to raise the output to 2.00 volts. With this setting again record the results for different voltage requests, and find the lowest value that can be

achieved.

TOTAL PERIOD		RESULT	
VOLTAGE REQUEST		VOLTMETER	A/D
DECIMAL	HEX		
2.40	F0	_____	_____
2.00	C8	_____	_____
1.50	96	_____	_____
1.00	64	_____	_____

The departure from linearity should be obvious. Any request lower than C8 will produce too low an output, and any request greater than C8 will produce too high an output. When we close the loop in Section 6.2.3 we will overcome this sensitivity to external conditions.

PWM VOLTAGE CONTROL - INITIALIZE

6-68

A D D R		CODE					
CODING SHEET	8 20	0	3E	MVI	A, 82		Program 8255#1
		1	82				Port IB In for A/D
		2	D3	OUT	CNT1		
		3	07				
		4	3E	MVI	A, 92		Program 8255#2
		5	92				
		6	D3	OUT	CNT2		
		7	0F				
		8	3E	MVI	A, 34		Program Timer 0
		9	34				Both bytes
MICROCOMPUTER TRAINING SYSTEM	A	D3	OUT	TIMCT			Mode 2
	B	17					Binary
	C	3E	MVI	A, 7A			Program Timer 1
	D	7A					Both bytes
	E	D3	OUT	TIMCT			Mode 5
	F	17					Binary
	8 21	0	3E	MVI	A, 94		Program Timer 2
		1	94				Low byte
		2	D3	OUT	TIMCT		Mode 2
		3	17				Binary
INTEGRATED COMPUTER SYSTEMS		4	3E	MVI	A, 08		Load Timer 2
		5	08				for ÷ 8
		6	D3	OUT	TIM2		
		7	16				
		8	21	LXI	H, 83A0		Initialize memory
		9	A0				for FILTR
		A	83				
		B	36	MVI	M, 04		M = 4
		C	04				
		D	AF	XRA	A		Clear 2 ^m EL
	E	23	INX	H		Address 83A1	
	F	77	MOV	M, A			
	8 22	0	23	INX	H		Address 83A2
		1	77	MOV	M, A		
		2	3C	INR	A		Set port I/O
		3	D3	OUT	PORTIC		for automatic
		4	06				A/D input
		5	C3	JMP	82B0		Jump past
		6	B0				interrupt service
		7	82				and FILTR
		8					FIGURE 6-17a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	0				INTERRUPT SERVICE
	1				
	2				
	3				
	4				
	5				
	6				
	7				
822	8	FS		PUSH PSW	RST5 - Timer 0
	9	3E		MVI A, 80	Set Port 1A7 high
	A	80			to start charging
	B	D3		OUT PORT 1A	All other bits low
	C	04			
	D	C3		JMP 8240	Jump past RST6
	E	40			
	F	82			
823	0	FS		PUSH PSW	RST6 - Timer 1
	1	3E		MVI A, 00	Set Port 1A7 low
	2	00			to start discharge
	3	D3		OUT PORT 1A	(all other bits also)
	4	04			Note identical timing
	5	3E		MVI A, 03	as in RST5
	6	03			
	7	D3		OUT CNT2	Reenable and clear
	8	0F			Timer 1 interrupt
	9	F1		POP PSW	Exit
	A	FB		EI	
	B	C9		RET	
	C	00		NOP	
	D	00		NOP	
	E	00		NOP	
	F	00		NOP	
824	0	3E		MVI A, 01	Reenable and clear
	1	01			Timer 0 interrupt
	2	D3		OUT CNT2	
	3	0F			
	4	F1		POP PSW	Exit
	5	FB		EI	
	6	C9		RET	
	7				
	8				

FIGURE 6-17b

CODING SHEET

8	25	0	DB	IN	PORT2B	Read interrupt status byte
		1	0D			
		2	E6	ANI	08	Mask for A/D Comparator
		3	08			
		4	3E	MVI	A, FF	If not ready return (A) = FF and Zero set
		5	FF			
		6	C8	RZ		
		7	DB	IN	PORT1B	If ready read A/D voltage
		8	05			
		9	F5	PUSH	PSW	Save voltage, flag
	A		3E	MVI	A, 06	Reset A/D counter but leave interrupt disabled
	B		06			
	C		D3	OUT	CNT2	
	D		0F			
	E		F1	POP	PSW	(A) ← voltage
	F		C9	RET		Not zero flag

MICROCOMPUTER TRAINING SYSTEM

8	0					
	1					
	2			SUBROUTINE VOLTM		
	3			MEASURES INPUT VOLTAGE		
	4			USING AUTOMATIC A/D		
	5					
	6			NO ENTRY DATA		
	7					
	8			RETURNS		
	9			(A) = VOLTAGE		
	A			NOT ZERO FLAG		
	B					
	C			IF A/D NOT READY		
	D			(A) = FF		
	E			ZERO FLAG SET		
	F					

INTEGRATED COMPUTER SYSTEMS

8	0			ALL OTHER REGISTERS		
	1			PRESERVED		
	2					
	3					
	4					
	5					
	6					
	7					
	8					

FIGURE 6-17c

SUBROUTINE FILTER

A D D R CODE

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	27	0	DS		PUSH	D			Save (DE)
		1	CS		PUSH	B			Save (BC)
		2	46		MOV	B, M			(B) ← m
		3	48		MOV	C, B			(C) ← m
		4	23		INX	H			Address $2^m E_i - 1$
		5	ES		PUSH	H			Save address
		6	5E		MOV	E, M			} (DE) ← $2^m E_i - 1$
		7	23		INX	H			
		8	56		MOV	D, M			} (HL) ← $2^m E_i - 1$
		9	6B		MOV	L, E			
		A	62		MOV	H, D			
827	B	29			DAD	H			} (HL) ← $2^{2m} E_i - 1$
	C	0D			DCR	C			
	D	C2			JNZ	827B			
	E	7B							
	F	82							
828	0	4F			MOV	C, A			(C) ← V_i
	1	7D			MOV	A, L			} (DE) ← (HL) - (DE) = $2^m(2^m - 1) E_i - 1$
	2	93			SUB	E			
	3	5F			MOV	E, A			
	4	7C			MOV	A, H			
	5	9A			SBB	D			
	6	57			MOV	D, A			} (HL) ← V_i
	7	69			MOV	L, C			
	8	26			MVI	H, 00			
	9	00							(HL) ← $(2^m - 1) E_i - 1$
	A	CD			CALL	SHFTN			Divide by 2^m
	B	A0							(DE) ← $(2^m - 1) E_i - 1$
	C	82							
	D	EB			XCHG				} (DE) ← $V_i + (2^m - 1) E_i - 1$ = $2^m E_i$
	E	19			DAD	D			
	F	EB			XCHG				
8	0				CONTINUED				AT 8290
	1								
	2				ENTER	(A) = V_i			(new value)
	3				(HL)	" = m			(filter constant)
	4				(HL) + 1	?			$2^m E_i - 1$
	5				(HL) + 2)			
	6				RETURN				
	7				(HL) + 3) = (A) = (H) = E_i			
	8					(L) = V_i			

FIGURE 6-17d.

FILTR continued and SHFTN

A D D R		CODE					
CODING SHEET	8 29 0	E3	XTHL				(HL) ← Address $2^m E_i$
	1	73	MOV	M, E			} Store $2^m E_i$
	2	23	INX	H			
	3	72	MOV	M, D			
	4	23	INX	H			Address E_i
	5	1B	DCX	D			To roundup only if
	6	CD	CALL	SHFTN			fractional part $> 1/2$
	7	A0					
	8	82					
	9	77	MOV	M, A			Store E_i
MICROCOMPUTER TRAINING SYSTEM	A	E1	POP	H			(L) ← V_i
	B	67	MOV	H, A			(H) ← E_i
	C	C1	POP	B			Restore BCDE
	D	D1	POP	D			
	E	C9	RET				Exit
	F	00	NOP				
	8 2A 0	48	MOV	C, B			SHFTN (C) ← m
	1	AF	XRA	A			Loop - clear carry
	2	7A	MOV	A, D			} Shift (DE) right to divide by 2
	3	1F	RAR				
4	57	MOV	D, A				
5	7B	MOV	A, E				
6	1F	RAR					
7	5F	MOV	E, A				
8	0D	DCR	C			Loop m times	
9	C2	JNZ	82A1			to divide by 2^m	
INTEGRATED COMPUTER SYSTEMS	A	A1					
	B	82					
	C	D0	RNC				Exit if LSB = 0
	D	13	INX	D			Else roundup
	E	7B	MOV	A, E			(A) ← less
	F	C9	RET				significant byte
	8	0					
	1						
	2						
	3						
4							
5							
6							
7							
8							

FIGURE 6-17e

PWM VOLTAGE - SET INITIAL VALUES

A D O R		CODE					
CODING SHEET	8	ZB0	11	LXI	D,0400	Set total period to 500 μ sec	
		1	00				
		2	04				
		3	CD	CALL	RUN	Run key function of KYTIM loads timer 0 for total period	
		4	3E				
		5	83				
		6	11	LXI	D,00C8	Set voltage to 2.00 volts	
		7	C8				
		8	00				
		9	CD	CALL	NEXT	Next key function of KYTIM calculates width, loads timer 1	
MICROCOMPUTER TRAINING SYSTEM	A	20				KYTIM calculates width, loads timer 1	
	B	83				Enable Timer 0	
	C	EF	RST5			Enable Timer 1	
	D	F7	RST6				
	E	00	NOP				
	F	00	NOP				
	INTEGRATED COMPUTER SYSTEMS	8	0				
			1				
			2				
			3				
		4					
		5					
		6					
		7					
		8					
		9					
	A						
	B						
	C						
	D						
	E						
	F						
	8	0					
		1					
		2					
		3					
		4					
		5					
		6					
		7					
		8					

FIGURE 6-17f

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

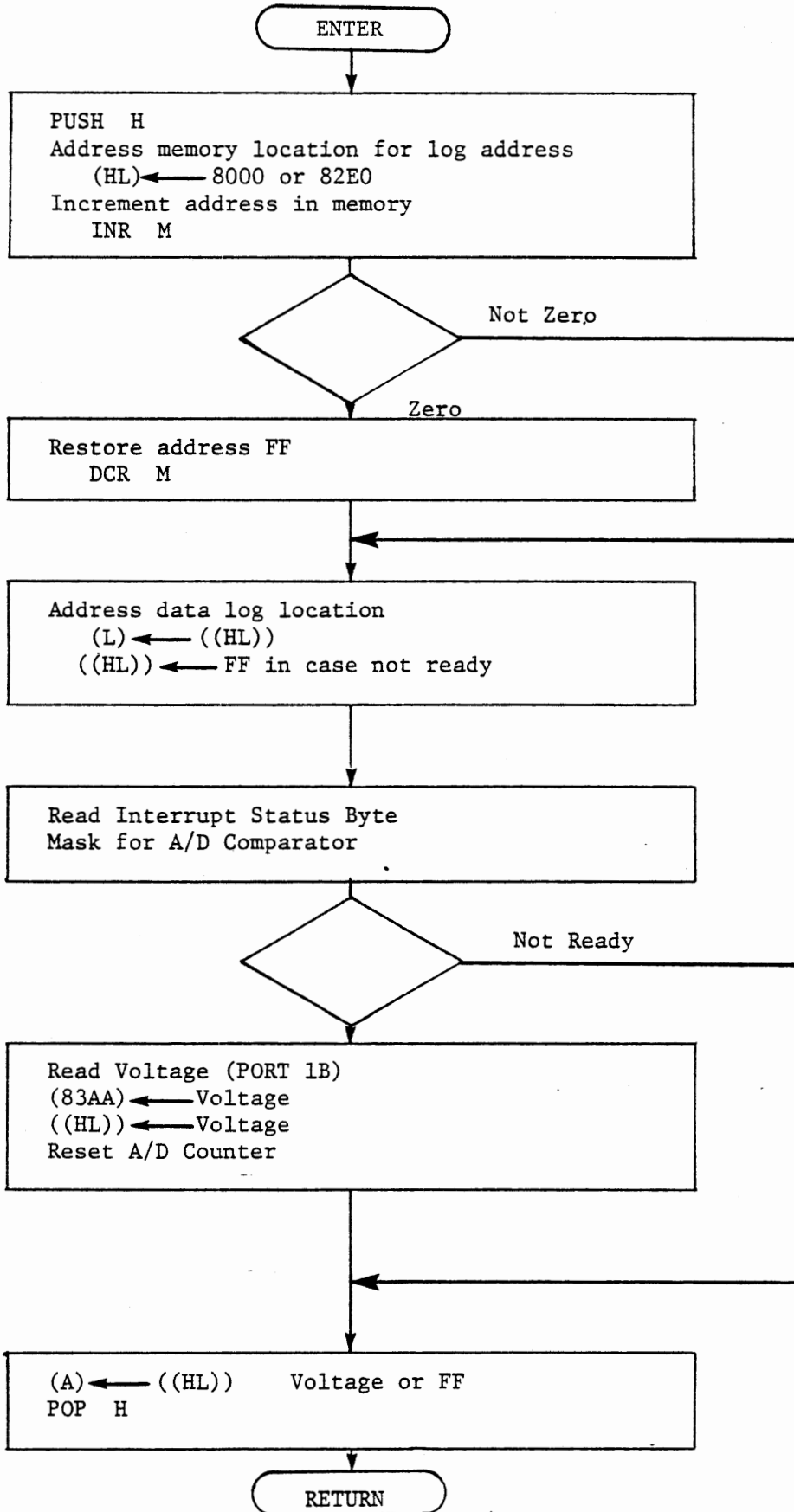
A	D	D	R	CODE	PWM	VOLTAGE	CONTROL	- MAIN	LOOP
8	2	C	0	DB	IN	PORT	OA	Read Keyboard	
			1	00				(FF if no key)	
			2	3C	INR	A			
			3	C4	CNZ	KYTIM		If Key pressed	
			4	00				call for data entry	
			5	83				and process	
			6	CD	CALL	VOLTM		Measure voltage	
			7	50				(A) ← A/D voltage	
			8	82				(FF if not ready)	
			9	00	NOP			Save 8 bytes	
			A	00				for closed loop	
			B	00				control	
			C	00					
			D	00					
			E	00					
			F	00					
8	2	D	0	00	V				
8	2	D	1	21	LXI	H, 83A0		Memory address	
			2	A0				for FILTER	
			3	83					
			4	CD	CALL	FILTR			
			5	70				(L) ← raw voltage	
			6	82				(H) ← filtered	
			7	11	LXI	D, 83FF		Display raw	
			8	FF				voltage (at right)	
			9	83				and filtered voltage	
			A	CD	CALL	DWD2			
			B	D4					
			C	02					
			D	C3	JMP	82C0		Loop	
			E	C0					
			F	82					
8			0						
			1						
			2						
			3						
			4						
			5						
			6						
			7						
			8						

FIGURE 6-17a

A D O R		CODE	P W M - S U B R				K Y T I M		
CODING SHEET	8 3 0 0	0 0	N O P					For DI during debug,	
		1	C D	C A L L	E N T W D			(A) ← command	
		2	4 6					(HL) ← total period	
		3	0 3					or voltage	
		4	E B	X C H G				(DE) ← data	
		5	2 1	L X I	H	8 3 0 8		Address dispatch	
		6	0 8					table - 10	
		7	8 3						
		8	8 5	A D D	L			Add command	
		9	6 F	M O V	L, A			} (ST) ← dispatch address	
	A	6 E	M O V	L, M					
	B	E 5	P U S H	H					
MICROCOMPUTER TRAINING SYSTEM		C	F 3	D I				} Set Port 1A6 high for scope trigger without changing port 1A7.	
		D	D B	I N	P O R T 1 A				
		E	0 4						
		F	F 6	O R I	4 0				
	8 3 1 0	4 0							
		1	F B	E I					
		2	D 3	O U T	P O R T 1 A				
		3	0 4						
		4	A F	X R A	A			Clear A and CY.	
		5	0 0	N O P					
	6	0 0	N O P				During debug RST4		
	7	C 9	R E T				or BRKPT here		
INTEGRATED COMPUTER SYSTEMS	8 3 1 8	1 7	M E M				Undefined		
		9	1 7	R E G			Undefined		
		A	1 7	A D D R			Trigger scope		
		B	2 1	S T E P			Store desired voltage		
		C	3 E	R U N			Set total period		
		D	2 0	N E X T			Store voltage, set width		
		E	1 7	B R K			Undefined		
		F	1 7	C L R			Undefined		
		8	0						
		1							
	2								
	3								
	4								
	5								
	6								
	7								
	8								

FIGURE 6-17 h

THIS PAGE INTENTIONALLY LEFT BLANK



LOGGING VOLTMETER

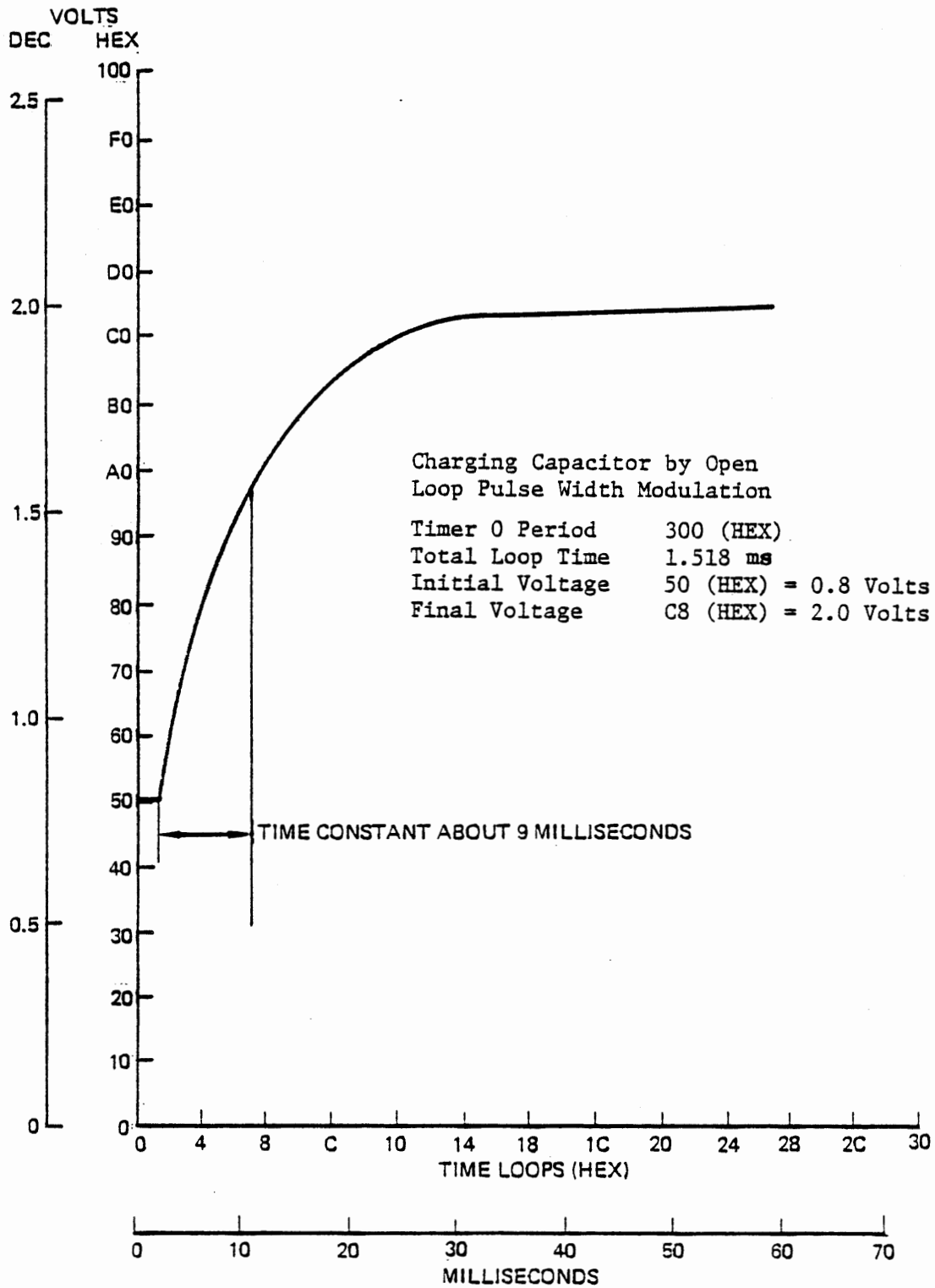
FIGURE 6-18

.2.3 Observing Response Time

If an oscilloscope is available, observe the response of the capacitor voltage to a new input. Trigger the oscilloscope from Port 1A6, which is set high when a key is pressed and set low when the next interrupt occurs. If you do not have an oscilloscope but do have 1024 bytes of memory you can observe the response by logging data. If you have only 512 bytes of memory you can log a limited amount of data.

Whenever a new voltage request is made through KYTIM we will start a new log, storing the voltage each time it is measured both at the fixed memory location 83AA and at the next available location in memory area 8001 - 80FF (or 82E1 - 82FF if only 512 bytes of memory are available). Memory location 8000 (or 82E0) stores the low byte of the log address. KYTIM starts the log by loading this memory byte with its own address (00 or E0). VOLTM increments the content of 8000 (or 82E0) and uses the incremented value as the low byte of the address for storing the measured voltage, as shown in Figure 6-18.

Since we are interested in the behavior of the system for a relatively brief period after the new voltage request is entered, and do not want to destroy those data with later measurements, the logging address is incremented only up to FF, and then remains fixed. This limit also allows the use of 82E1 - 82FF for a shorter data log, without writing into the program memory, with program changes only at the two places where this address is loaded to (HL), in KYTIM and VOLTM. Figure 6-19 gives the revised voltmeter program.



PWM - Open Loop Response

Figure 6-20

6.2.3.1 Program Usage

To use the log, set the desired total period (with RUN), enter a voltage (with NEXT) and then enter another voltage (with NEXT). Now press RST and review the data stored at 8000 - 80FF by entering ADDR 8000 and NEXT (or 82E0 and NEXT if w/o 1K option). Figure 6-20 is a plot of such data with a total period of 300 (hex), initial voltage 50 (hex) and a final voltage of C8. The curve shows the exponential charging of the capacitor toward 2.0 volts with an effective time constant of about 6 loop times or about 9 milliseconds.

A voltage has been recorded for each repetition of the main loop. These are not exactly equal intervals of time, principally because the number of interrupts during the main loop varies. With a Timer 0 period of 300 (hex) there will usually be four interrupts from each timer during the main loop, and occasionally one more, giving a total loop time of 1.49 to 1.59 milliseconds, and an average of 1.518. This value was used for the millisecond time scale in Figure 6-20.

The number of interrupts and the total loop time can be calculated from:

$$n = \frac{t_l}{t_p - t_i}$$

$$t_t = nt_p$$

where n = number of interrupts per loop

t_l = time for one pass through the main loop
and subroutines with no interrupts

t_p = total period of charge/discharge cycle
(loaded to Timer 0)

t_i = timer for processing interrupts
(RST5 plus RST6)

For the author's solution the values are given below.

t_l = 2092.3 clocks

t_i = 251.1 clocks

t_p = 768 clocks

n = 4.048 average interrupts/loop

t_t = 3108.7 clocks per loop

= 1.518 milliseconds per loop

6.2.4 Closing the Loop

EXERCISE:

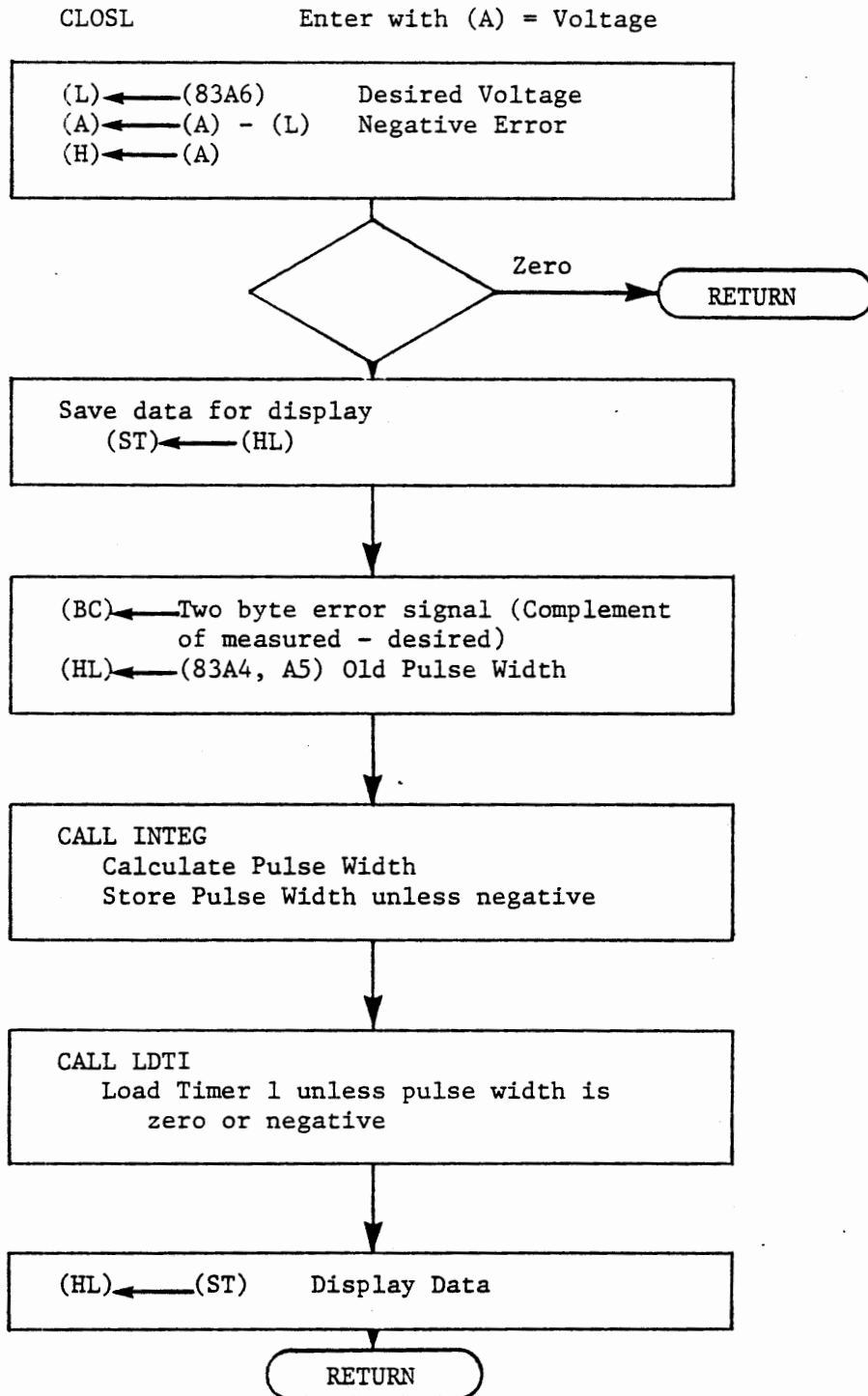
With the program of the preceding section we can generate a PWM voltage whose value is predictable if the circuit conditions are known. Open loop control is satisfactory in such a case. Changing the SENSE pot setting alters the resulting voltage and introduces an error, which can be corrected in either of two ways. We can change the mathematical model that relates pulse width to voltage, or we can simply adjust the pulse width to achieve the desired value. Note that in this system the total period (nominally 030E) represents the "mathematical model"; a correction to this value can approximate the intended relationship of two counts of pulse width equal to one count of voltage, although it cannot remove the non-linearity. Alternately, we can adjust the pulse width to some value different than twice the desired voltage. Either of these methods can be applied by a computer program in response to an observed difference between the measured voltage and the desired voltage. We used the first method by manual entry of new periods in the preceding exercise. Now we will apply the second method automatically.

.2.4.1 Error Signal Calculation

The error signal is the difference between the desired value and the measured value of the controlled variable. We obtain a measured value by reading the A/D input. Subroutine KYTIM has stored the desired value in memory at 83A6. (Only single byte values are meaningful now). The error signal is the desired value minus the measured value, which is positive when the measured value is too low, negative when it is too high. To adjust the driving force to correct the output we will add a positive error signal to the present charging pulse width, or subtract the magnitude of a negative error.

It turns out to be more convenient to subtract the desired voltage from the measured voltage, giving the complement of the error signal. Moreover, this seems to be a more meaningful value to display, being positive when the output is too high. Then we will take its twos complement for the correctly signed error signal to be added to the pulse width.

The error signal could range from -FF to +FF if the full voltage range of the A/D converter were available. Actually the range is somewhat less, but it is certainly greater than an eight bit value. The subtraction of measured voltage minus desired voltage gives a nine bit result in A and CY, with CY representing the sign. We will display only the eight bit value, since most of the time the sign will be obvious. For the calculation, however, we will convert it to its two byte twos complement.



PWM SUBROUTINE CLOSL

FIGURE 6-21a

This can be done by:

CMA	complement magnitude
MOV C,A	(C) <--- magnitude byte
CMC	complement sign
SBB A	(A) <--- 00 or FF
MOV B,A	(B) <--- sign byte
INX B	increment for two's complement

Now the properly signed error signal can be added to the pulse width (loaded into HL) by DAD B, giving a new pulse width.

When no error exists the pulse width will be constant; a positive error will increase the width and therefore the voltage; a negative error will decrease the width. Since the error signal is added into the steady state force, we have integral control. In Section 6.2.7 we will discuss the relationship between this simple control system and the integral control equation. First, however, we will develop subroutine CLOSL to perform the calculation and control the pulse width, and we will observe the results. CLOSL is to be located at 8360 - 839F, and will be called after the return from VOLTM with (A) = measured voltage. You should now complete the main loop according to Figure 6-12. Note that CLOSL is to return data in (HL) for display by DWORD. FILTR needs the voltage returned by VOLTM, so the main loop saves that value by PUSH PSW before the call to CLOSL and recovers it before calling FILTR.

CLOSL is specified in Section 6.2.4.2 and shown in Figure 6-21a.

CLOSL itself calculates the error signal and loads the old pulse width. It calls another subroutine, INTEG, to calculate the new pulse width. Then CLOSL calls LDT1 (a module of KYTIM) to load Timer 1, provided that the pulse width is positive and greater than zero.

INTEG calculates the new pulse width, in this version, simply by adding the error signal to the old width. It tests for a negative result and stores the pulse width (which is the integral of errors) if it is positive. (A zero integral is permitted, although zero is forbidden to be loaded to the timer).

The specification for INTEG in Section 6.2.4.3 states requirements that are to be met by both versions of INTEG that will be developed.

.2.4.2 Subroutine CLOSL Specification

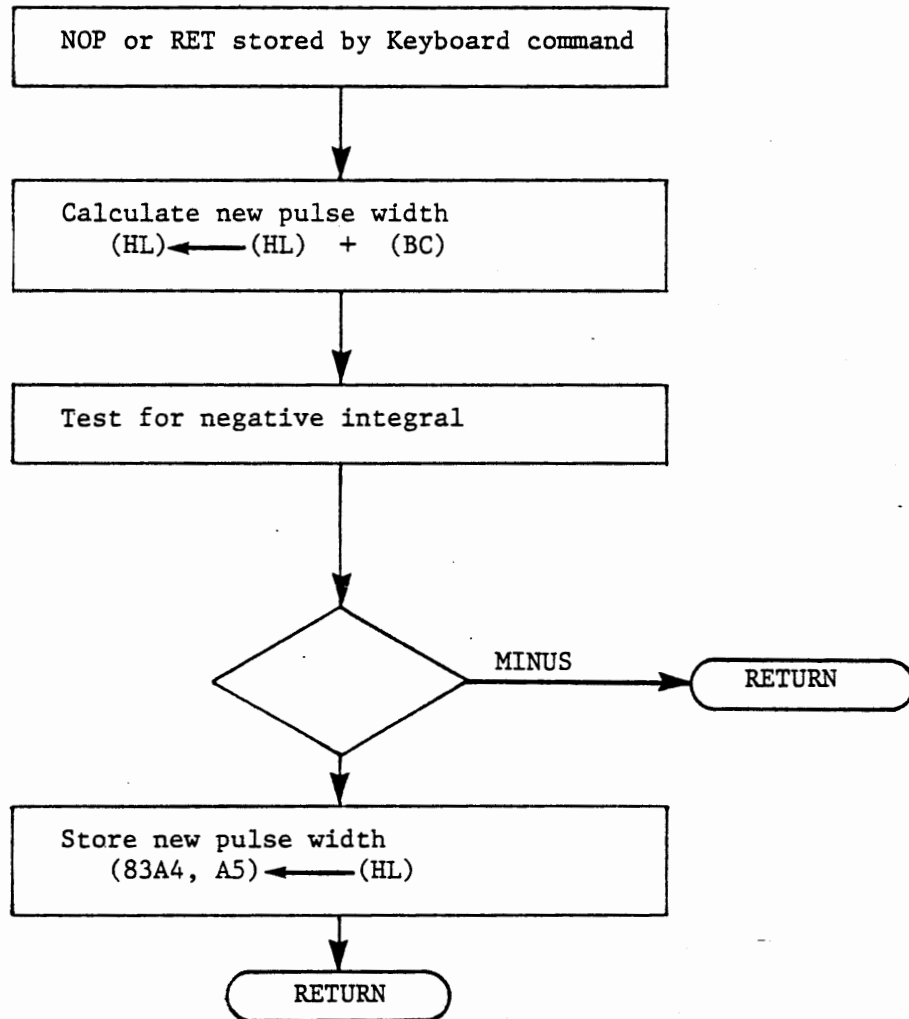
Function Calculate the error signal and set a new pulse width
Return negative error signal and desired voltage
ready for display by DWORD.

Enter (A) = Measured Voltage
(83A6) = Desired Voltage
(83A4,A5) = Old Pulse Width

All registers are used

Calls INTEG for pulse width calculation
LDT1 to load Timer 1

INTEG Called by CLOSL
 (BC) = Error Signal
 (HL) = Old Pulse Width



PWM SUBROUTINE INTEG

FIGURE 6-21b

.2.4.3 Subroutine INTEG Specification

Function Calculate new pulse width.

If result is positive (greater than zero)
store result.

Enter (BC) = Error Signal

(HL) = Old Pulse Width

Return (HL) = New Pulse Width

(83A4,A5) = New Pulse Width

Alternate Returns

If new pulse width is less than, or equal to zero, return without storing result.

Provision is made for insertion by keyboard control of a NOP or RET instruction at the entry to INTEG. The RET will disable closed loop control.

THIS PAGE INTENTIONALLY LEFT BLANK

.2.4.4 Additional Command Keys

We will now define four additional command keys to be processed by KYTIM:

REG - Force output low temporarily for observing response.
(Dispatch to 8350).

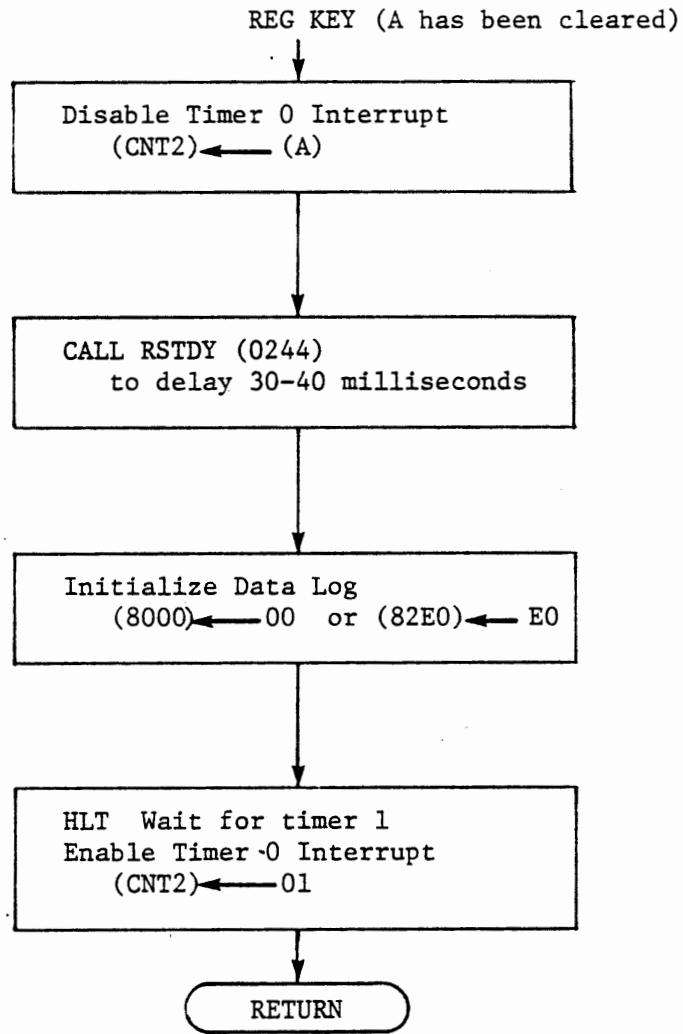
MEM - Store data to be used by a subsequent version of CLOSL. Store the data returned by ENTWD into memory locations 83A8,A9.
(Dispatch to 8345).

BRK - Set open loop operation. (Dispatch to 834A).

CLR - Set closed loop operation. (Dispatch to 834C).

The change between open and closed loop operation will be accomplished by modifying subroutine INTEG. In response to CLR, store a NOP instruction at 8378 to allow INTEG to complete its functions. In response to BRK, store a RET instruction at 8378 to cause an immediate exit from INTEG.

(All of the addresses above refer to the author's solution. Your program may require different addresses).



REG MODULE OF KYTIM

FIGURE 6-22

We will be interested in observing the response of the closed loop control system to a disturbance. This can be done by oscilloscope observation or by logging data. For convenience in using the oscilloscope the REG key will force a low output for long enough to discharge the capacitor. This is done by disabling the timer 0 interrupt and calling a monitor subroutine to generate the delay. At return from the delay start a new data log (as in NEXT and STEP), wait for a Timer 1 interrupt, and then re-enable and clear the Timer 0 interrupt. (See Figure 6-22).

Monitor subroutine RSTDY (located at 0244) is the delay function in GETKY. It repeatedly scans the keyboard, and returns after 30 milliseconds provided no key has been pressed. (This delay time is extended to about 35 milliseconds here by the Timer 1 interrupts).

Note that Timer 0, operating in mode 2, continues to run and reload itself even though its interrupt is disabled. Its output repeatedly triggers Timer 1 as in normal operation. The first interrupt from Timer 1 sets Port 1A7 low; it is not set high again until Timer 0 is enabled after the delay. An HLT instruction before enabling Timer 0 causes the first charge/discharge cycle after the delay to have its normal timing.

The main loop and subroutines KYTIM, CLOSL, and INTEG are given in Figure 6-23. Locations 8200 through 82BF are unchanged, and 82C0 - 8344 require changes only where marked * . The additions to KYTIM and the new subroutines CLOSL and INTEG are located in 8345 through 837F.

A D D R CODE PWM VOLTAGE CONTROL - MAIN LOOP

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	2C0	DB		IN	PORTOA	Read Keyboard (FF if no key)
	1	00				
	2	3C		INR	A	
	3	C4		CNZ	KYTIM	If Key pressed
	4	00				call for data entry
	5	83				and process
	6	CD		CALL	VOLTM	Measure voltage
	7	50				(A) ← A/D voltage
	8	82				(FF if not ready)
	9	F5	*	PUSH	PSW	Save voltage
	A	CD	*	CALL	CLOSL	Close the loop
	B	60	*			(L) ← desired voltage
	C	83	*			(H) ← error signal
	D	CD	*	CALL	DWORD	Display error
	E	D1	*			signal (at left)
	F	02	*			and desired voltage
8	2D0	F1	*	POP	PSW	(A) ← measured voltage
8	2D1	21		LXI	H, 83A0	Memory address
	2	A0				for FILTR
	3	83				
	4	CD		CALL	FILTR	
	5	70				(L) ← raw voltage
	6	82				(H) ← filtered
	7	11		LXI	D, 83FF	Display raw
	8	FF				voltage (at right)
	9	83				and filtered voltage
	A	CD		CALL	DWD2	
	B	D4				
	C	02				
	D	C3		JMP	82C0	Loop
	E	C0				
	F	82				
8	0					
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					

FIGURE 6-23a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	30	0	00		NOP					for DI during debug.
		1	CD		CALL	ENTWD				(A) ← command
		2	46							(HL) ← total period
		3	03							or voltage
		4	EB		XCHG					(DE) ← data
		5	21		LXI	H, 8308				Address dispatch
		6	08							table -10
		7	83							
		8	85		ADD	L				Add command
		9	6F		MOV	L, A				} (ST) ← dispatch address
	A		6E		MOV	L, M				
	B		E5		PUSH	H				
		C	F3		DI					} Set Port 1A6 high for scope trigger without changing port 1A7.
		D	DB		IN	PORT1A				
		E	04							
		F	F6		ORI	40				
8	31	0	40							
		1	FB		EI					
		2	D3		OUT	PORT1A				
		3	04							
		4	AF		XRA	A				Clear A and CY.
		5	00		NOP					
		6	00		NOP					During debug RST4
		7	C9		RET					or BRKPT here
	831	8	45	*	MEM					
		9	50	*	REG					
		A	17		ADDR					Trigger scope
		B	21		STEP					Store desired voltage
		C	3E		RUN					Set total period
		D	20		NEXT					Store voltage, set width
		E	4A	*	BRK					Set open loop
		F	4C	*	CLR					Set closed loop
8		0								
		1								
		2								
		3								
		4								
		5								
		6								
		7								
		8								

FIGURE 6-23b

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	3	6	0	2A	LHLD	83A6	CLOS	
			1	A6				(L) ← desired voltage
			2	83				At entry (A) = measured
			3	95	SUB	L		(A) ← - error signal
			4	67	MOV	H, A		(H) ← - error signal
			5	C8	R _E			Exit if error = 0
			6	E5	PUSH	H		Save display data
			7	2F	CMA			} (BC) ← + error as two bytes
			8	4F	MOV	C, A		
			9	3F	CMA			
			A	9F	SBB	A		
			B	47	MOV	B, A		
			C	03	INX	B		
			D	2A	LHLD	83A4		(HL) ← old pulse width
			E	A4				
			F	83				
8	3	7	0	CD	CALL	INTEG		Calculate and store new width
			1	78				
			2	83				
			3	CD	CALL	LDT1		Load new width to timer 1
			4	32				
			5	83				
			6	E1	POP	H		Recover display data
			7	C9	RET			
8	3	7	8	00	NO P / RET			INTEG (RET open loop)
			9	09	DAD	B		(HL) ← old width + error
			A	7C	MOV	A, H		Test for negative integral.
			B	B7	ORA	A		
			C	F8	RMINUS			Exit if negative
			D	22	SHLD	83A4		Store integral if positive
			E	A4				
			F	83				
8	3	8	0	C9	RET			
			1					
			2					
			3					
			4					
			5					
			6					
			7					
			8					

FIGURE 6-23e

.2.5 Closed Loop Operation

With closed loop control the program will force the output voltage to equal the requested voltage. You can enter a voltage with the NEXT key, which calculates a new pulse width, or with STEP, which does not. In either case the program will adjust the duty cycle to generate the requested voltage. The voltage will now be independent of the total period and the OPTO SENSE pot setting. It will fluctuate over a range of about six counts (60 millivolts), from C5 to CB. The fluctuation is displayed in the measured voltage (at the right) and the error signal (at the left). These will be changing so rapidly as to be unreadable. Since the measurement and display are handled in the main loop, pressing a key will stop the measurements and allow you to read the voltage. By doing this repeatedly you can observe the range of measurements. The ADDR key will do this without changing any stored data or controls. The fluctuation of the voltage is an inherent and undesirable effect of closed loop control. Fortunately it can be reduced by several means that we will investigate later.

Perform the experiments described in the following sections. Results of these experiments can be observed with your voltmeter. The experiments of Section 6.2.6 require either an oscilloscope or the laborious task of plotting data from the log made by the VOLTM subroutine.

6.2.5.1 Seeking Desired Voltage

Enter the following data and commands:

```

CLR           Set closed loop control

400,RUN      Set 0.5 millisecond period

C8,STEP      Set 2.0 volts

```

Observe the filtered voltage in the second pair of digits from the right, and observe the voltmeter reading. They should agree closely. Adjust the ANALOG IN pot, and observe that this now changes the actual output as observed on the voltmeter, because the closed loop control forces its measured input voltage to equal the requested voltage. Now enter the following voltages and observe the resulting output.

Total Period 0.5 ms (400 hex)

Voltage Request		Result	
Decimal	Hex	Voltmeter	A/D
2.40	F0	_____	_____
2.00	C8	_____	_____
1.50	96	_____	_____
1.00	64	_____	_____

6.2.5.2 External Resistance Variation

Enter the following data and commands:

CLR Set closed loop control
 400,RUN Set 0.5 millisecond period
 C8,STEP Set 2.0 volts

Adjust the OPTO SENSE pot over its full range from left to right and back to the left again. There should be no appreciable change in the voltmeter reading.

BRK Set open loop control

Adjust the SENSE pot to reduce the voltage to 1.50 volts (96 hex).

CLR Set closed loop control

Request the several voltages again and observe the results.

Total Period 0.5 ms (400 hex)

Voltage Request		Result	
Decimal	Hex	Voltmeter	A/D
2.40	F0	_____	_____
2.00	C8	_____	_____
1.50	96	_____	_____
1.00	64	_____	_____

The results should be essentially the same as before.

6.2.5.3 Total Period Variation

Return the SENSE pot to the full left position. Enter:

```
CLR          Set closed loop control
400,RUN      Set 0.5 millisecond period
C8,STEP      Set 2.0 volts
```

Now enter different total periods and observe the results.

Period		Voltage	
Milliseconds	Hex	Voltmeter	A/D
0.375	0300	_____	_____
0.50	0400	_____	_____
0.75	0600	_____	_____
1.00	0800	_____	_____
2.00	1000	_____	_____
32.00	0000	_____	_____

Here a greater difference between the voltmeter and the A/D converter may occur because of the wide variation in the voltage within a charge/discharge cycle. With the two millisecond period the capacitor charges and discharges by 300 millivolts in each cycle. At 32 milliseconds the voltage is actually swinging from 0.6 to 3.6 volts, but the average is held to 2.1 volts by closed loop control.

After the 32 millisecond test enter 400,RUN. The voltage will rise very nearly to 5.0 volts, because the charging time will have been set

to about 15 milliseconds, much longer than the newly entered total period. Since the A/D converter cannot measure the off-scale voltage it returns FF. CLOSL calculates an error signal of $C8 - FF = -37$ and repeatedly reduces the pulse width by this amount until the voltage returns to a value within range, and thereafter adjusts the pulse width appropriately.

6.2.5.4 Response to Disturbance

Enter the following data and commands:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts
REG	Force voltage low.

The output voltage will momentarily drop to about 0.2 volt and rise again to the requested voltage. We will observe this response in detail in section 6.2.6. The voltmeter will show the drop, but it will not be fast enough to go down more than a few tenths of a volt before closed loop control resumes and restores the desired voltage.

6.2.5.5 Open Loop Operation

The open loop control program is able to maintain a voltage very well, but is unable to compensate for changes in external conditions, as we observed earlier. Once an appropriate pulse width has been set by the closed loop system we can open the loop and the voltage will remain essentially constant. Enter these data and commands:

CLR	Set close loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts
REG	Force voltage low
BRK	Set open loop control
REG	Force voltage low

Observe that the open loop system returns to the requested voltage once an appropriate pulse width has been set by closed loop control. Now with open loop operation, adjust the SENSE pot to obtain 1.50 volts. Now press:

CLR	Set closed loop control
BRK	Set open loop control
REG	Force output low

Again closed loop control has set the pulse width and open loop control can restore the voltage after a disturbance. This is sometimes an acceptable mode of operation for a control system where external variables change slowly. Closed loop control can be invoked periodically, or when conditions are known to have changed. After the necessary adjustments have been made to the control force an open loop system can maintain the operation.

Students who are not especially interested in closed loop control may want to skip the remainder of Chapter 6. It is concerned with methods of improving the performance of a closed loop control system.

6.2.6 Closed Loop Response

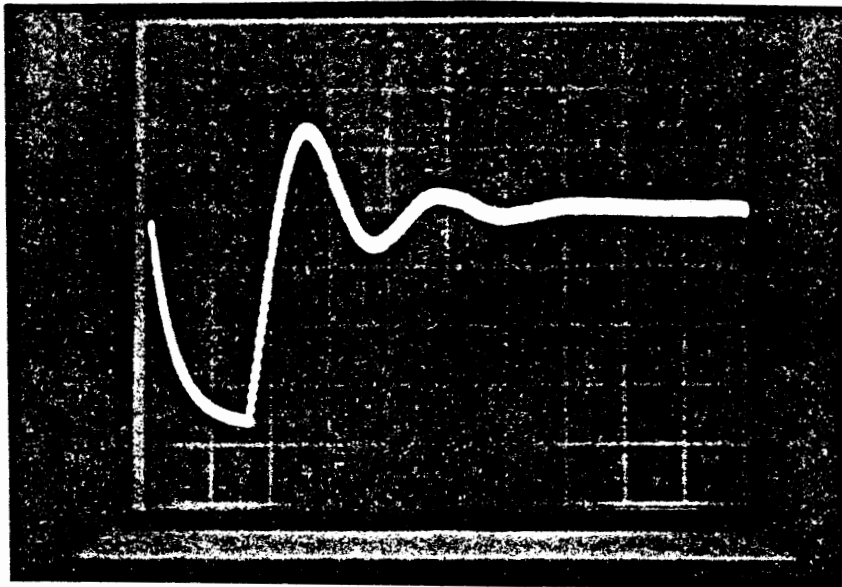
Observing the response to a disturbance under various conditions demonstrates very important features of closed loop control systems. This can be done most conveniently with an oscilloscope, or data can be logged and plotted. Section 6.2.6.1 describes the use of the oscilloscope and shows open and closed loop waveforms. Section 6.2.6.2 presents closed loop results obtained by the data log, and discusses the waveforms. The effect of changing the total period is shown in 6.2.6.3.

6.2.6.1 Oscilloscope Observation

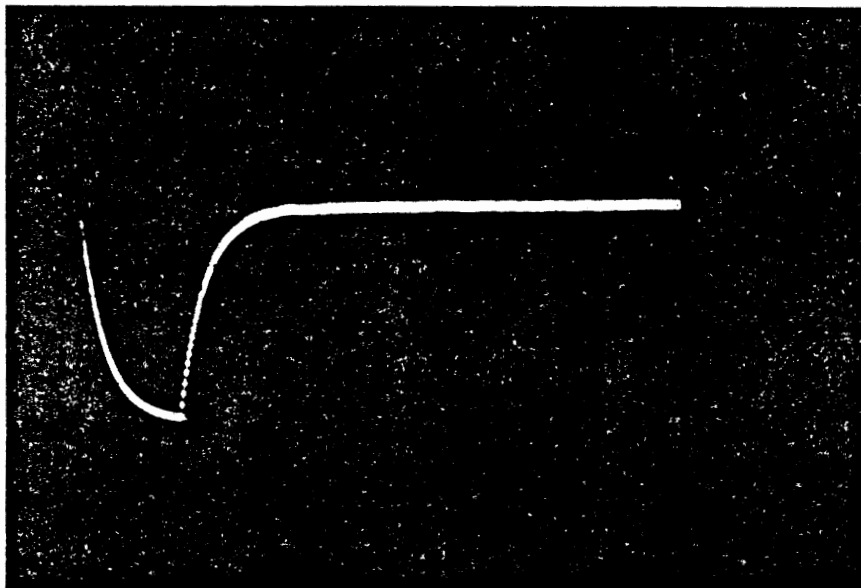
The oscilloscope is to be triggered by Port 1A6, which is set when a command key has been pressed and released. The capacitor voltage is to be observed.

Use a time scale of 20 milliseconds per division and a voltage scale of 0.5 volts/division. Enter the following:

CLR	Set closed loop control
400,RUN	Set 0.5 millisecond period
C8,STEP	Set 2.0 volts



CLOSED LOOP RESPONSE



OPEN LOOP RESPONSE

SCALES: 015 VOLT/DIV 20 MS/DIV

OPEN AND CLOSED LOOP WAVEFORMS

FIGURE 6-24

Test the scope triggering by repeatedly pressing ADDR. A single sweep should occur each time you release the key. This may take some adjustment of the oscilloscope trigger controls.

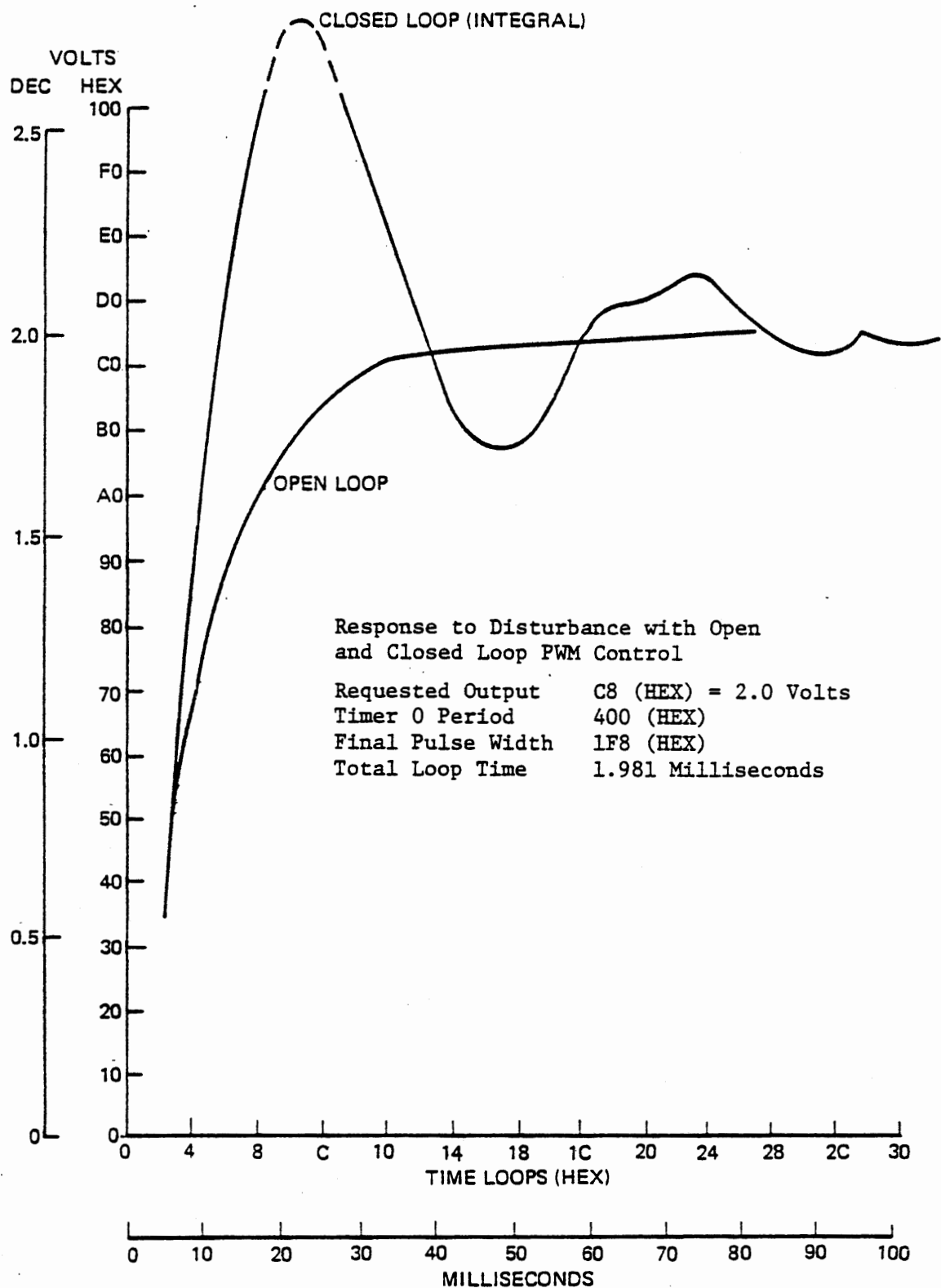
When you press and release REG the output will be forced low for about 35 milliseconds and then closed loop control will be resumed. The oscilloscope should display a waveform similar to the upper photograph in Figure 6-24. Now:

BRK Set open loop control

REG Force output low

A waveform similar to the lower photograph in Figure 6-24 should be seen.

On repeated operations of REG with open loop control the waveform should be very consistent. It merely shows the charging of the capacitor in response to a constant duty cycle PWM voltage. In closed loop control there will be substantial variations in the waveform, principally because of the random relationship between the time that the A/D conversion is completed and the time that CLOSL acts on the result. The reasons for the waveshape are discussed in the next section.

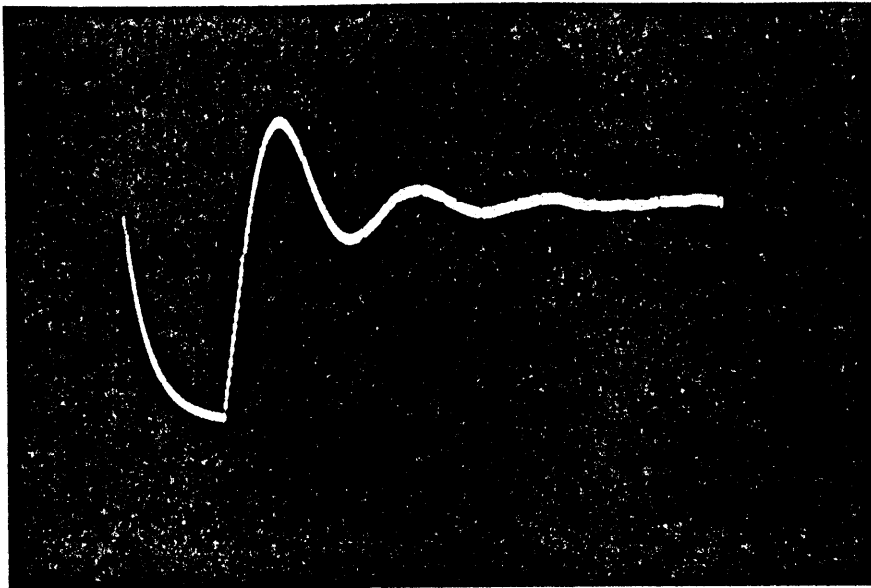


Closed Loop Response Waveform

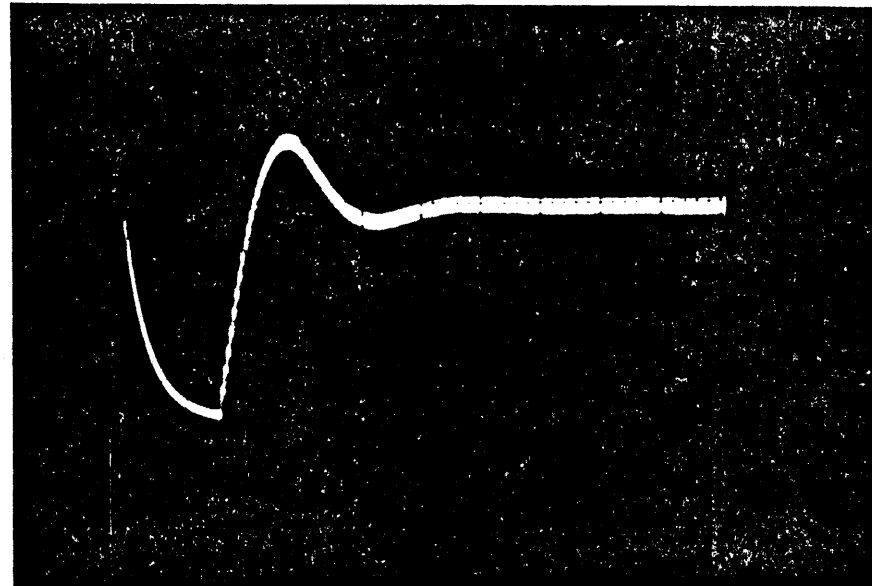
Figure 6-25

.2.6.2 Closed Loop Response Waveform

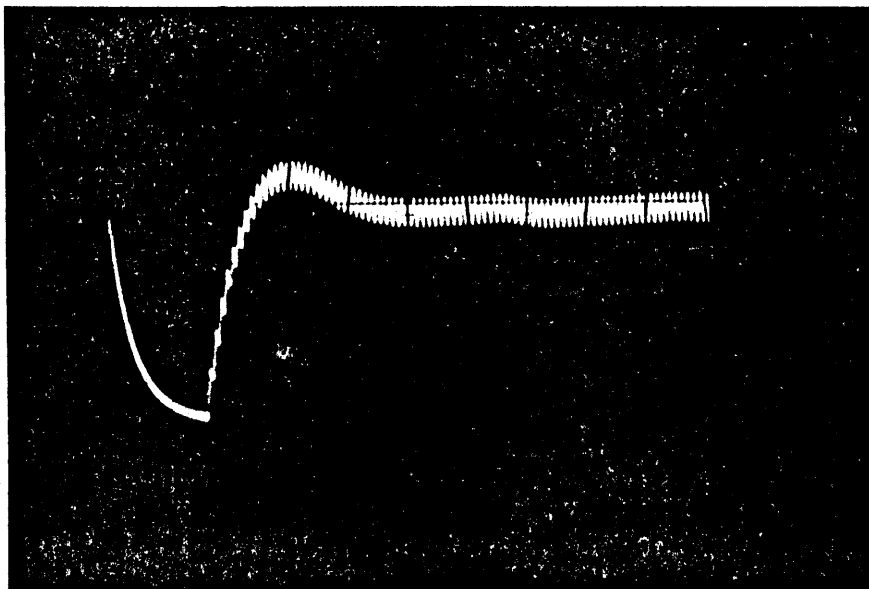
Waveforms observed as the closed loop control system restores the desired voltage after a disturbance is shown in Figures 6-24 and 6-25. The large overshoot as the desired voltage is approached, and the continuing oscillation above and below the desired voltage are inherent and undesirable results of Integral Control. When the low output voltage is measured the computer calculates a large error signal and adjusts the pulse width accordingly. At the next measurement a smaller but still substantial error is observed and a further adjustment is made to the pulse width. This continues until the actual voltage has reached and passed the desired voltage. At this point the pulse width is set too wide for the desired voltage. A negative error is detected and the pulse width is reduced. The capacitor is still charging however, so the voltage continues to rise. Moreover, the error detected is small and the adjustment made is not yet sufficient to bring the voltage back down to the desired value. The result is that the voltage rises substantially above the desired value before it starts down. This is called "overshoot". A number of measurements and adjustments are made before the voltage again reaches the desired value. By now the closed loop control system has reduced the pulse width too far, and undershoot occurs. The process continues indefinitely, reaching a steady state of oscillation above and below the desired value. The amount of overshoot and undershoot, the amplitude of the oscillation, and the time before a steady state is reached will be seen to depend on the total period of the pulse width modulation.



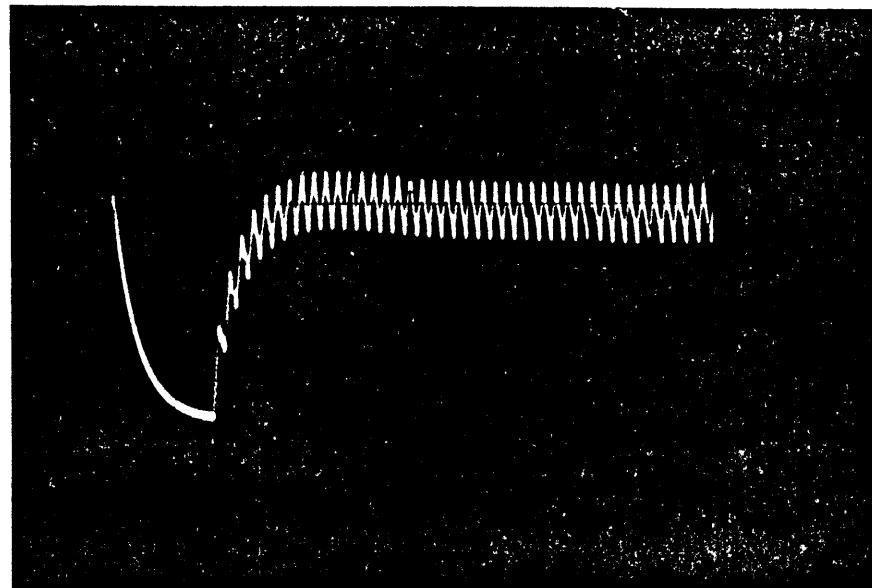
TOTAL PERIOD 400 (hex) 0.5 MS



TOTAL PERIOD 800 (hex) 1.0 MS



TOTAL PERIOD 1000 (hex) 2.0 MS



TOTAL PERIOD 2000 (hex) 4.0 MS

SCALES: 0.5 VOLTS/DIV, 20 MS/DIV

EFFECT OF TOTAL PERIOD

FIG 6-26

6.2.6.3 Effect of Total Period

Figure 6-26 shows oscilloscope traces for four different values of total period. It is apparent that by using a long period we can greatly reduce the overshoot and oscillation, but at the cost of introducing a large voltage fluctuation in each charge/discharge cycle. With a total period of 2,000 (hex) the overshoot is small but the voltage varies by about 0.4 volt during each cycle. This is clearly not a desirable scheme. When we develop a more sophisticated closed loop control system in Section 6.3 we will overcome this problem.

6.2.6.4 Gain of Integral Control

We have exercised integral control by adding the error signal to the steady state pulse width. Let us examine the meaning of this procedure in terms of the integral control equation:

$$(a) \quad F = G \int E$$

Since the Integral represents the sum of all past error signals plus the new measurement, we can say:

$$(b) \quad F = GE + F'$$

Where F' is the previous value of the control force.

The relation t_c/t_p represents the control force, since we can scale these two values without changing the result, but changing either above does affect the output. Therefore:

$$(c) \quad F = \frac{t_c}{t_p}$$

$$(d) \quad F' = \frac{t_{c'}}{t_p} \quad \text{for constant total period}$$

$$(e) \quad \frac{t_c}{t_p} = GE + \frac{t_{c'}}{t_p} \quad \text{from (b) and (d)}$$

The procedure we have used added the error signal E to the old pulse width $t_{c'}$ to obtain a new t_c .

$$(f) \quad t_c = E + t_{c'}$$

Dividing by t_p :

$$(g) \quad \frac{t_c}{t_p} = \frac{E}{t_p} + \frac{t_{c'}}{t_p}$$

From inspection of equations (e) and (g) it is apparent that $G = 1/t_p$. When we increased the total period to reduce the overshoot we were reducing the gain of the control system. It is much more effective to do this by arithmetic in the control calculation, and this will be done in Section 6.3.

6.2.6.5 Pure Proportional Control

At the beginning of Section 6.2 we presented the control equation for proportional control with a steady state force:

$$(a) \quad F = GE+S$$

The program of Section 6.2.7 will introduce a proportional term separate from the integral term we have been using. To see the effect of proportional control above, change the RM instruction in INTEG to RET. In the program of Figure 6-23e, this change is:

```
837C   C9       RET      (was RM)
```

The pulse width will be calculated as before but the integral will never be stored. This gives proportional control with a steady state term set by the open loop system. Do this experiment:

```
BRK           Set open loop control

40,NEXT       Set minimum output

C8,STEP       No effect in open loop

C8,NEXT       Set nominal pulse width

              (Output will go to initial value)

CLR           Set proportional control
```

Proportional control will increase the output voltage, but will not reach the desired value. This is because the nominal pulse width

calculated in response to NEXT remains as the steady state value. The pulse width is increased by the proportional control system, but only by the amount of the error measured at that instant with no cumulative correction. In the system we have here, pure proportional control is little more effective than open loop control. It is useful in systems where no steady state control force is needed, or in combination with integral control.

.3 Proportional Plus Integral Control

A control system that requires a steady state force to be adjusted for variable external conditions demands integral control. Random disturbances are better overcome by proportional control. Therefore a combination of both forms of control is very commonly used; it is called Proportional Plus Integral control. The control equation becomes:

$$F = G_p E + G_i \int E$$

We determined at the end of section 6.2 that $G_i = 1/t_p$ in our present system, where the error signal is added into the integral and the result sets the charging pulse width.

If we then add the error term again before loading the timer but do not change the integral in response to this addition, we will have a proportional plus integral system with equal gains.

We recognized at the end of section 6.2 a need to reduce the integral gain to avoid large overshoot. We also observed that (with a fixed steady state term) a large error signal was needed to affect the output significantly. The proportional control would have been more effective with a higher gain, since then the correction applied would have been greater for any given error signal. It is very common in

proportional plus integral control systems for the proportional gain to be much greater than the integral gain.

6.3.1 Applying Gain to Error Signal

In our new system we will provide for dividing the error signal by some value before adding it into the integral term, and multiplying it by some other value before adding it as the proportional term.

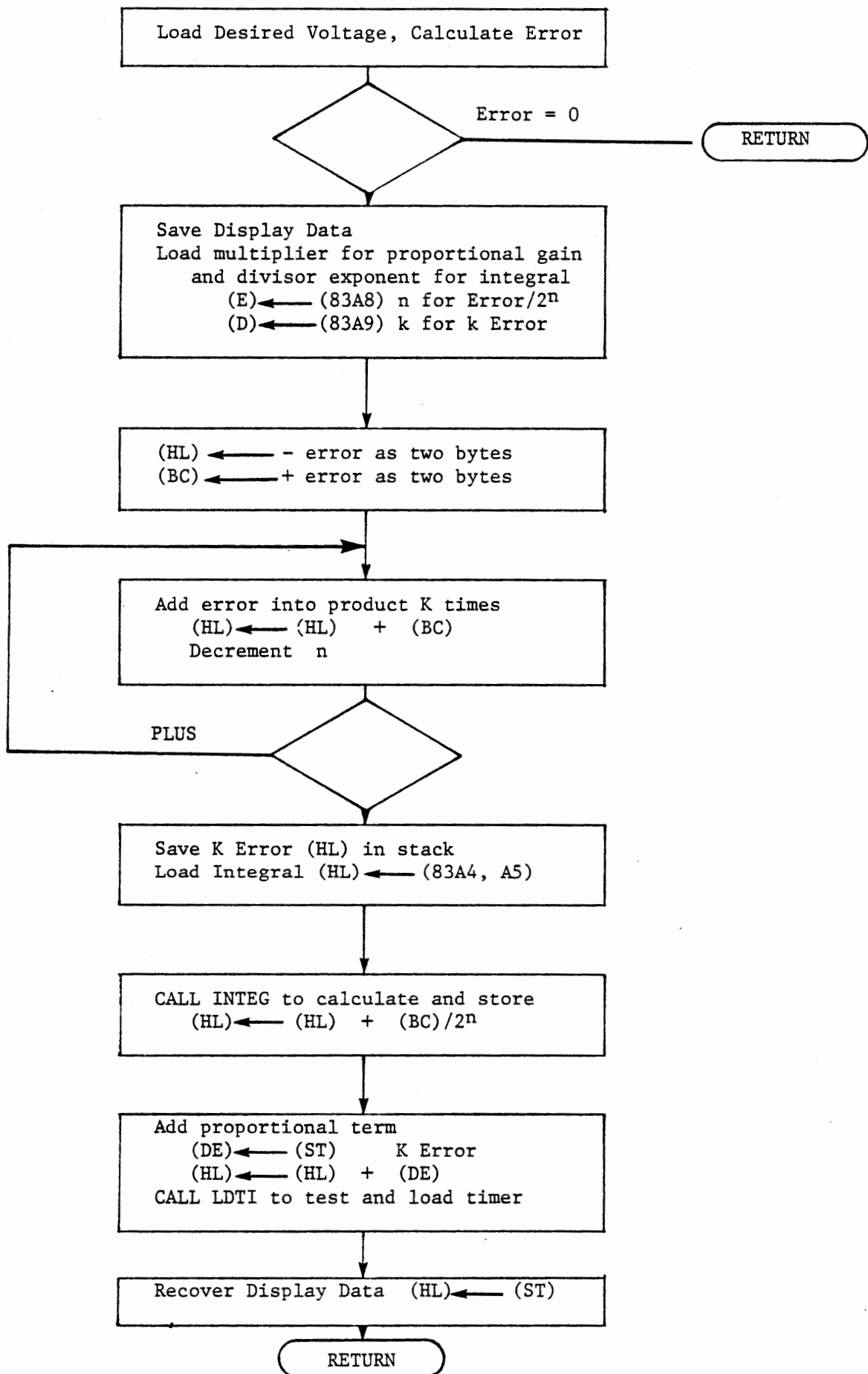
For ease of computation the integral gain divisor will be of the form 2^n , so that the division is merely a shift of n bits to the right. Typical values for n will be 0 to 4, giving division by 1, 2, 4, 8 or 16. The multiplication for the proportional term will be done by repeated addition, so the number used (again typically 0 to 4) will now actually be the multiplier. Our control equation will now be

$$F = \frac{K}{t_P} + \frac{1}{2^n t_P} \int E$$

For convenience in further discussion we will ignore the total period here and speak of K and $1/2^n$ as the proportional and integral gains. These are the data elements stored by the MEM key, at (83A8,A9). Both values are to be entered: K first, followed by n, followed by MEM. For instance, 203 MEM will set k=2 and n=3, giving a proportional gain of 2 and integral gain of 1/8. Note that the data entry procedure stores k at 83A9 and n at 83A8.

The calculation procedure is described briefly below, with detailed flow charts in figure 6-27 and the program in figure 6-28.

Calculate error signal
Multiply by K and save K Error
Divide error signal by 2^n
Add $\text{Error}/2^n$ to old integral
Test for positive value
Store new integral unless negative
Add K Error to new integral
Test for positive value greater than zero
Load timer if new pulse width >0



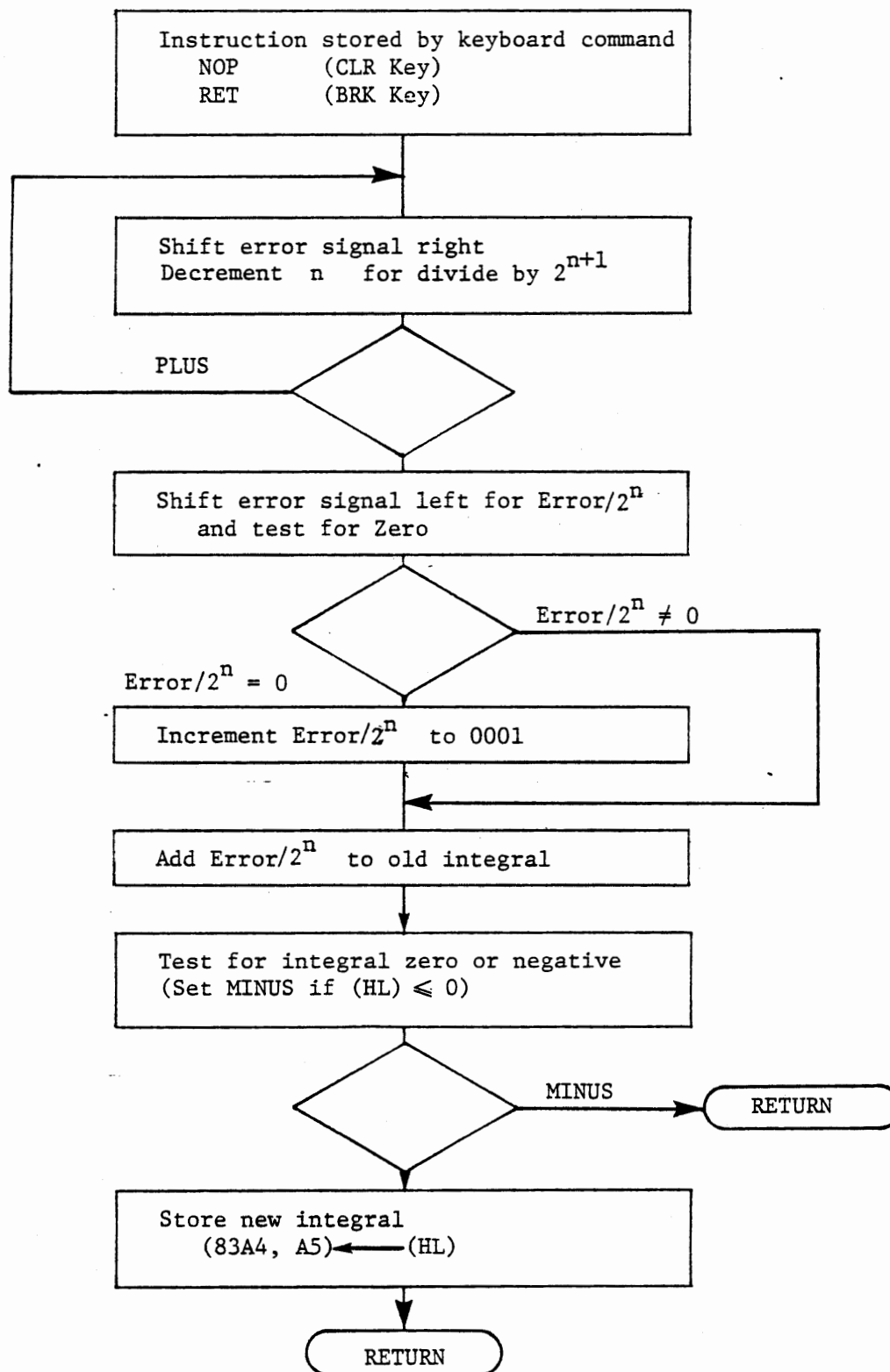
PWM SUBROUTINE CLOSL - VERSION 2

FIGURE 6-27a

3.2 Subroutine CLOSL Version 2

CLOSL will again generate data for the display and exit if no error exists. Now it must load the proportional gain multiplier K and the exponent n of the integral gain divisor 2^n . These are loaded to (D) and (E) respectively. The multiplication process shown in figure 27a is efficient for a multiplier whose value is small and possibly zero. The product is initially loaded with the two's complement of the multiplicand instead of being cleared. The multiplicand is added once if the multiplier is zero, giving a zero result. It is repetitively added while the multiplier is decremented, and the loop continues until the multiplier becomes negative (FF) and the decrement sets the MINUS flag (SIGN Bit).

A revised version of INTEG calculates the new integral, stores it and returns its value. The proportional term is added to this and the result is loaded to Timer 1. (provided it is positive and not zero) by a call to LDTI.



PWM - SUBROUTINE INTEG VERSION 2

FIGURE 6-27b

3.3 Subroutine INTEG Version 2

Again provision is made for modifying the program by BRK and CLR, storing NOP or RET at the start of INTEG. CLR will set integral control (by entering NOP). BRK will disable integral control, leaving pure proportional control. If the proportional gain multiplier is set to zero the process is then identical to open loop operation. To divide the error signal by 2^n we shift the content of register pair BC right as we decrement the divisor exponent n in register E. Since the high byte of the error signal is always either 00 or FF it is not necessary to change that value as we shift, but it must be used to shift in a zero or one to the high bit of the low byte.

```
SHIFT    MOV     A,B
          RAR
          MOV     A,C
          RAR
          MOV     C,A
          DCR     E
          JPLUS  SHIFT
```

As in the multiplication in CLOSL the loop continues until the count reaches FF so that it is executed once if the exponent is zero. This results in division by 2^{n+1} . It is easy to restore the correct value at the end of the loop by a shift left. In order to test for a zero result at the same time we use ADC A instead of RAL. This has the same effect as RAL except that it sets or clears all flags according to the result. (See Course 525, section 7.1.4). The result

of the left shift is now placed in Register C.

When n is greater than zero and the error signal is a small positive value, the result of the division may be zero even though there was an error. It is desirable to increase the integral in this case, especially since an equally small negative error will reduce the integral. (FFFF shifted right remains FFFF). Therefore, if ADC A set the zero flag we will increment the result in (BC). Now a positive error will always increase the integral and a negative error will always decrease it, no matter how small the gain. A zero error does not affect the pulse width because of the Return if Zero in CLOSL immediately after the error calculation.

We are not yet ready to load the timer, since the proportional term is still to be added. Therefore we must test for a negative integral in subroutine INTEG before storing the result. Zero is not forbidden here, and negative integrals would be acceptable but, as we will demonstrate, there is a possibility of losing control without this protection.

For ease of reference Figure 6-28 includes the main loop, subroutines KYTIM (with LDT1), CLOSL, and INTEG. The only changes from the preceding program are at 834D and 8367 through 839F.

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	2C0	DB	IN	PORT0A	Read Keyboard
	1	00			(FF if no key)
	2	3C	INR	A	
	3	C4	CNZ	KYTIM	Is key pressed
	4	00			call for data entry
	5	83			and process
	6	CD	CALL	VOLTM	Measure voltage
	7	50			(A) ← A/D voltage
	8	82			(FF if not ready)
	9	F5	PUSH	PSW	Save voltage
	A	CD	CALL	CLOSL	Close the loop
	B	60			(L) ← desired voltage
	C	83			(H) ← error signal
	D	CD	CALL	DWORD	Display error
	E	D1			signal (at left)
	F	02			and desired voltage
8	2D0	F1	POP	PSW	(A) ← measured voltage
8	2D1	21	LXI	H, 83A0	Memory address
	2	A0			for FILTER
	3	83			
	4	CD	CALL	FILTR	
	5	70			(L) ← raw voltage
	6	82			(H) ← filtered
	7	11	LXI	D, 83FF	Display raw
	8	FF			voltage (at right)
	9	83			and filtered voltage
	A	CD	CALL	DWDZ	
	B	D4			
	C	02			
	D	C3	JMP	82C0	Loop
	E	C0			
	F	82			
8	0				
	1				
	2		IDENTICAL TO 6-23a		
	3				
	4				
	5				
	6				
	7				
	8				FIGURE 6-28a

A D D R		CODE	PWM - SUBR KYTIM					
CODING SHEET	8	300	00	NOP			for DI during debug	
		1	CD	CALL	ENTWD		(A) ← command	
		2	46				(HL) ← total period	
		3	03				or voltage	
		4	EB	XCHG			(DE) ← data	
		5	21	LXI	H, 8308		Address dispatch	
		6	08				table - 10	
		7	83					
		8	85	ADD	L		Add command	
		9	6F	MOV	L, A		} (ST) ← dispatch address	
	A	6E	MOV	L, M				
	B	E5	PUSH	H				
MICROCOMPUTER TRAINING SYSTEM		C	F3	DI			} Set Port 1A6 high	
		D	DB	IN	PORT 1A			for scope trigger
		E	04					without changing
		F	F6	ORI	40		port 1A7.	
		8	310	40				
		1	FB	EI				
		2	D3	OUT	PORT 1A			
		3	04					
		4	AF	XRA	A		Clear A and CY.	
		5	00	NOP				
	6	00	NOP			During debug RST4		
	7	C9	RET			or BRKPT here		
INTEGRATED COMPUTER SYSTEMS	8318	45	MEM					
		9	50	REG				
		A	17	ADDR			Trigger scope	
		B	21	STEP			Store desired voltage	
		C	3E	RUN			Set total period	
		D	20	NEXT			Store voltage, set width	
		E	4A	BRK				
		F	4C	CLR				
		8	0					
		1						
	2		IDENTICAL	TO	6-23b			
	3							
	4							
	5							
	6							
	7							
	8					FIGURE 6-28b		

A	D	D	R	CODE	OPERATION	OPERANDS	DESCRIPTION
8	3	2	0	37	STC		NEXT
			1	21	LXI	H, 8000	STEP (A and CY clear)
			2	00	**		Load data logging
			3	80	**		address with its
			4	75	MOV	M, L	own location
			5	EB	XCHG		(HL) ← input data
			6	22	SHLD	83A6	Store desired voltage
			7	A6			
			8	83			
			9	DO	RNC		Exit if STEP
			A	11	LXI	D, FFEE	To subtract 0012
			B	EE			from desired voltage
			C	FF			
			D	19	DAD	D	(HL) ← N - V ₂
			E	29	DAD	H	(HL) ← pulse width
832	F			22	SHLD	83A4	STRW
833	0			A4			Store pulse width
			1	83			
833	2			2B	DCX	H	LDT1
			3	7C	MOV	A, H	Test pulse width
			4	B7	ORA	A	for zero or negative
			5	23	INX	H	
			6	F8	RMINUS		Exit if ≤ 0
			7	7D	MOV	A, L	Load Timer 1
			8	D3	OUT	TIM1	with charging
			9	15			pulse width
			A	7C	MOV	A, H	
			B	D3	OUT	TIM1	
			C	15			
			D	C9	RET		
833	E			7B	MOV	A, E	RUN
			F	D3	OUT	TIM0	Load Timer 0
834	0			14			with total period
			1	7A	MOV	A, D	
			2	D3	OUT	TIM0	
			3	14			
			4	C9	RET		
			5		IDENTICAL		TO G-23C
			6		** USE	82E0	FOR SHORT LOG
			7				
			8				FIGURE G-28C

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE	SUBR	C	LOSL	VERSION	2
8	3	6	0	2A	LHLD	83A6			
			1	A6					(L) ← desired voltage
			2	83					
			3	95	SUB	L			(A) ← - error signal
			4	67	MOV	H, A			(H) ← - error signal
			5	C8	RZ				Return if error = 0
			6	E5	PUSH	H			Save for display
			7	2A	LHLD	83A8			} (E) ← M for Error/2 ^M (D) ← K for K Error
			8	A8					
			9	83					
			A	EB	XCHG				
			B	6F	MOV	L, A			} (HL) ← - error (BC) ← + error
			C	2F	CMA				
			D	4F	MOV	C, A			
			E	9F	SBB	A			
			F	67	MOV	H, A			
8	3	7	0	2F	CMA				
			1	47	MOV	B, A			
			2	03	INX	B			
8	3	7	3	09	DAD	B			} (HL) ← K Error if K = 0 (HL) ← 0000
			4	15	DCR	D			
			5	F2	JPLUS	8373			
			6	73					
			7	83					
			8	E5	PUSH	H			(ST) ← K Error
			9	2A	LHLD	83A4			(HL) ← old integral
			A	A4					
			B	83					
			C	CD	CALL	INTEG			
			D	86					
			E	83					
			F	D1	POP	D			
8	3	8	0	19	DAD	D			
			1	CD	CALL	LDT1			
			2	32					
			3	83					
			4	E1	POP	H			
			5	C9	RET				
			6		ENTER	(A) = MEASURED VOLTAGE			
			7		RETURN	(HL) FOR DISPLAY			
			8		ALL	REGISTERS USED			

FIGURE 6-28.e

THIS PAGE INTENTIONALLY LEFT BLANK

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	0										
	1										
	2										
	3										
	4										
	5										
838	6	00		NOP / RET						RET stored by BRK	
838	7	78		MOV A, B						for no integral	
	8	1F		RAR							
	9	79		MOV A, C							
	A	1F		RAR							
	B	4F		MOV C, A						(BC) ← Error / 2 ^{m+1}	
	C	1D		DCR E							
	D	F2		JPLUS 8387							
	E	87									
	F	83									
839	0	8F		ADC A							
	1	4F		MOV C, A						(BC) ← Error / 2 ^m	
	2	C2		JNZ 8396						Skip unless result = 0	
	3	96									
	4	83								If Error / 2 ^m = 0	
	5	03		INX B						make it 0001	
839	6	09		DAD B						(HL) ← new integral	
	7	7C		MOV A, H						Test for negative	
	8	B7		ORA A						integral	
	9	F8		RMINUS						Exit if negative	
	A	22		SHLD 83A4						Store integral	
	B	A4									
	C	83									
	D	C9		RET							
	E										
	F										
8	0										
	1			CALLED BY CLOSL WITH							
	2			(BC) = ERROR SIGNAL							
	3			(HL) = OLD INTEGRAL							
	4			(E) = M for divide by 2 ^m							
	5			RETURNS NEW INTEGRAL							
	6			(HL) = ERROR / 2 ^m							
	7			STORED UNLESS NEGATIVE							
	8									FIGURE 6-25	

6.3.4 Experiments with PI Control

The effect of proportional plus integral control is to achieve rapid response to a disturbance or to a change in the desired output value without the objectionable overshoot and oscillation associated with pure integral control. If an oscilloscope is available the observations are easily made; lacking an oscilloscope the data can be logged and plotted. Waveform photographs are presented here for several of the experiments.

6.3.4.1 Open Loop Control

To demonstrate that open loop control is still available in the system with version 2 of CLOSL and INTEG, enter the following data and commands:

400,RUN	Set 0.5 ms total period
MEM	Set zero proportional gain
BRK	Disable integral control
40,NEXT	Set minimum output
C8,NEXT	Request 2.0 volts

The voltage will rise to its initial value of about 1.5 volts.

Adjust the SENSE pot and observe that the voltage changes.

Restore the SENSE pot to the full left position.

3.4.2 Pure Proportional Control

Set pure proportional control by:

```

400,RUN      Set 0.5 ms total period
MEM          Set zero proportional gain
BRK         Disable integral control
64,NEXT      Request 1.0 volts
100,MEM      Set proportional gain = 1

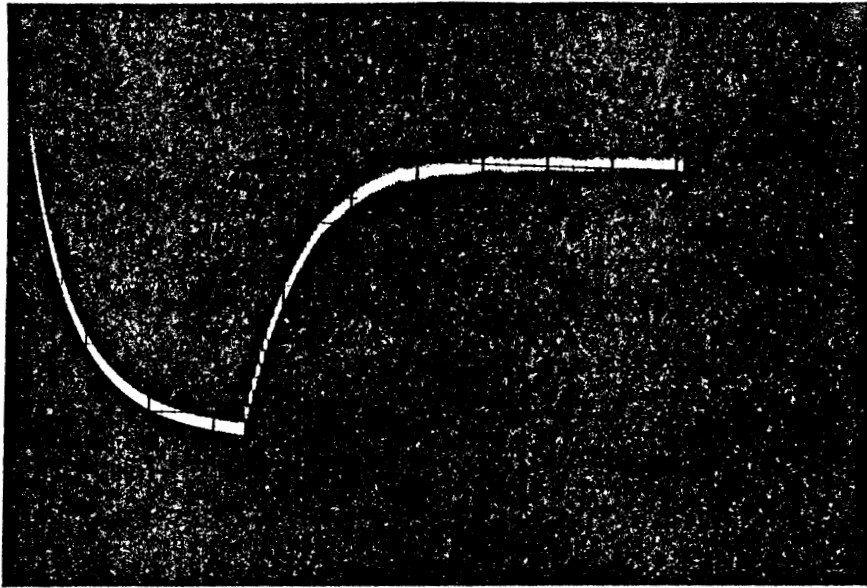
```

The output voltage will rise somewhat as proportional control increases the charging time above the fixed value calculated by NEXT.

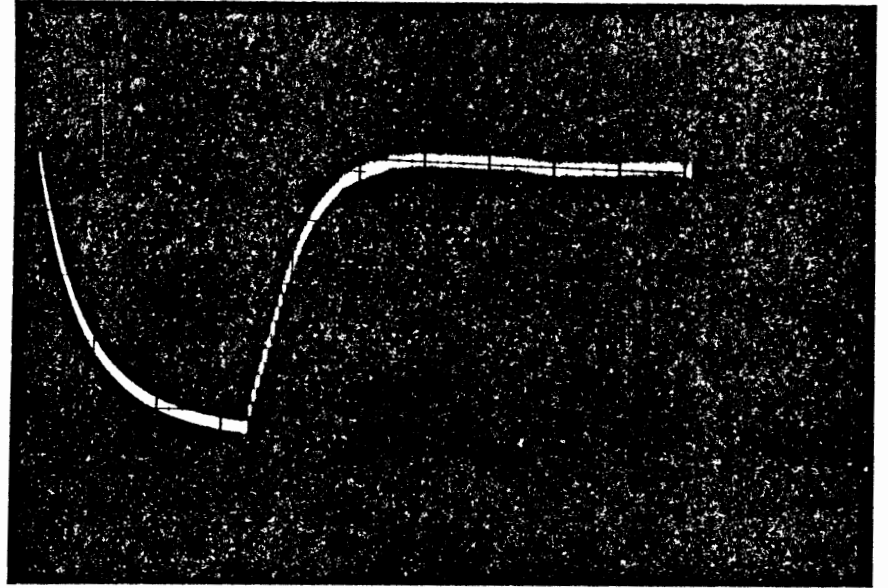
Observe the voltage generated with different proportional gains.

COMMAND	GAIN	OUTPUT VOLTAGE
MEM	0	_____
100,MEM	1	_____
200,MEM	2	_____
300,MEM	3	_____
400,MEM	4	_____
600,MEM	6	_____
800,MEM	8	_____
1000,MEM	16	_____

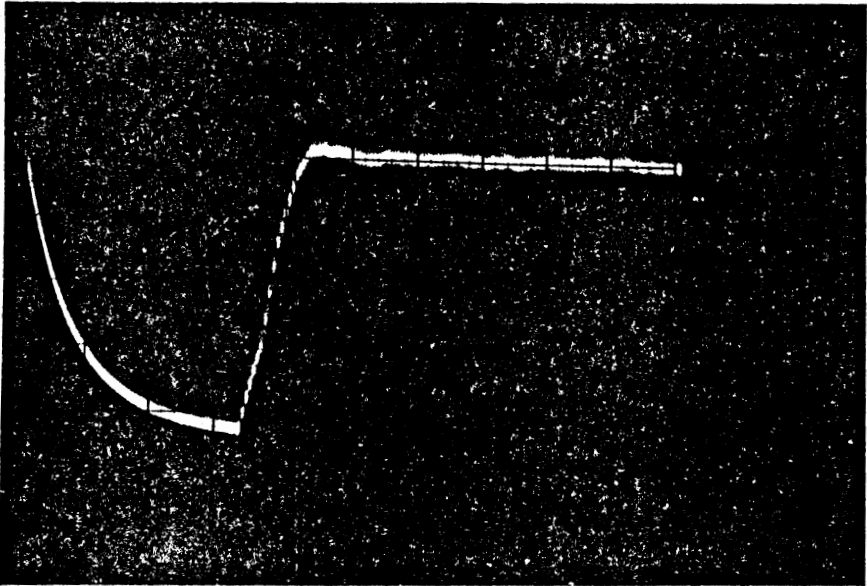
At a sufficiently high gain the multiplied error signal will lead to an unacceptable pulse width which will be rejected by LDT1.



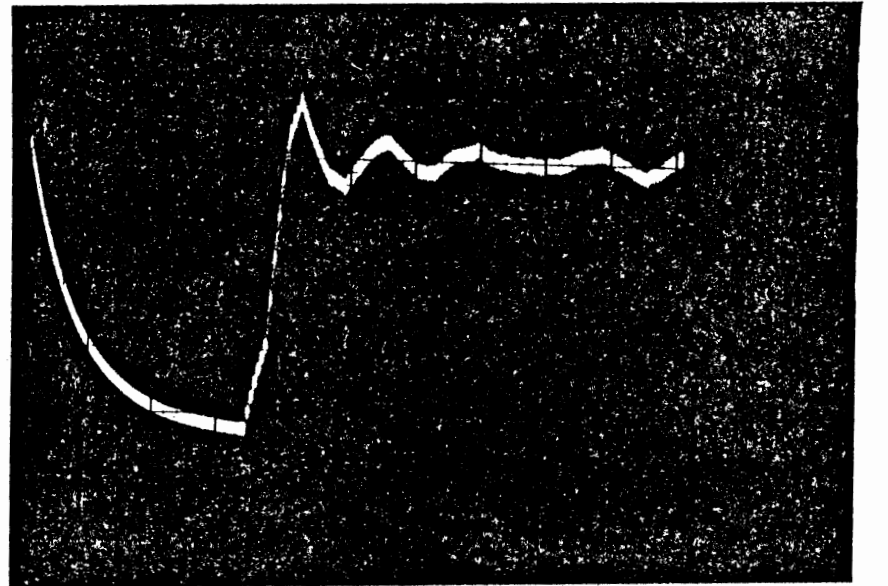
Proportional Gain = 0 (Open Loop)



Proportional Gain = 1



Proportional Gain = 4



Proportional Gain = 8

RESPONSE WITH PROPORTIONAL CONTROL

FIGURE 6-29

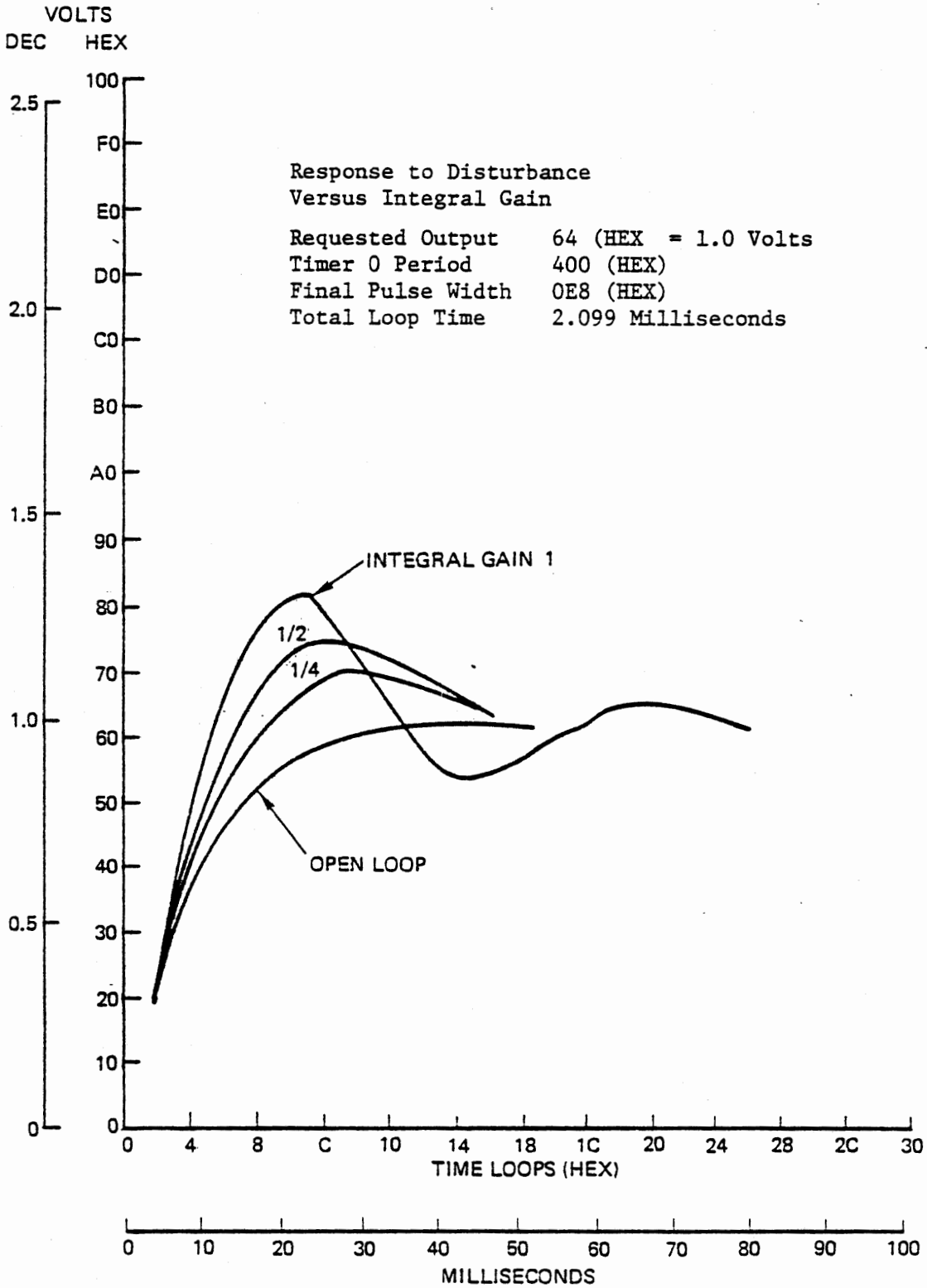
If an oscilloscope is available, do the following experiment. Set the oscilloscope to 0.2 volts/division, 10 milliseconds per division, to observe the rise time.

64,NEXT	Request 1.0 volt
MEM	Set integral gain = 1 and proportional gain = 0
CLR	Set closed loop to adjust the pulse width
BRK	Set open loop
REG	Force output low.

Observe the response to the disturbance.

100,MEM	Set proportional gain = 1
REG	Force output low

Repeat with successively higher proportional gain values to observe the response. Figure 6-29 shows several response waveforms. The speed of response increases as the proportional gain is increased.



Response Versus Integral Gain

Figure 6-30

.3.4.3 Pure Integral Control

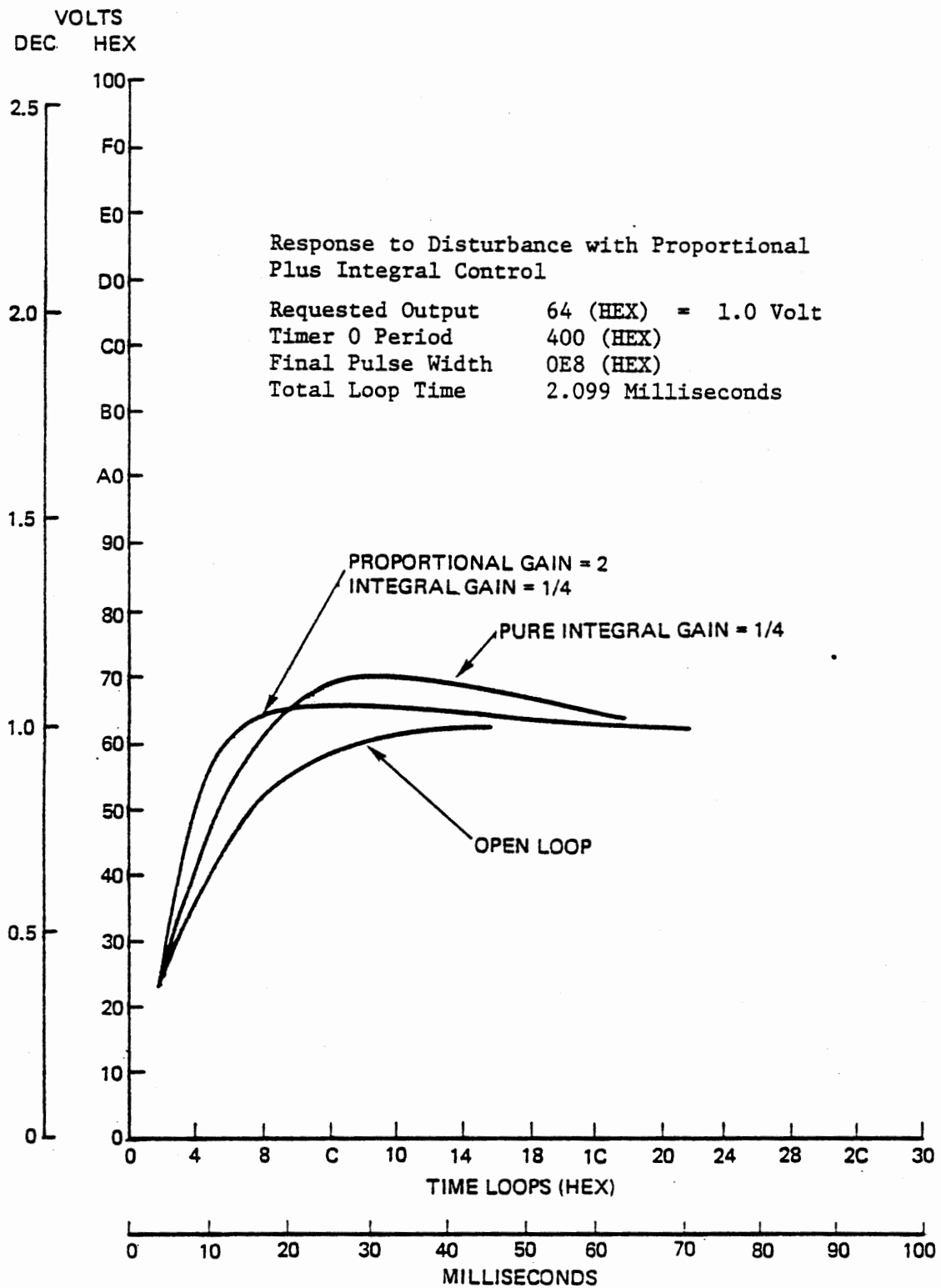
The effect of integral control on the response waveform has been observed previously. The following experiments shows the effect of reduced integral gain. If you have an oscilloscope make all of these observations. Otherwise, plot the response from the data log for gain = 1 and gain = 1/4. Enter these commands:

```
MEM          Set proportional gain = 0
             and n = 0 for integral gain = 1
CLR          Enable integral control
64,NEXT     Request 1.0 volt
REG         Force output low
```

Observe or plot the response.

```
1,MEM       Set integral gain = 1/2
REG         Observe response
2,MEM       Set integral gain = 1/4
REG         Observe or plot response
3,MEM       Set integral gain = 1/8
REG         Observe response
BRK         Set open loop
REG         Observe response
```

Continue to decrease the gain and observe the response. Figure 6-30 shows responses for selected gains, and for open loop operation.



Proportional Plus Integral Response

Figure 6-31

3.4.4 Proportional Plus Integral Control

Directly after the preceding experiment observe or plot the response with proportional plus integral control.

```

202, MEM      Set proportional gain = 2
              and integral gain = 1/4
REG          Observe or plot response

```

This demonstrates an effective control system. Less overshoot occurs than with pure integral control using the same gain because the higher gain proportional control promptly corrects the overshoot. A fast response to the disturbance is observed. Figure 6-31 compares the response obtained here with some of our earlier results.

3.4.5 Response to Voltage Request

The preceding observations of response time have all maintained a constant desired voltage, forcing the output low under open loop control. Observe the response to a change in desired voltage:

```

BRK, MEM      Set open loop operation
C8, NEXT      Request 2.0 volts
xxx, RUN      Set total period to obtain requested voltage
40, NEXT      Request minimum output and observe response
C8, NEXT      Request 2.0 volts and observe output

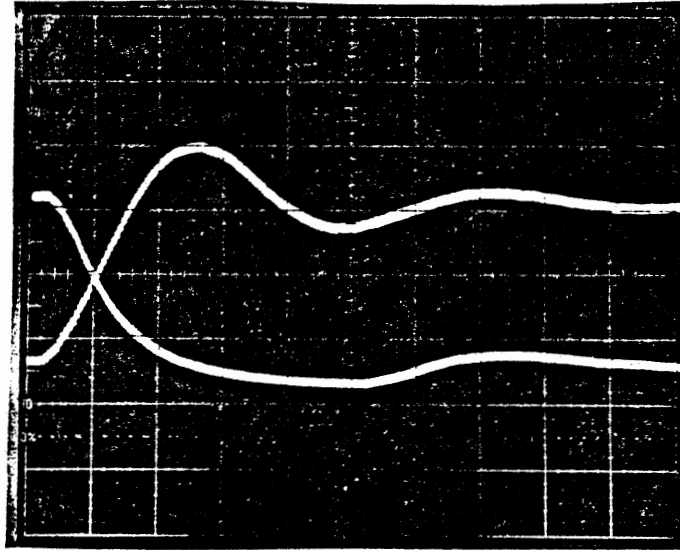
```

INCREASING PROPORTIONAL GAIN

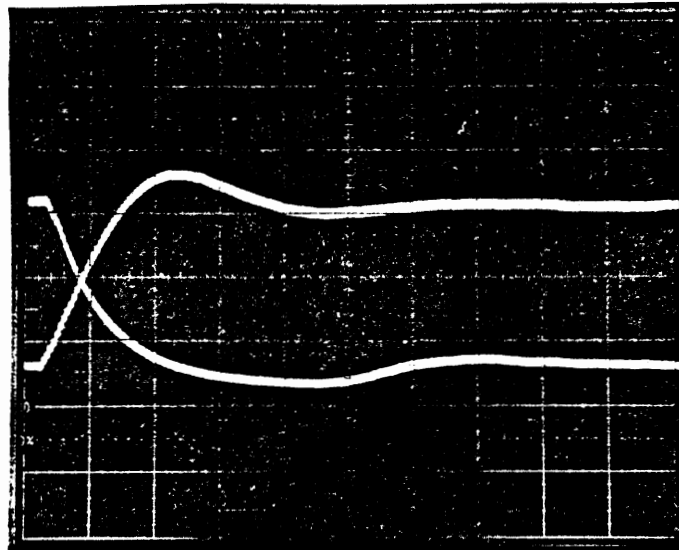
Proportional Gain = 0
Integral Gain = 1

Voltage was C8
Voltage requested 40

Voltage was 40
Voltage requested C8



Proportional Gain = 1
Integral Gain = 1

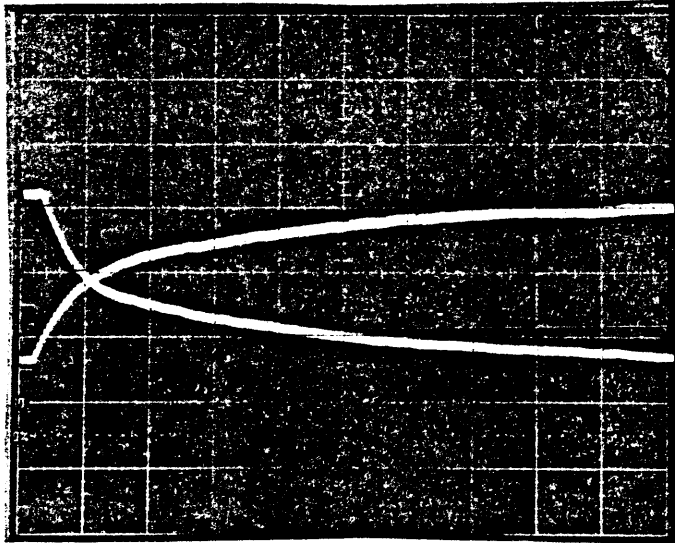


RESPONSE TO VOLTAGE REQUEST

FIGURE 6-32a

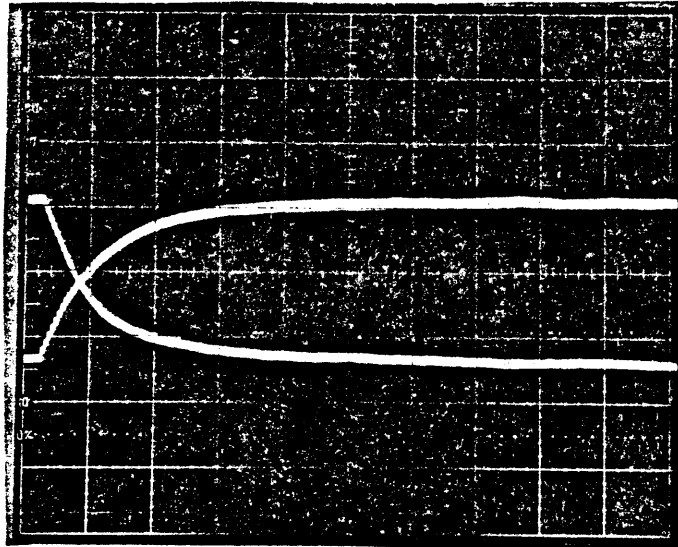
Proportional Gain = 2

Integral Gain = $\frac{1}{4}$



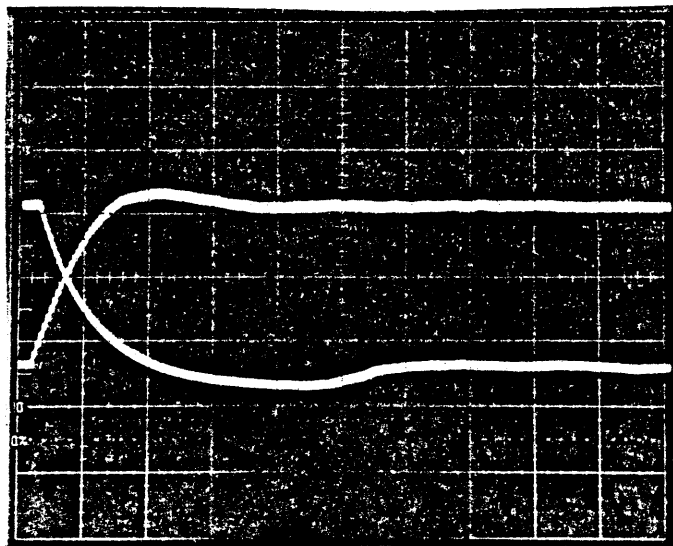
Proportional Gain = 2

Integral Gain = $\frac{1}{2}$



Proportional Gain = 2

Integral Gain = 1



RESPONSE TO VOLTAGE REQUEST (CONT'D)

FIGURE 6-32b

Now set proportional plus integral control by:

202, MEM	Set proportional gain = 2 and integral gain = 1/4
CLR	Enable integral control
40, STEP	Request minimum output and observe response
C8, STEP	Request 2.0 volts and observe response

Results of this test are shown in Figure 6-32. Note that with open loop control the rise and fall are similar, but with closed loop control they are distinctly different.

6.3.5 Full Scale Control and Overflow

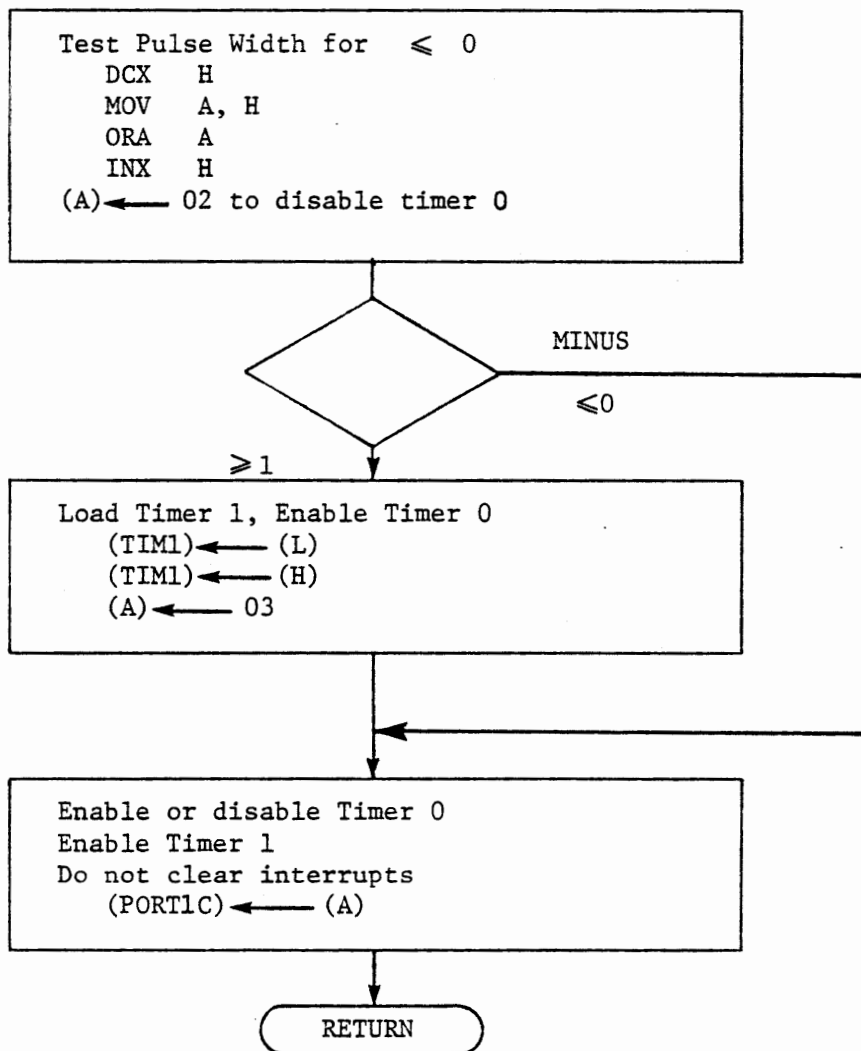
In subroutine LDT1 we have protected against loading Timer 1 with an unintended long time period when the calculated pulse width is zero or negative. The reaction when this occurs is to leave the preceding value of the pulse width unchanged. This has been acceptable but is not the best arrangement. It would be better to recognize the condition and disable charging altogether if the measured voltage is so much greater than the desired value that proportional control cannot operate correctly. This would allow the voltage to drop more quickly, and also remove the limitation imposed by the time taken in the interrupt service routines.

(Recall that Timer 0 interrupt starts charging the capacitor by setting Port 1A7 high, and no matter how small a value is loaded to

Timer 1 the output will remain high until the Timer 0 interrupt is finished and the Timer 1 interrupt sets the output low. This takes 66 clock periods after OUT PORT1A in Timer 0 service, plus 50 clock periods for the Timer 1 interrupt and processing through its OUT PORT1A. This gives a minimum pulse of 57 microseconds and an output of 0.8 volts if the total period is 300 hex).

When we abandon proportional control and set a minimum (or maximum) control force we are employing "full scale control". We can do this here by changing the LDT1 module of KYTIM. To make space for the larger LDT1 we will abandon the special function of the NEXT command. Replace the RNC command at 8329 with RET, so that NEXT and STEP will be processed alike, merely starting a log and storing the desired voltage. Change the call to LDT1 to call address 832A.

LDT1 Enter with (HL) = Pulse Width



LDT1 WITH FULL SCALE CONTROL

FIGURE 6-33

.3.5.1 LDT1 with Full Scale Control

Enter the revised program for LDT1 in memory locations 832A through 833D, according to Figure 6-33. With this program the calculated pulse width is tested as before for a negative or zero value. If the width is greater than zero it is loaded into Timer 1 and the interrupt from Timer 0 is enabled. If the width is zero or negative the timer is not loaded, and Timer 1 is disabled. Note that the enable or disable is not to clear the interrupt, so it is done by writing to Port 2C, not to CNT2. If the bit set function were used to enable the interrupt it would occasionally occur just as the timer generated an interrupt, thereby inhibiting a charging pulse. With the method of writing to Port 1C, normal operation will be totally unaffected, since the interrupt is always enabled. After a time of full scale control with Timer 0 disabled, the calculated pulse width will again become acceptable and Port 2C0 will be set high. An interrupt will occur immediately. A charging pulse will start, but its duration will be random, depending on when the next Timer 1 interrupt occurs.

Since we have destroyed the function of NEXT that stored a value for the integral and loaded Timer 1, we must alter the initialization procedure. The new process sets the total period as before. It stores C8 at 83A6 for the desired voltage and stores 0400 as the integral at (83A4,A5). Now closed loop control will load Timer 1 with this value, quickly driving the output off scale. Then the integral control system will reduce the pulse width to an appropriate value. Note that this procedure is not effective without integral control.

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	2B	0	11	LXI	D, 0400	Set total period to 0.5 milliseconds
		1	00			
		2	04			
		3	CD	CALL	RUN	Load Timer 0
		4	3E			
		5	83			
		6	2E	MVI	L, A6	Address 83A6
		7	A6			(H) = 83 already
		8	36	MVI	M, C8	Store 2.0 volts
		9	C8			as desired voltage
	A		EB	XCHG		(HL) ← 0400
	B		22	SHLD	83A4	Store 0400 as
	C		A4			integral.
	D		83			
	E		EF	RST5		Enable timer
	F		F7	RST6		interrupts
8		0				
		1				
		2		FINISH	INITIALIZATION	
		3		WITHOUT	CALL TO NEXT	
		4				
		5		SETS	TOTAL PERIOD	
		6		AND	DESIRED VOLTAGE	
		7				
		8		STORES	TOTAL PERIOD (0400)	
		9		AS	VALUE OF INTEGRAL	
	A					
	B					
	C					
	D					
	E					
	F					
8		0				
		1				
		2				
		3				
		4				
		5				
		6				
		7				
		8				FIGURE 6-34a

A D D R		CODE	PWM	VOLTAGE CONTROL - MAIN LOOP			
CODING SHEET	8 2C 0	DB	IN	PORT	OA	Read Keyboard (FF if no key)	
		00					
		3C	INR	A			
		C4	CNZ	KY	TI	M	If key pressed call for data entry and process
		00					
		83					
		CD	CALL	VOLT	M	Measure voltage (A) ← A/D voltage (FF if not ready)	
		50					
		82					
		F5	PUSH	PSW			Save voltage
MICROCOMPUTER TRAINING SYSTEM	A	CD	CALL	CLOS	L	Close the loop (L) ← desired voltage (H) ← error signal	
	B	60					
	C	83					
	D	CD	CALL	DWORD		Display error signal (at left) and desired voltage	
	E	D1					
	F	02					
	8 2D 0	F1	POP	PSW		(A) ← measured voltage	
	8 2D 1	21	LXI	H, 83A0		Memory address for FILTR	
		A0					
		83					
INTEGRATED COMPUTER SYSTEMS		CD	CALL	FILTR		(L) ← raw voltage (H) ← filtered	
		70					
		82					
		11	LXI	D, 83FF		Display raw voltage (at right) and filtered voltage	
		FF					
		83					
	A	CD	CALL	DWD2			
	B	D4					
	C	02					
	D	C3	JMP	82C0		Loop	
E	C0						
F	82						
8	0						
	1						
	2		IDENTICAL		TO 6-23a		
	3						
	4						
	5						
	6						
	7						
	8				FIGURE 6-34 b		

A D D R		CODE	PWM - SUBR KYTIM				
CODING SHEET	8 30 0	00	NOP				for DI during labw.
		1	CD	CALL	ENTWD		(A) ← command
		2	46				(HL) ← total period
		3	03				or voltage
		4	EB	XCHG			(DE) ← data
		5	21	LXI	H, 8308		Address dispatch
		6	08				table - 10
		7	83				
		8	85	ADD	L		Add command
		9	6F	MOV	L, A		} (ST) ← dispatch address
	A	6E	MOV	L, M			
	B	E5	PUSH	H			
MICROCOMPUTER TRAINING SYSTEM		C	F3	DI			} Set Port 1A6 high for scope trigger without changing port 1A7.
		D	DB	IN	PORT 1A		
		E	04				
		F	F6	ORI	40		
		8 31 0	40				
		1	FB	EI			
		2	D3	OUT	PORT 1A		
		3	04				
INTEGRATED COMPUTER SYSTEMS		4	AF	XRA	A		Clear A and CY.
		5	00	NOP.			
		6	00	NOP			During debug RST4
		7	C9	RET			or BRKPT here
		8 31 8	45	MEM			
		9	50	REG			
		A	17	ADDR			Trigger scope
		B	21	STEP			Store desired voltage
		C	3E	RUN			Set total period
		D	20	NEXT			Store voltage, set width
	E	4A	BRK				
	F	4C	CLR				
	8	0					
	1						
	2		IDENTICAL		TO	6-23b	
	3						
	4						
	5						
	6						
	7						
	8					FIGURE 6-3.4 C	

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE	OPERATION	ADDRESS	DESCRIPTION
8				0			
				1			
				2			
				3			
				4			
834				5	EB	XCHG	MEM
				6	22	SHLD	83A8 Store input data for CLOS
				7	A8		
				8	83		
				9	C9	RET	
834	A			3E	MVI	A, C9	BRK - Open Loop
	B			C9			Enter RET in INTEG
834	C			32	STA	8386	CLR - Close Loop
	D			86			Enter NOP in INTEG
	E			83			
	F			C9	RET		
835	0			D3	OUT	CNT2	REG - Force Output Low
	1			0F			Disable Timer 0 Interrupt
	2			CD	CALL	RSTDY	Monitor Function
	3			44			Delays 30 milliseconds
	4			02			plus interrupt time
	5			21	LXI	H, 8000	Start new log
	6			00	**		
	7			80	**		
	8			75	MOV	M, L	
	9			76	HLT		Wait for timer 1
	A			3E	MVI	A, 01	Enable and clear
	B			01			timer 0 interrupt
	C			D3	OUT	CNT2	
	D			0F			
	E			C9	RET		
	F			00	NOP		
8				0			
				1			
				2			
				3			
				4			
				5	**	82E0	FOR SHORT LOG
				6			
				7			
				8			FIGURE 6-34e

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8360	2A	LHLD	83A6	
1	A6			(L) ← desired voltage
2	83			
3	95	SUB	L	(A) ← - error signal
4	67	MOV	H, A	(H) ← - error signal
5	C8	RZ		Return if error = 0
6	E5	PUSH	H	Save for display
7	2A	LHLD	83A8	
8	A8			} (E) ← m for Error/2 ^m (D) ← K for K Error
9	83			
A	EB	XCHG		
B	6F	MOV	L, A	
C	2F	CMA		
D	4F	MOV	C, A	(HL) ← - error
E	9F	SBB	A	(BC) ← + error
F	67	MOV	H, A	
8370	2F	CMA		
1	47	MOV	B, A	
2	03	INX	B	
8373	09	DAD	B	
4	15	DCR	D	(HL) ← K Error
5	F2	JPLUS	8373	if K = 0
6	73			(HL) ← 0000
7	83			
8	E5	PUSH	H	(ST) ← K Error
9	2A	LHLD	83A4	(HL) ← old integral
A	A4			
B	83			
C	CD	CALL	INTEG	
D	86			
E	83			
F	D1	POP	D	
8380	19	DAD	D	
1	CD	CALL	LDT1	New address
2	2A *			for LDT1 with
3	83			full scale control
4	E1	POP	H	
5	C9	RET		
6		ENTER	(A)	= MEASURED VOLTAGE
7		RETURN	(HL)	FOR DISPLAY
8		ALL	REGISTERS	USED

FIGURE 6-34 f

3.5.2 Full Scale Control Experiments

To test the ability of full scale control to generate a low output signal enter:

```

400,RUN      Set 0.5 ms total period
MEM          Set proportional gain = 0
              and integral gain = 1
CLR          Enable integral control
STEP (or NEXT) Request zero output

```

Timer 0 interrupt will be disabled, so Port 1A7 will be continuously low. (Observe this in the LED). The output voltage will be determined by the division of V_g across the voltmeter resistance and the 10K resistor R_2 .

$$v = \frac{V_g R_m}{R_m + R_2}$$

Note that the voltage can be changed by switching scales on the voltmeter (which changes the voltmeter's resistance). If you disconnect the voltmeter the voltage measured by the A/D converter will be almost exactly equal to V_g , the zero offset voltage of the open collector driver of the output port. The voltmeter will reduce that voltage slightly if a 3 volt scale is used and significantly on more sensitive scales. (If you are using an electronic voltmeter with a high impedance input the voltmeter will not affect the output

voltage at all).

Now, reset the computer and check the integral stored at (83A4,A5). It will be some value between zero and the output voltage that was achieved. After this value was stored, INTEG attempted to reduce it by the observed error. The result was negative so it was not stored. A lower integral gain would allow it to be reduced further. The negative (or zero) pulse width passed to LDT1 gave full scale control and a minimum output.

Run the program again and enter:

```
MEM          Set proportional gain = 0
              and integral gain = 1
CLR          Enable integral control
STEP        Request zero output
BRK         Disable integral control
```

The voltage will rise to the minimum determined by the interrupt service processing. Without closed loop control no attempt is made to reduce the pulse width below zero, so LDT1 will enable Timer 0 interrupt. Enter:

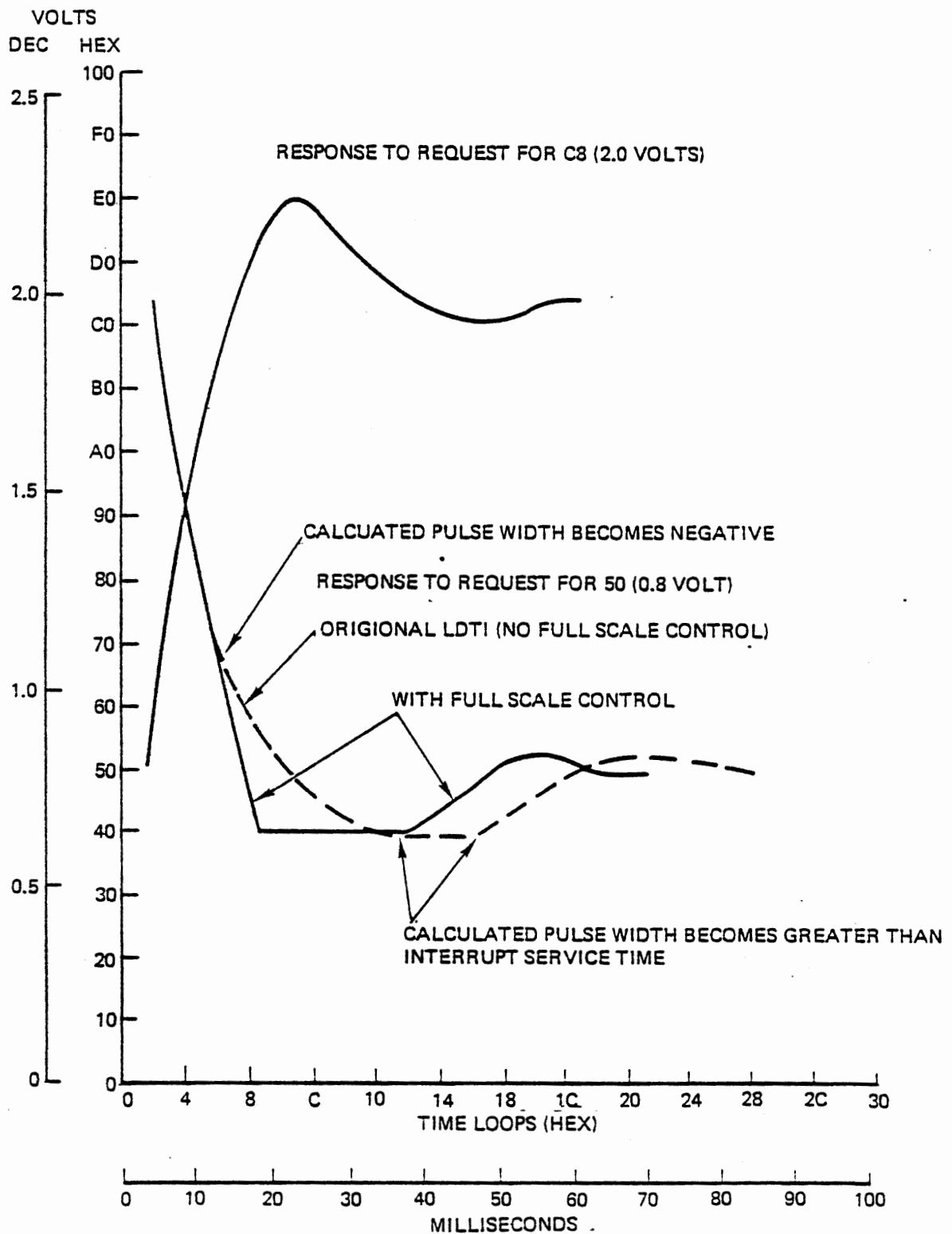
```
100,MEM      Set proportional gain = 1
```

Now the proportional control will generate negative pulse widths and full scale control will be invoked.

Observe the response to voltage requests with various proportional and integral gains. Figure 6-35 compares the response to requests for 50

(hex) and C8, with the original version of LDT1 and the new version with full scale control. For this plot both gains are set to unity.

100, MEM	Set proportional gain = 1 and integral gain = 1
CLR	Enable integral control
C8, STEP	Request 2.0 volts
50, STEP	Request 0.8 volt and observe response
C8, STEP	Request 2.0 volts and observe response



Full Scale Response to Voltage Request

Figure 6-35

6.3.5.3 Maximum Output Full Scale Control

Clearly full scale control could also be exercised in the other direction. If proportional control could not achieve a desired voltage or demanded too great a pulse width, Timer 1 could be disabled to leave Port 1A7 on continuously. In fact, the present integral control system has full scale control, since the charging pulse width can be increased up to 7FFF, about 16 milliseconds or 32 times the 0.5 millisecond "total period". If the Timer 1 period is greater than the Timer 0 period, each new output pulse from Timer 0 retriggers Timer 1, whose count is then reloaded automatically and never reaches zero. No Timer 1 interrupts are generated and Port 1A7 stays high.

Full scale control is very commonly employed in proportional control systems where the designer recognizes a need for signals outside of the proportional range. Especially in pure proportional systems (i.e., no integral control) it is often desirable to use very high proportional gain to obtain fast response to small error signals. This usually limits the proportional range to fairly small error signals and demands full scale control for large errors.

6.3.5.4 Integral Overflow

We have limited the integral of error signals stored by INTEG to the range 0000 to 7FFF. This limitation tends to distort the results, since the lower limit is often encountered, but it is almost impossible to reach the upper limit. We are also wasting half of the range of a two byte variable. If this were important (which it is

not) we could extend the range by allowing and correctly processing negative values of the integral.

At present INTEG refrains from storing the integral if it is negative. This applies the limits of 0000 to 7FFF. If this test is not applied the integral will temporarily go negative when the desired voltage is switched from a high value too a low value, giving a large negative error signal. This invokes full scale control. Provided that the measured voltage eventually becomes less than the desired voltage, giving a positive error signal, the integral will (after some time) become positive again and settle at a value that gives the correct pulse width.

To experiment with this, remove the following instructions from CLOSL and INTEG. (Replace each of them with NOP).

8366	E5	PUSH H	Save display data
8384	E1	POP H	Recover display data
8399	F8	RM	Exit if integral <0

Removing PUSH H and POP H will cause CLOSL to return the calculated pulse width for display instead of the error signal and desired voltage. Removing RM will cause INTEG to store the integral regardless of its value. Now run the program with the following data and commands. (We will use pure integral control so that the displayed pulse width will be the integral value).

MEM Set proportional gain = 0
 and integral gain = 1

CLR Enable integral control

FA,STEP Request 2.5 volts

50,STEP Request 0.8 volt

The two voltage requests may be repeated as often as necessary. After 50,STEP you will see an F appear momentarily in the left hand display digit, showing that the calculated pulse width has become negative. Thereafter the integral will become and remain positive.

20,STEP Request 0.32 volt

To reach this low a voltage full scale control must be invoked a large part of the time. The integral displayed at the left will show FFxx with an occasional appearance of 00xx.

STEP Request zero output

Since a zero volt output cannot be achieved even with full scale control the error signal will always be negative. The integral will be repeatedly reduced from FFxx down to 80xx, then to 7Fxx where it suddenly appears to be a large positive value. Until this point, full scale control has kept Timer 0 interrupt disabled and Port 1A7 low. Now Timer 1 is loaded with a long pulse width (about 16 milliseconds) and Timer 0 is enabled. Port 1A7 is set high and the output rises above 2.55 volts. The large negative error signal is calculated, rapidly reducing the integral until it goes negative again. You can observe the peculiar behavior on the voltmeter or at the LED for Port

1A7. A computer, like a person, may go crazy when presented with an impossible task and no escape mechanism. The program works well as long as it is able to achieve its objective, but it needs the protection of the RM instruction for the impossible request.

Restore the three instructions that were deleted for this experiment.

7.4 Proportional - Integral - Differential Control

Consider a system with substantial inertia, such as steering a ship or aircraft. When the aiming point is changed or a sudden disturbance such as a wind gust or wave causes a large error signal, the proportional term in the control equation generates a large (possibly full scale) control force. The inertia prevents an instantaneous response, so after a brief time this force is further increased by the integral term. Now as the ship starts to respond the proportional term $G_p E$ decreases but the integral term $G_i \int E$ is still increasing. The ship now swings toward the aiming point, and its inertia will carry it beyond the desired point. An error in the opposite direction appears, the proportional term in the control equation becomes negative, and eventually the ship settles on its new course, provided the control system is stable. There may be significant and undesirable oscillations before the new course is achieved if high gains are used in the control system, and it is possible too have unstable operation where the oscillations are maintained indefinitely.

Differential or rate control can be applied to detect and respond to the fact that the ship is approaching the desired aiming point. As it begins to turn, even though the error signal may still be substantial, it is observed that the error is decreasing. This implies that a smaller control force should now be applied, or even that the control force should be reversed to overcome the ship's inertia. The control equation becomes:

$$F = G_p E + G_i \int E + G_d \Delta E$$

The differential term is not limited to reducing the control force as the error decreases, but also adds to the control force when the error is observed to increase. In a system subjected to disturbances the differential term may dominate the result, maintaining such small errors that the proportional term is generally very small and the integral term is almost constant.

Applying the differential control to a system having as little inertia as the capacitance in our PWM voltage control problem has very little effect. For the student interested in pursuing Proportional - integral -Differential Control (PID) it is suggested that the filtered voltage returned by FILTR be used as the input to the closed loop control equations. Here FILTR gives the effect of a system with large inertia.

7.5 Summary

In the preceding sections the most important concepts of feedback control systems have been demonstrated. Although controlling the voltage on a capacitor is a trivial and perhaps unexciting example, it gave us an easy way to observe and measure the behavior of the system. We have seen the response to a disturbance with proportional control; the need for integral control to provide a steady state force when external conditions and some of the problems such as overshoot, oscillation, and arithmetic overflow. Chapter 7 will apply the same principles to control of a motor.

THIS PAGE INTENTIONALLY LEFT BLANK

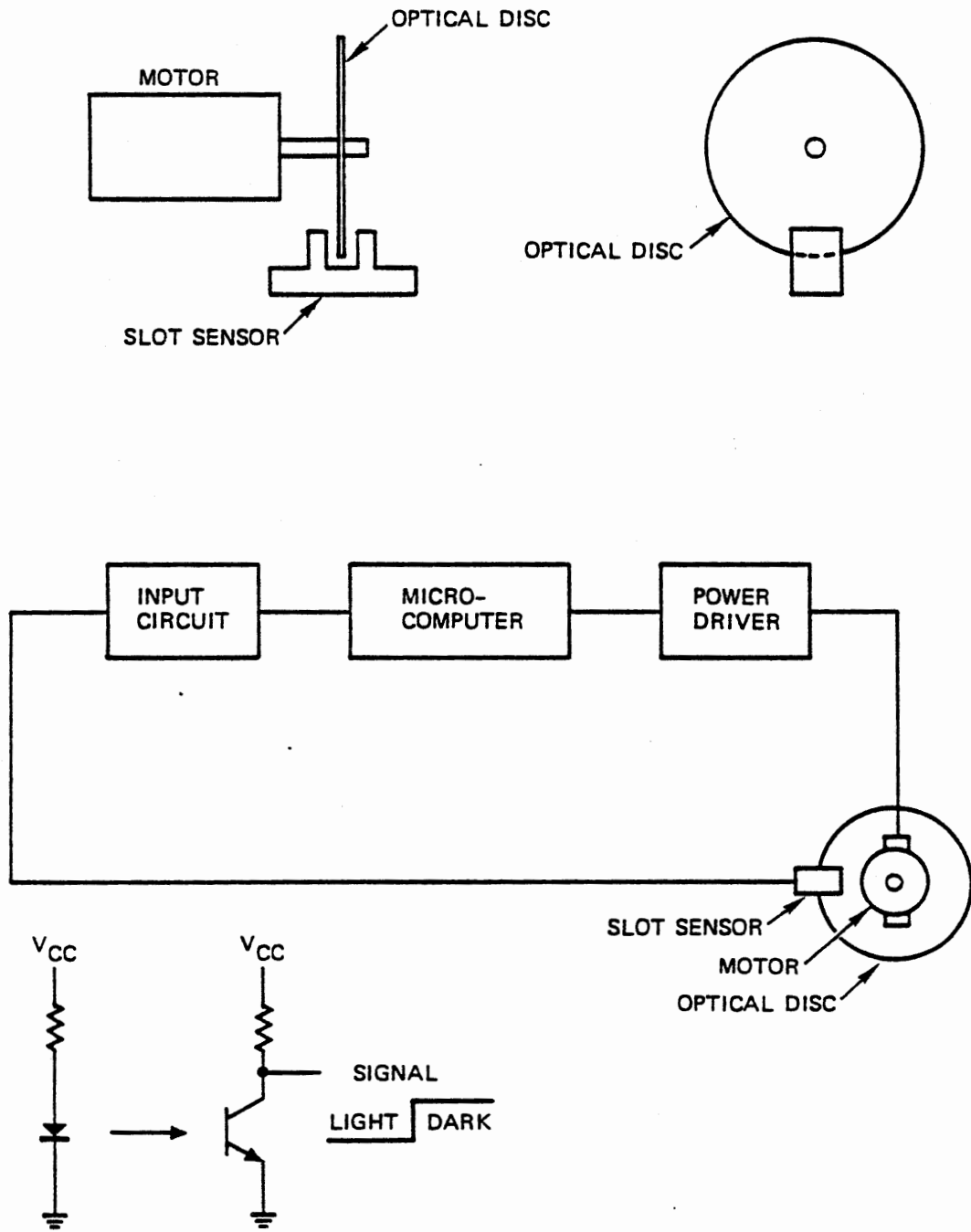
MICROCOMPUTER INTERFACING WORKBOOK

CHAPTER 7

MOTOR CONTROL

7. MOTOR CONTROL

Power to a dc motor is to be controlled to set its speed. We will develop a technique to measure speed, using open loop control of power by pulse width modulation; then we will close the loop using proportional plus integral control. Both the program itself and our development of it will resemble the pulse width modulated voltage control of Chapter 6.



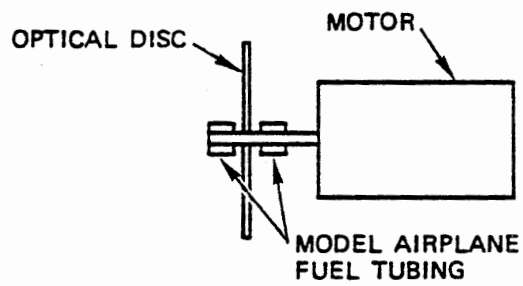
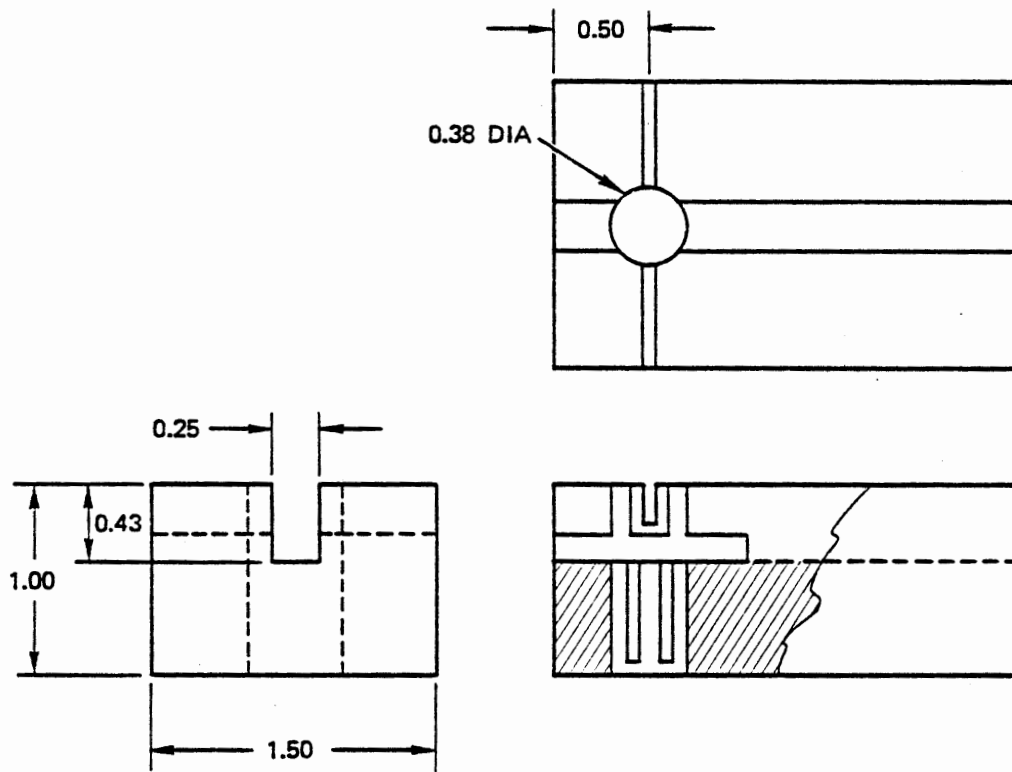
MOTOR AND SLOT SENSOR

Figure 7-1

7.1 OPTICAL DISC AND SLOT SENSOR

Motor speed will be measured by observing an optical disc with transparent and opaque segments rotating between a light emitting diode and a phototransistor. Figure 7-1 suggests the physical arrangement and the system block diagram.

The "slot sensor" combines an infrared light emitting diode aimed at a phototransistor across a gap of 0.1 inch. When power is applied to the LED and the phototransistor as shown in Figure 7-1, the infrared light falling on the base of the phototransistor turns it on just as base current would in a normal transistor. The transistor current then drives the output signal low. When the light is blocked the phototransistor is turned off and the output signal becomes high.



MOTOR, SENSOR AND DISC MOUNTING

Figure 7-2

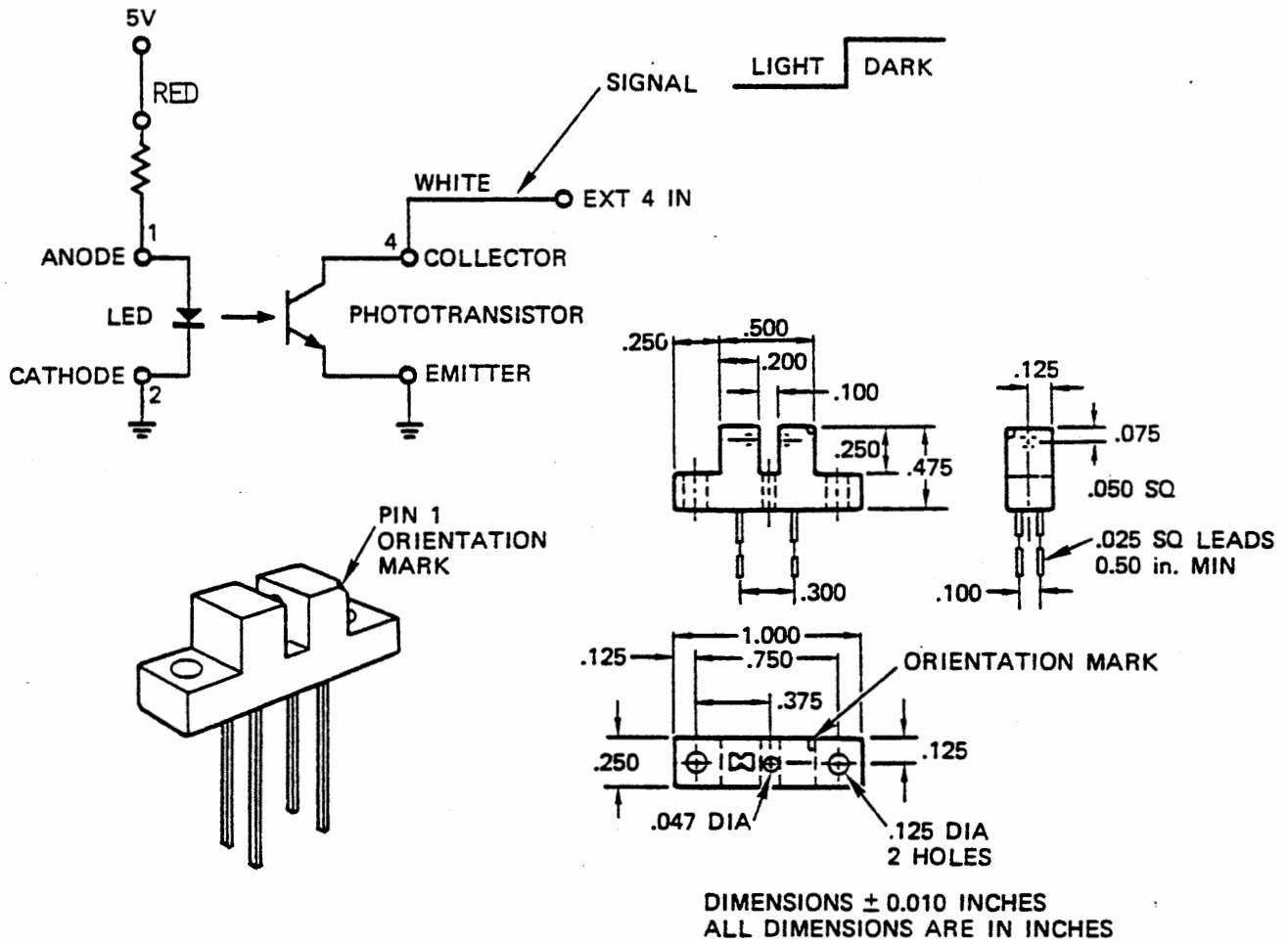
7.1.1 Motor, Sensor and Disc Mounting

The motor and slot sensor can be mounted on a block of styrofoam or balsa wood. Drill a 3/8 diameter hole through the block for the slot sensor leads. Cut a slot wide enough to grip the sensor (.25 inch) and deep enough to make the top of the sensor flush with the top of the block. This need not be at all precise.

Now cut a narrow slot across the width of the block to accept and guide the optical disc through the slot sensor opening.

Mount the optical disc on the motor in either of two ways. You can cut a small X at the center of the disc and force the motor shaft through that hole. The springy mylar will grip the shaft. If you find that the disc slips on the shaft, a small piece of tubing can be squeezed together to grip the disc.

If you have cut the slot to fit the sensor closely it will hold it with no other mounting. If it is too loose, build up the projecting ends of the sensor with Scotch tape. The motor can be held in place on top of the block with tape or rubber bands.



CABLE			
PIN	FUNCTION	COLOR	CONNECTION
1	LED ANODE	RED	+5 VOLTS
2	LED CATHODE	BLACK	GROUND
3	EMITTER	ORANGE	GROUND
4	COLLECTOR	WHITE	EXT 4 IN

OPTICAL SLOT SENSOR

Figure 7-3a

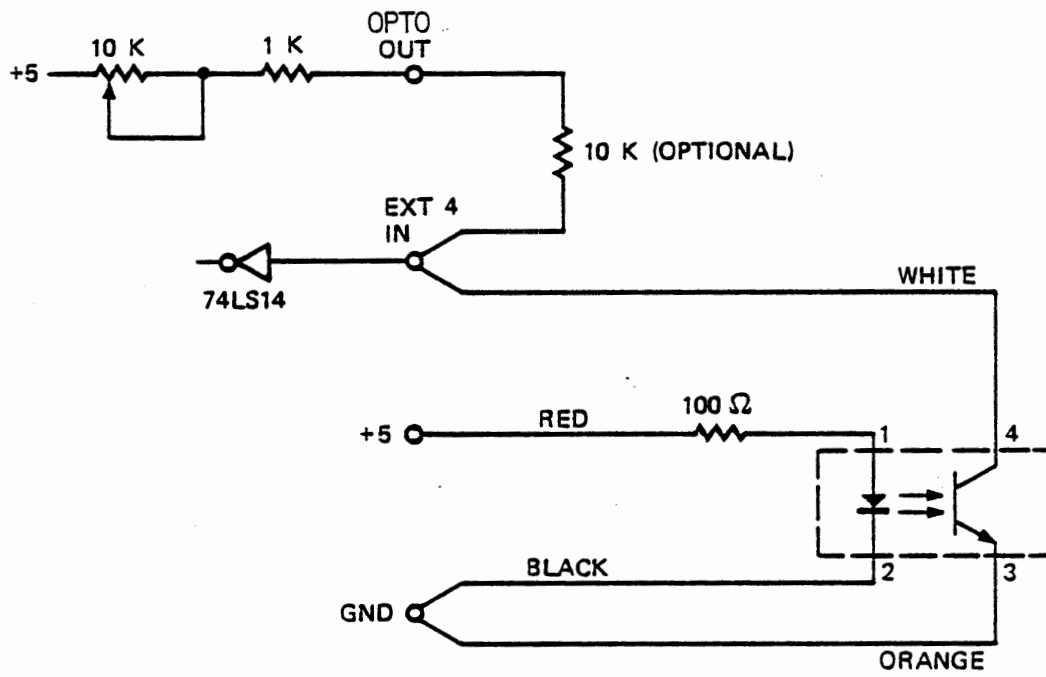


Figure 7-3b

7.1.2 Slot Sensor Connections and Test

The light emitting diode and phototransistor of the slot sensor are shown in Figure 7-3a. The pin numbers and color codes of the cable are shown. Note that a 100 ohm resistor is built into the red lead of the cable, so no external resistor is needed. Check that the cable is wired correctly, and plug the cable ends into the tie blocks as indicated in Figure 7-3b. Be very careful about these connections, because the slot sensor can be damaged or destroyed by reverse voltages.

Connect a 10K ohm resistor between EXT4 and OPTO OUT. Together with the OPTO SENSE pot this provides a pullup resistance from EXT4 to ground.

Now when a clear segment of the disc lies in the light path of the slot sensor, infrared light from the LED falls on the phototransistor and pulls the output signal close to ground. When a dark segment interrupts the light the phototransistor turns off and the voltage is pulled up close to 5 volts. Observe this with the voltmeter.

The 74LS14 Schmitt trigger circuit at the EXT4 input requires that the signal switch below 0.6 volts to guarantee correct operation. The slot sensor may not have enough gain to achieve this with the pullup resistance. If it does not, remove the connection to OPTO OUT. Now the phototransistor will pull the input down when a clear segment is observed. When a dark segment is present the internal pullup resistance of the Schmitt trigger will pull the input voltage up to

about 1.2 volts and the dark segment will be recognized. The circuit will operate correctly without the external pullup resistor, but will be more sensitive to noise pickup.

A test program is given in Figure 4-4. Connect EXT4 to ANALOG IN in addition to the pullup resistor, and remove the voltmeter. The test program displays the state of the EXT4 input in LED number 6, and uses the automatic A/D input to measure and display the voltage. With this program you can observe the switching and the actual voltage at the input as you rotate the optical disc and adjust the OPTO SENSE pot. Be sure that the voltage goes below 0.60 (3C hex) when a clear segment is observed. If it does not, remove the connection to OPTO OUT or substitute a higher valued resistor.

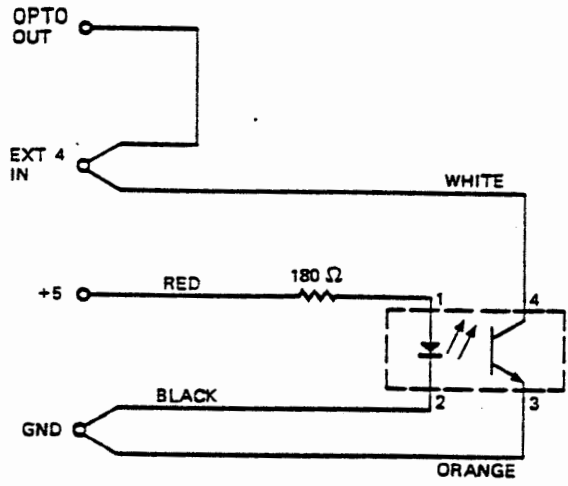
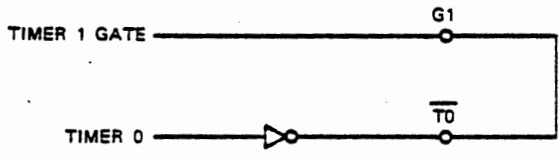
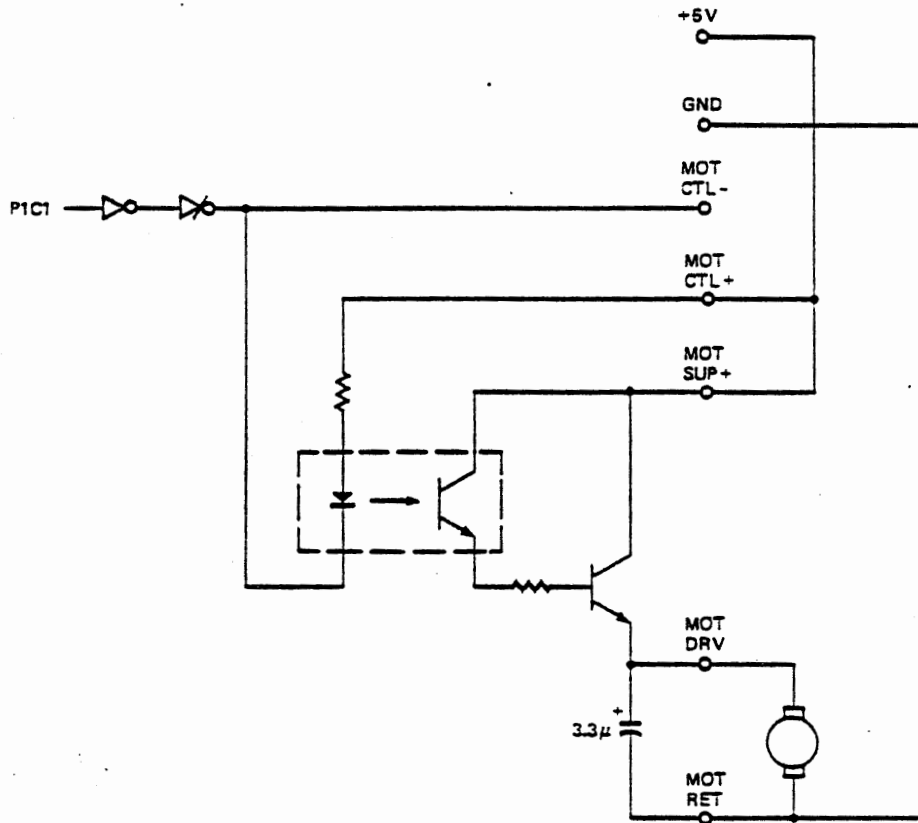
When a dark segment appears in the slot, the voltage must switch above 2.0 volts if an external pullup resistor is used. Without any pullup, the Schmitt trigger should clamp the voltage at about 1.2 volts.

Be sure that the slot sensor operates correctly before going on with the program development. Unreliable operation of the detector will result in a useless control system.

TEST FOR SLOT SENSOR - A/D INTERRUPT 7 - 11

	A	D	D	R	CODE					
CODING SHEET	8	2	3	0	F5		PUSH	PSW		
				1	DB		IN	PORT1B		
				2	05					} (L) ← voltage
				3	6F		MOV	L, A		
				4	3E		MVI	A, 06		Reset
				5	06					
				6	D3		OUT	CNT2		
				7	0F					
				8	7D		MOV	A, L		
				9	CD		CALL	DBYTE		
MICROCOMPUTER TRAINING SYSTEM	A				95					
	B				02					
	C				F1		POP	PSW		
	D				FB		EI			
	E				C9		RET			
	F									
	8				0					
					1					
					2					
					3					
INTEGRATED COMPUTER SYSTEMS				4						
				5						
				6						
				7						
				8						
					0					
					1					
					2					
					3					
					4					

Figure 7 - 4b



MOTOR CONNECTIONS

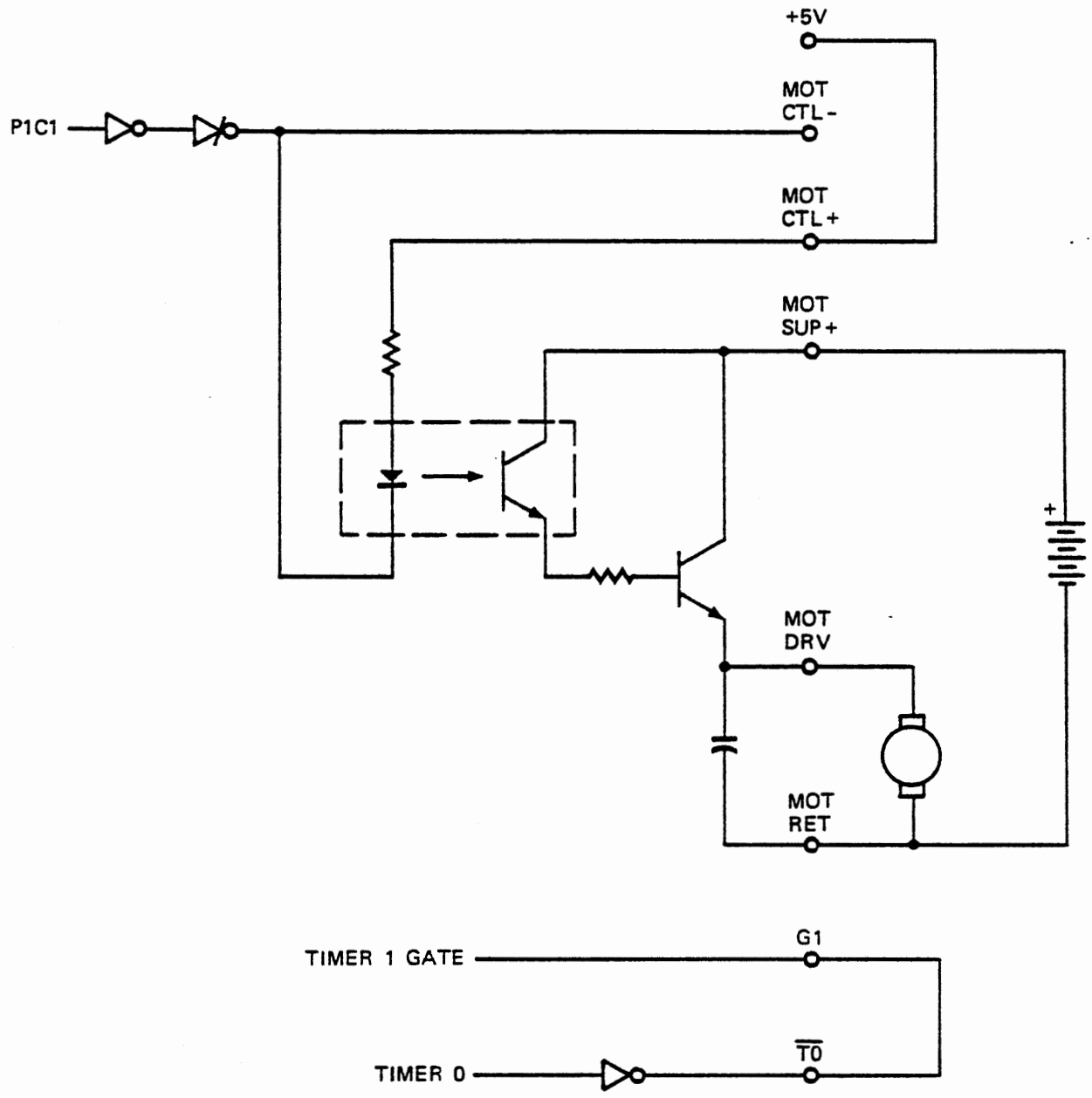
Figure 7-5

7.1.3 Motor Connection

The motor will be driven from the five volt supply using the power transistor as a switch for pulse width modulation control. Figure 7-5a shows the circuit connections for the motor, while Figure 7-5b shows the connections for the slot sensor.

The power transistor and the optical coupler that drives it are isolated from the system power supplies to allow use of an external supply. For this experiment you can use the five volt system supply, although in general it is very poor design practice to place a noise generating load such as a motor on the computer's regulated supply. Figure 7-6 shows how the connections would be made with an external power source such as a lantern battery. Note that here there is no electrical connection between the motor and the computer.

Port 1C1, and output of an 8255, drives an inverter and an open collector inverter to control the optical coupler. When 1C1 is set low the open collector inverter draws current through the LED of the optical coupler. Infrared light from the LED turns on the phototransistor, which in turn provides base drive to the power transistor. This allows current to flow in the motor circuit either from the system power supply (as in Figure 7-5) or from the external supply (Figure 7-6).

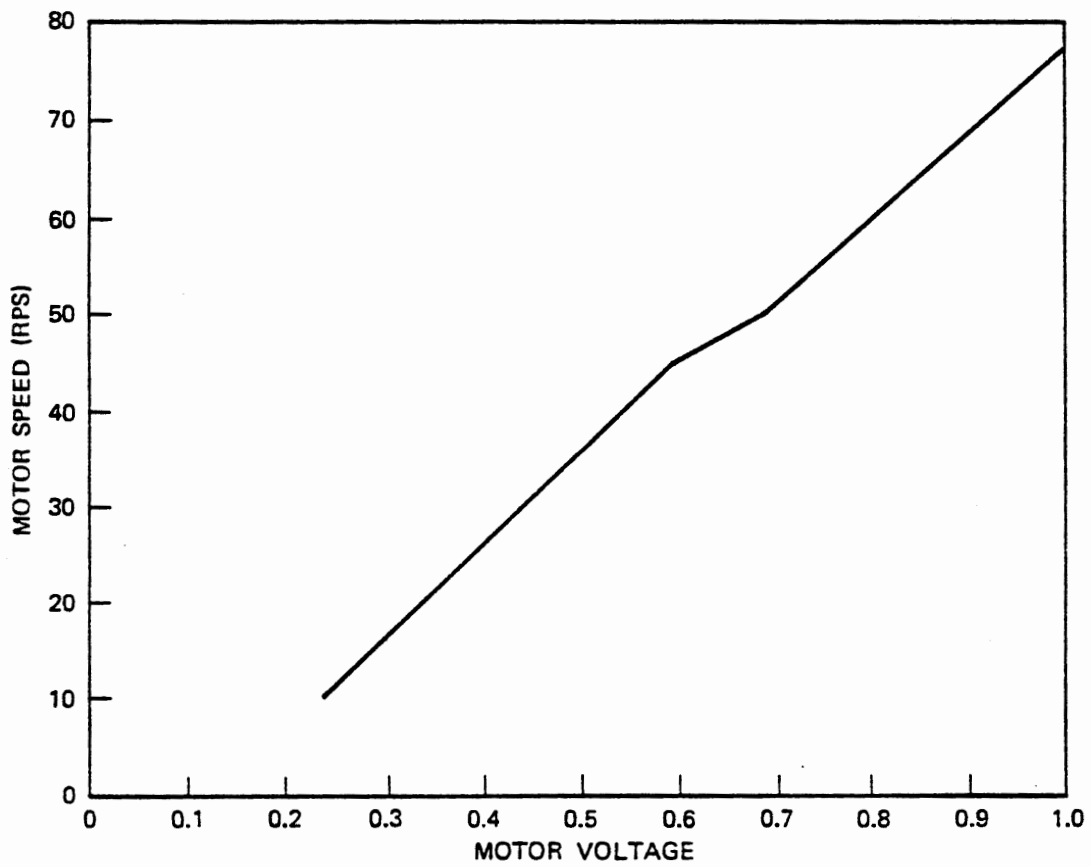
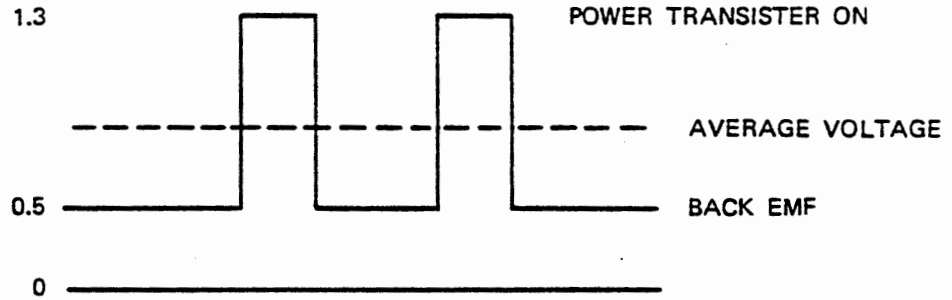


MOTOR CONNECTIONS WITH EXTERNAL POWER

Figure 7-6

Port 1C1 is set to input mode by system reset. In this condition it appears as a high input to the first inverter, so the open collector output is in the high impedance state and the optical coupler is switched off, so the motor does not run. When Port 1C is programmed for output during initialization its outputs become low, and power is applied to the motor. In general the initialization procedure should set Port 1C1 high fairly soon after the port has been programmed, so that the motor will not be turned on until intended.

Figures 7-5 and 7-6 also indicate that a connection is to be made from Timer 0 output to Timer 1 gate, for PWM control.



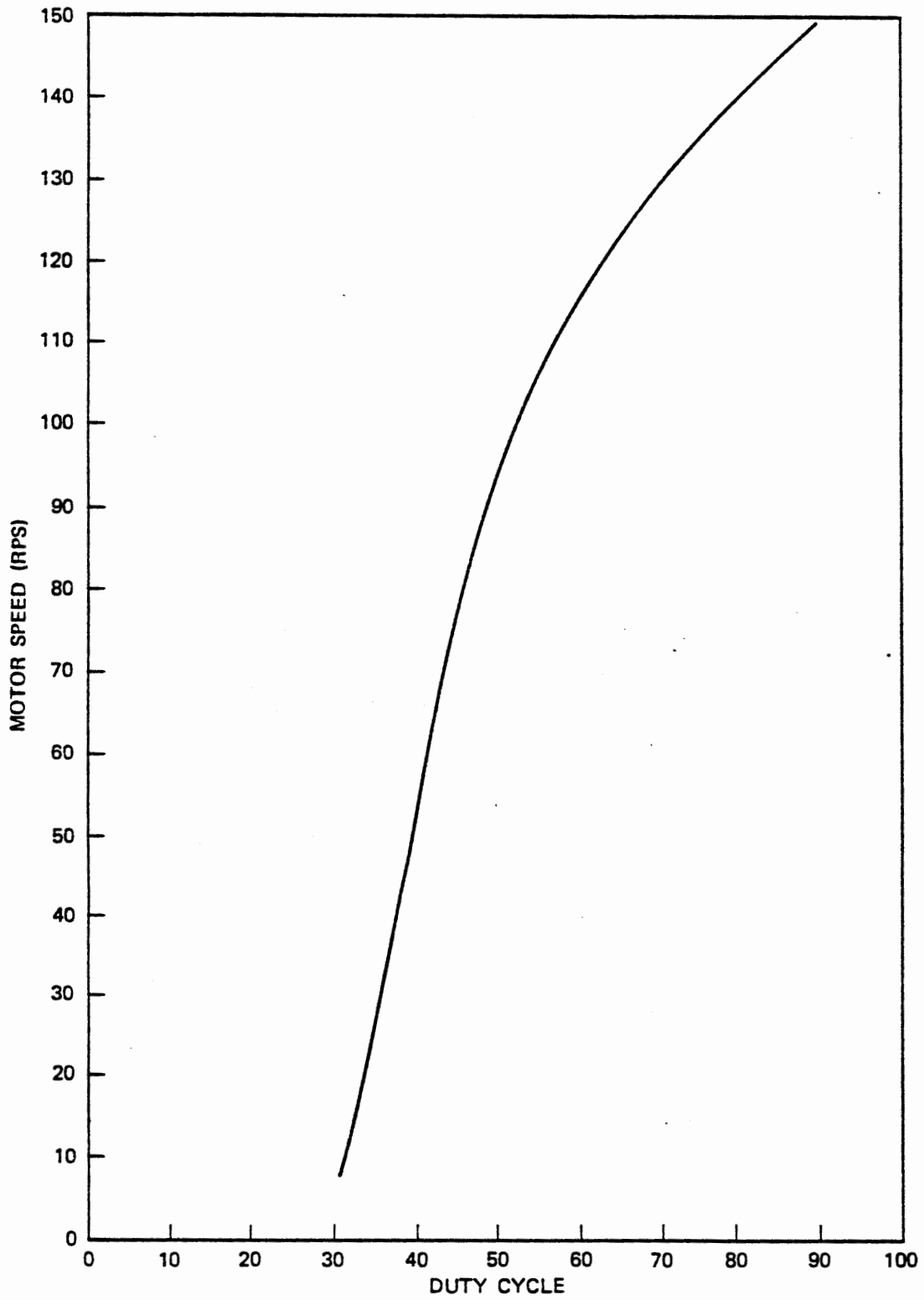
MOTOR SPEED VS VOLTAGE
(Measured with closed loop control)

Figure 7-7

7.1.4 Motor Characteristics

The motor is a three pole commutated dc motor. Its speed with constant load is approximately proportional to the voltage across the motor, as indicated in Figure 7-7. The anomaly in Figure 7-7 in the vicinity of 0.6 to 0.7 volts is related to effects of synchronism between the driving pulses and the motor commutation. The data for Figure 7-7 was taken with the closed loop control program that is developed in this section. Other control schemes would show a linear relationship.

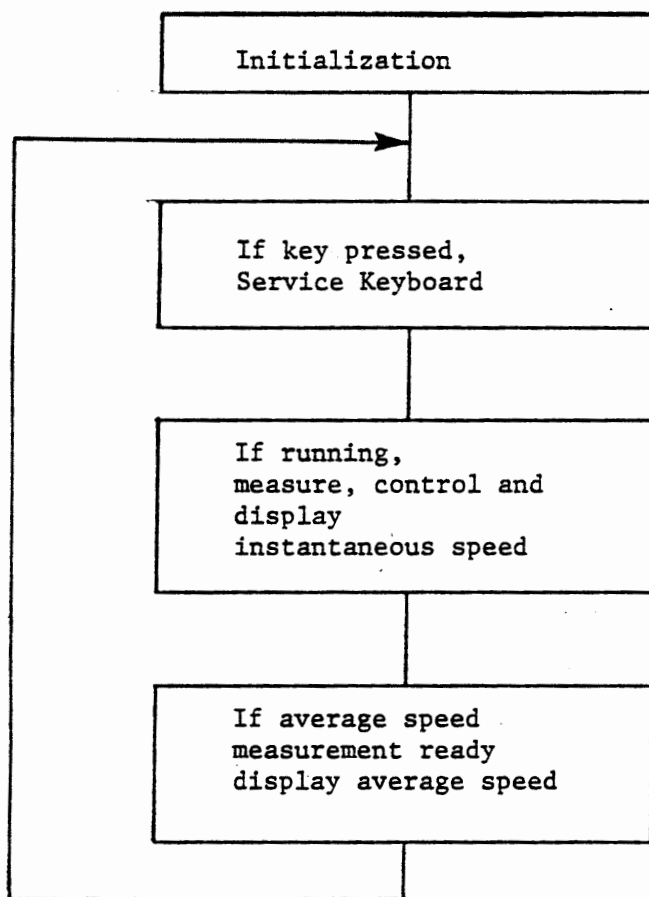
The average voltage measured at the motor is not linear with pulse width (or duty cycle) as the capacitor voltage was in Chapter 6. The motor itself generates a voltage, which depends on its speed. This is called "Back EMF". (EMF stands for ElectroMotive Force, which means voltage). This voltage is present whenever the motor is running, even though the power transistor is turned off part of the time. The voltage observed at the motor is shown at the top of Figure 7-7.



MOTOR SPEED VS DUTY CYCLE OPEN LOOP

Figure 7-8

In Figure 7-8 the motor speed is shown as a function of pulse duty cycle. This was measured in an open loop control system. A linear relationship exists from 30% to 50% only. Below 30% duty cycle the motor will run only sporadically, while above 50% the slope decreases. No external load was placed on the motor for these tests, but the motor bearing friction represents a substantial load at the higher speeds. With a load on the motor the speed versus duty cycle would be linear over a larger range. Because this toy motor has plastic bearings different motors will behave very differently, and a single motor will change its behavior from time to time. Therefore you cannot expect to duplicate these results with any precision.



Timer 0 Interrupt Service
Turn power on
Count time

Timer 1 Interrupt Service
Turn power off

EXT4 Interrupt Service
Count segments
Measure segment interval

MOTOR CONTROL PROGRAM STRUCTURE

FIGURE 7-9

7.2 CONTROL SYSTEM DEVELOPMENT

We will develop a closed loop control system following the general design structure and development approach that were used in the preceding chapter for voltage control. In this program keyboard commands permit setting a pulse duty cycle for open loop control or setting a desired voltage for closed loop control. There is provision for setting gains in the proportional plus integral control equation. The motor can be started by RUN and stopped by STEP.

As in the voltage control system the program comprises initialization, interrupt service, and a main loop which calls various subroutines as needed. The subroutines are listed in the table below. A keyboard service module is called when the main loop detects a key being pressed. If the motor is running a speed control subroutine is called once during each pass through the main loop, and the instantaneous speed is displayed. Each time a new average speed measurement is completed the main loop calls DWORD to display the average speed.

Pulse width modulation is used for power control. Timer 0 runs continuously in mode 2 to define the pulse frequency, or total period. Power to the motor is turned on by the Timer 0 interrupt. Timer 1 operates in mode 5 and is triggered by the output of Timer 0. The interval loaded to Timer 1 sets the pulse width (on-time) and its interrupt turns power off.

Both instantaneous and average speed are measured and displayed. The control loop acts on the instantaneous speed measurement. The only

new programming problem is the calculation of instantaneous speed.

Program Memory Assignments

8200 - 27	Initialization
8228 - 3F	Interrupt Manager
8240 - 5F	Timer Interrupt Service
8260 - 7F	EXT4 Interrupt Service
8280 - AF	Main Loop
82B0 - BF	Not used
82C0 - DF	Subroutine SPEED
82E0 - FF	Subroutine WIDTH
8300 - 1F	KYTIM - Entry and Dispatch
8320 - 4F	Key Processing Modules
8350 - 5F	Subroutine DECBI
8360 - 6F	Subroutines SMULT, SCUML
8370 - 8F	Subroutine DIVID
8390 - 9F	Not used

Data Memory Assignments

83A0	Binary Time Count
83A1	Motor Control Byte
83A2	Binary Segment Count
83A3,A4	Decimal Segment Count
83A5,A6	Average Speed
83A7,A8	Timer 2 Data
83A9	Desired Speed
83AA	Integral Gain
83AB	Proportional Gain
83AC,AD	Error Integral

7.2.1 Speed Measurement

Speed is measured by observing the EXT4 interrupts generated by the optical disc. Each time a clear segment of the disc appears between the LED and the phototransistor of the slot sensor, the phototransistor is turned on, its output signal goes low, and an EXT4 interrupt occurs. In Chapter 5 we measured pulse interval time (Section 5.1) and frequency (Section 5.2). Both techniques are used in this program.

Average speed is measured as a frequency, by counting interrupts over a fixed period of time. Since the optical disc has 16 clear segments we will receive 16 EXT4 interrupts per revolution. If we were to count EXT4 interrupts during 1/16 of a second the count would represent revolutions per second. For average speed we would like better resolution, so we will count for 10/16 second, obtaining the average speed in tenths of a revolution per second. Thus a count of 0506 would represent 50.6 rps. For convenience the counting and display are in decimal.

To control the speed well under variable load conditions we need more frequent measurements. Even the 1/16 second interval would be too infrequent for good speed control. Therefore the closed loop control is based on pulse interval measurement, giving "instantaneous speed" by division.

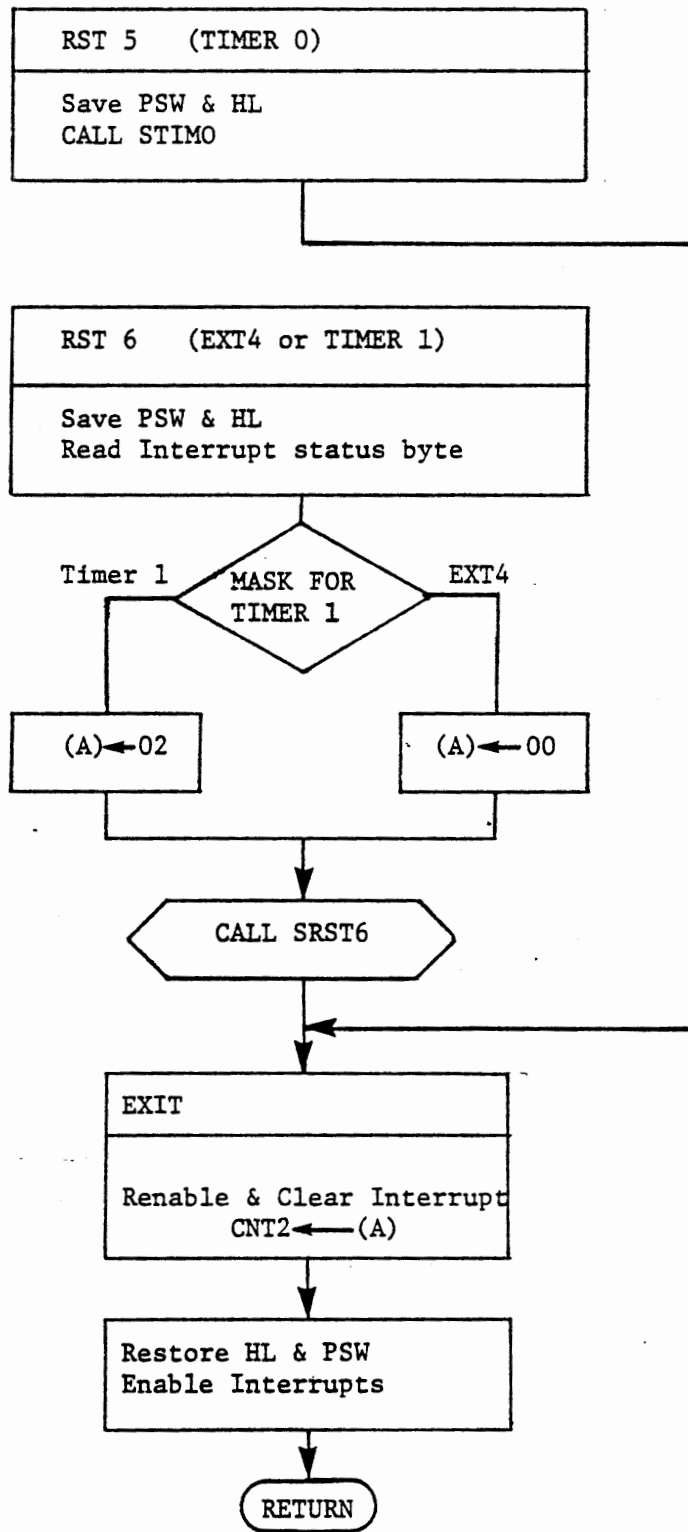
THIS PAGE INTENTIONALLY LEFT BLANK

$$\text{Speed} = \frac{16}{\text{time per segment}}$$

$$\text{Time per Segment} = \frac{\text{clocks per segment}}{2048000}$$

$$\text{Speed} = \frac{16 \times 2048000}{\text{clocks per segment}}$$

The subject of binary division has not been treated previously in this course nor in Course 525. A subroutine, DIVID, is given in the program solution.



MOTOR CONTROL INTERRUPT MANAGER

FIGURE 7-10

7.2.2 Interrupt Service

Three interrupts are used in the motor control system: Timer 0, Timer 1, and EXT4. Timer 0 is distinguished by its vector, RST5. Timer 1 and EXT4 both generate RST6 and must be distinguished by reading and masking the interrupt status byte.

7.2.2.1 Interrupt Manager

Figure 7-10 shows the interrupt manager. Each entry saves appropriate registers (PSW and HL only). A service subroutine STIMO is called at RST5, and then a jump is made to the exit module. At RST6 the same registers are saved, the interrupt status byte is read and masked by:

```

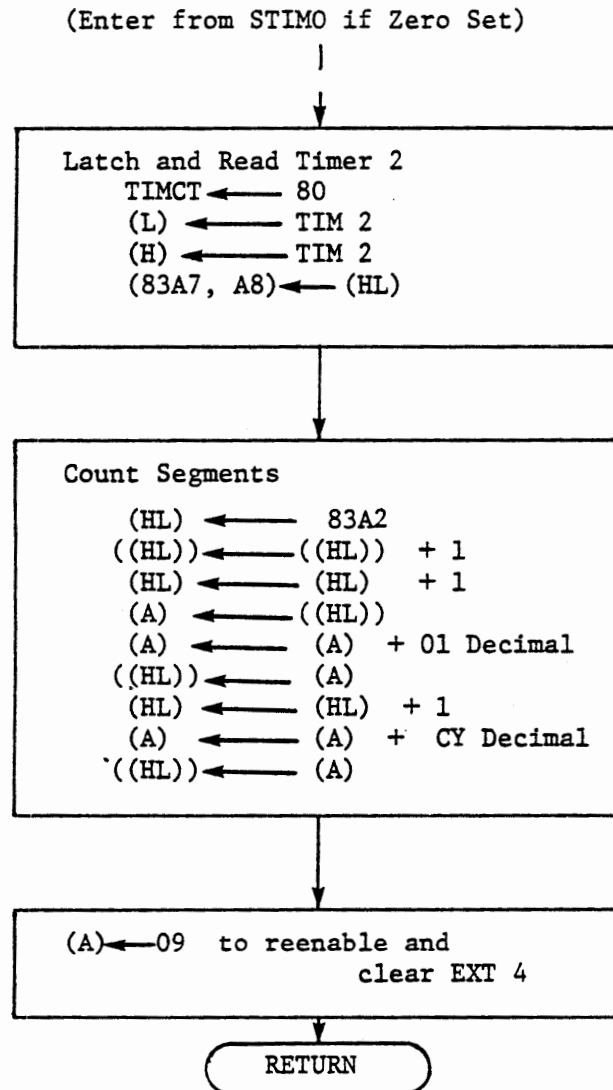
IN      PORT2B
ANI     02

```

Then a service subroutine is called. At entry to this subroutine register A contains 02 if Timer 1 generated the interrupt. Otherwise (A) = 00 and the zero flag is set. The subroutine executes a jump if zero to service EXT4, or proceeds directly to service Timer 1.

Each service subroutine must return in register A the necessary control byte to clear and reenables its interrupt flip flop. The exit module writes this byte to CNT2, restores the environment, and returns to the interrupted instruction.

Note that the service subroutines are restricted to using registers H, L, A, and the flags.



EXT4 INTERRUPT SERVICE

FIGURE 7-11

7.2.2.2 EXT4 Service

EXT4 interrupt occurs each time a dark segment of the optical disc appears between the LED and phototransistor of the slot sensor. EXT4 service performs two functions in response. It latches and reads Timer 2, which is running continuously, and stores the data for use by the speed measurements subroutine. In that subroutine (called by the main program loop) we subtract the latest measurement from the preceding measurement to obtain the time difference. We could clear and restart Timer 2 at each EXT4 interrupt, thereby making each measurement complete in itself. We will see later that this would require more rather than fewer instructions in the speed measurement subroutine, as well as requiring additional instructions here.

Note an interesting point with regard to the mode selected for Timer 2. We usually think of mode 2 for a timer which is to run continuously. When no output signal or interrupt is needed we can use mode 0 instead, because the timer continues to count down after it reaches zero, although its output will remain high after the first time it reaches zero. Mode 0 is used in this program to demonstrate the point.

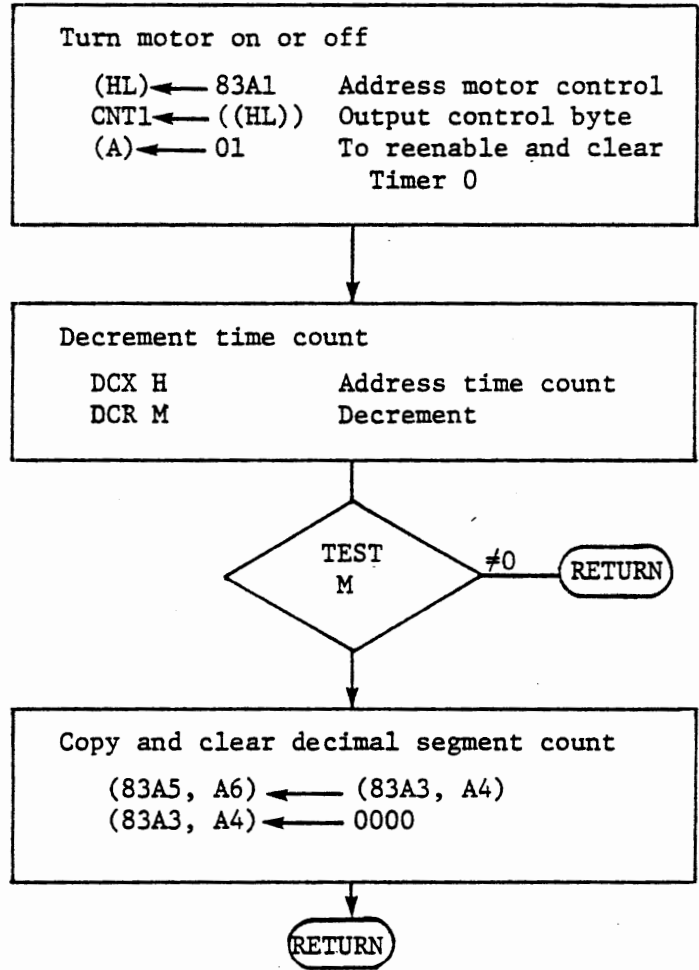
THIS PAGE INTENTIONALLY LEFT BLANK

7.2.2.3 Timer Service Subroutines

Pulse width modulation is controlled by Timer 0 and Timer 1 interrupts. Their service routines turn the power transistor on or off by writing a byte to Port 1C. If the byte is 00 it sets Port 1C1 low and turns the transistor on, while 02 sets Port 1C1 high and turns the transistor off. As in the previous program the Timer 0 interval is constant and defines the total period. If the motor is running, Timer 0 turns the power transistor on. The Timer 0 output also triggers Timer 1, whose interval sets the on-time. The Timer 1 interrupt turns the transistor off. Timer 1 has no other function.

Timer 1 Service

JZ	SXT4	If zero set service EXT4
OUT	PORT1C	Output 02 to turn motor off
MVI	A,03	To reenale Timer 1
RET		Exit



TIMER 0 SERVICE

FIGURE 7-12

Timer 0 has several functions. Since we often will want the motor to be stopped, the control byte to be written to Port 1C is stored in memory (at 83A1) by keyboard command. RUN stores 00 and STEP stores 02. This byte is loaded from memory and output at each Timer 0 interrupt. In addition Timer 0 decrements a time counter (at 83A0) and at zero it copies and clears the decimal segment count accumulated by EXT4 interrupts. The timing is arranged so that this count directly represents average speed. The completed count is stored at 83A5,A6 for display by the main loop.

There is a fortuitous time interval that is suitable for the PWM total period and also for measuring the 10/16 second interval for average speed. 256 counts in the binary timer counter equals 10/16 second if the PWM total period is set at:

$$\frac{10}{16 \times 256} = .00244140625 \text{ second}$$

although this may appear to be an awkward time interval to obtain, in fact the system clock generates it very easily

$$\frac{5000}{2048000} = .00244140626 \text{ second}$$

timer 0 is programmed in mode 2, decimal, high byte only, and loaded with 50. The interval, slightly less than 2.5 milliseconds, is quite suitable for pulse width modulation.

7.2.3 Initialization

Ports and Timers are to be programmed as follows:

```

8255 #1 A out B out C out
8255 #2 A in  B in  C out
Timer  0      High byte, mode 2, decimal
Timer  1      Both bytes, mode 5, binary
Timer  2      Both bytes, mode 0, binary

```

timer 0 is to be loaded with 50 (high byte) to generate the 2.44 millisecond total period. Timer 2 must be loaded (in both bytes) to start it; the value does not matter since it will always count down from zero after it first reaches zero.

Recall that the motor is off when Port 1C1 is high, and running when that output is low. After system reset all ports are programmed for input, which gives an apparent high output and turns the motor off. As soon as Port 1C is programmed for output its output signals go low, the power transistor is turned on and power is applied to the motor. (You can demonstrate this by stepping through your initialization procedure). Since none of the control data have been entered we want to stop the motor during initialization; this can be done by a call to the CLR key processing module in KYTIM. The command keys are defined in Section 7.2.5.

We have used RST5 and RST6 commands in the past to enable interrupts. This would work for RST5 in this program, but two calls to RST6 service would be required to enable both EXT4 and Timer 1. Even with

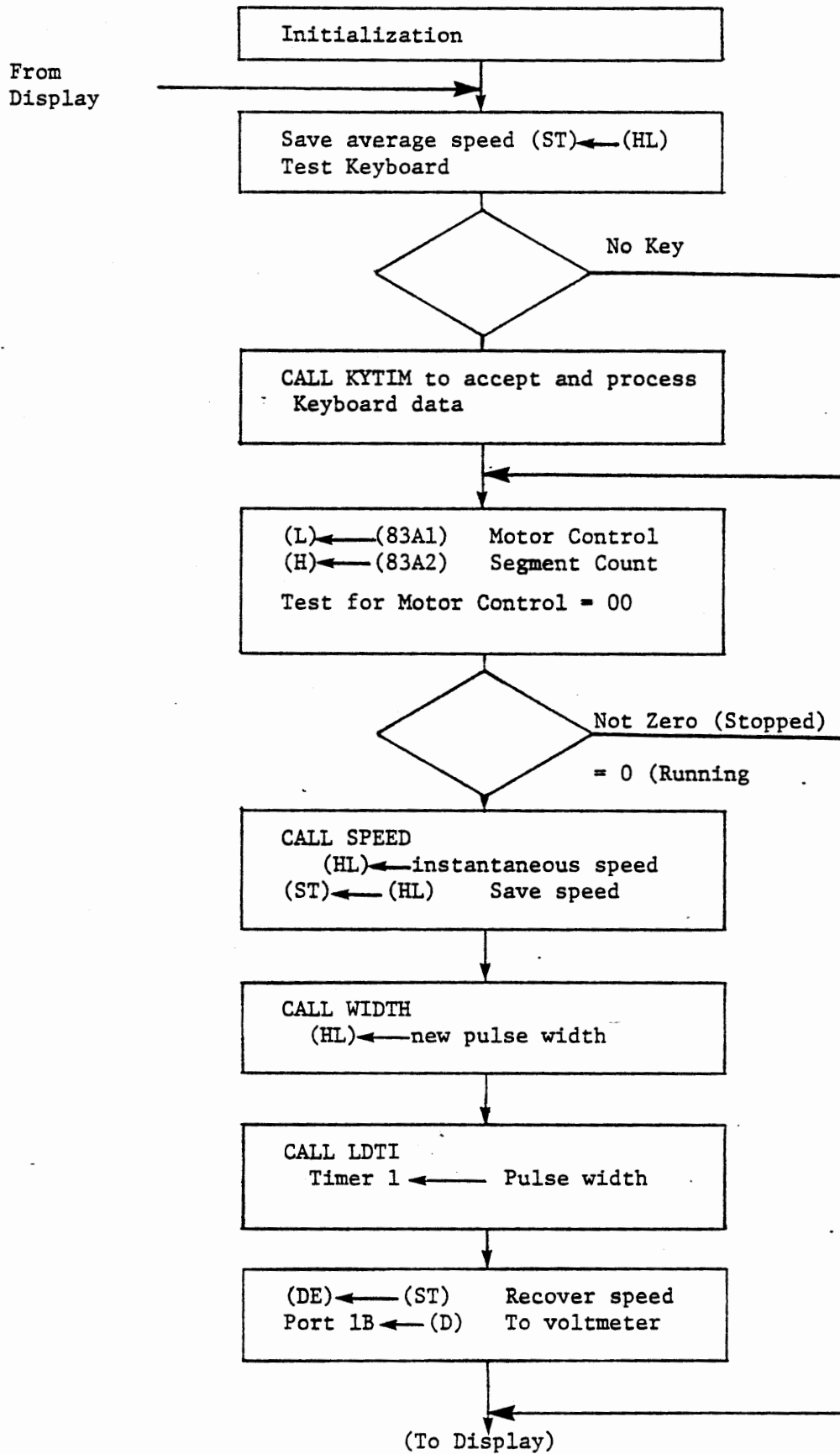
two calls there is no certainty that Timer 1 would be enabled because it is only serviced if its interrupt is present. Therefore we enable the interrupts by writing 13 to Port 2C. This usually results in all three interrupts being serviced immediately, since writing to Port 2C does not clear the interrupt flip flops. During debugging it is desirable to replace this process with the RST instructions, which will permit stepping through the interrupt service routines.

Final			Debug	
3E	MVI	A,13	EF	RST5
13			F7	RST6
D3	OUT	PORT 2C	F7	RST6
0E			00	NOP

While you step through the initialization the motor will run at full speed until the CLR function stops it. It is advisable to unplug the motor during this task.

You may have to force the service of Timer 1 by replacing the interrupt status byte after reading it. (Use REG, A, 02 after the status byte has been read by the IN PORT2B instruction).

When debugging of the initialization and interrupt service routines has been finished, replace the RST instructions with the load and output instructions. These are followed by a jump to the main loop.



MOTOR CONTROL - MAIN LOOP

FIGURE 7-13a

7.2.4 Main Program Loop

Figure 7-13 shows the main program loop. This is a more detailed version of Figure 7-9.

At the start of the loop register pair HL normally contains the average speed. (After initialization this value is meaningless). This value is saved, and the keyboard is tested by:

```

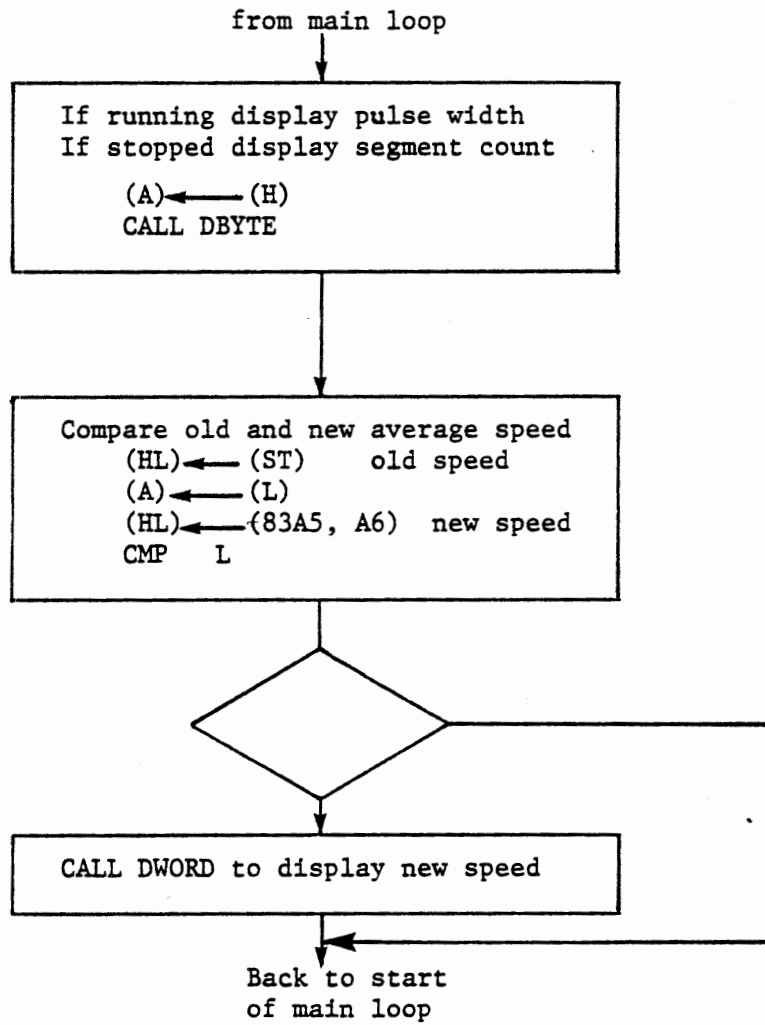
IN      PORT0A
INR     A
CNZ     KYTIM

```

Note that while Figure 7-13a indicates a conditional jump, a conditional call is actually used here.

As in the voltage control program the process operates in open loop mode while KYTIM waits for keyboard entry and processes the data entered.

The motor control byte stored for Timer 0 interrupt service is also used here to determine whether the motor is running. If it is stopped (control byte = 02) we skip all of the control functions and go to display the binary segment count. If the motor is running three subroutines are called. SPEED finds the instantaneous speed from a segment interval; WIDTH calculates a new pulse width; LDT1 loads Timer 1 with the new width.

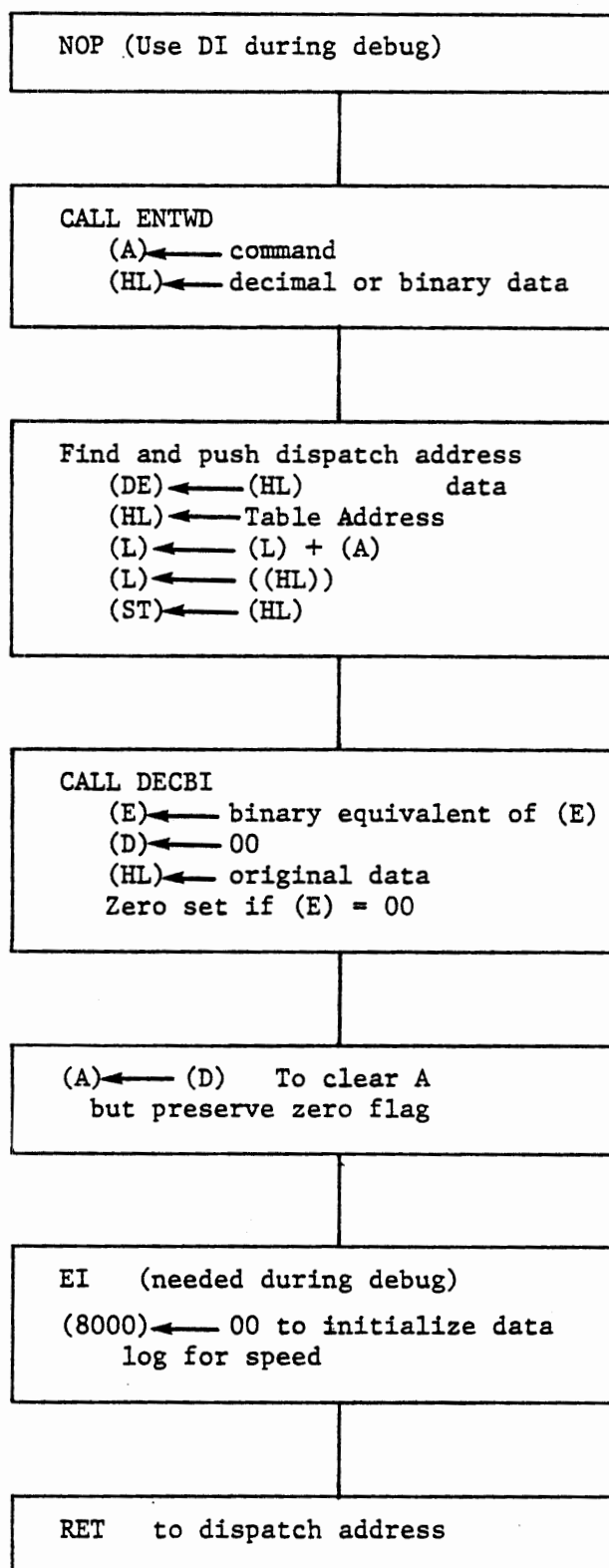


MOTOR CONTROL - DISPLAY

FIGURE 7-13b

The high byte of the instantaneous speed is output to the D/A converter. This allows observation of the speed with a voltmeter, which is interesting to watch when a load is placed on the motor. The high byte of the pulse width is displayed at the right, as shown in Figure 7-13b.

After displaying the pulse width or the segment count the main loop may display the average speed. The average speed is stored by Timer 0 interrupt once every $5/8$ second. Since the optical disc has 16 light segments, the decimal segment count during the $5/8$ second directly represents speed in tenths of a revolution per second. Thus a display of 0506 means 50.6 rps. To avoid wasting time in the control loop the program tests for a change in the average speed before displaying it. To permit this test the old value is stored at the beginning of the main loop and recovered before the (possibly) new value is loaded. The result is that DWORD is called only once in about 250 passes through the main loop. (It is sufficient to compare the low bytes of the old and new speeds, since it is unlikely that successive measurements will be alike in the low byte).



DYTIM - INPUT AND DISPATCH

FIGURE 7-14

7.2.5 Keyboard Input Subroutine KYTIM

This version of KYTIM uses the same keyboard entry and dispatch techniques used in previous programs. A decimal to binary conversion is required, since some data are entered as decimal values but are needed as binary values. This is done by subroutine DECBI (Section 7.2.6) which is entered with keyboard data in (DE), and returns with the input data moved to (HL) and the binary equivalent of the low byte in (E) and also in (A). Register D is cleared. The zero flag is set if (E) = 00. After return register A is cleared by MOV A,D, so the zero flag is preserved.

If you have 1024 bytes of memory you may want to log the speed for subsequent review. Clear the content of memory location 8000 to initiate a new log.

THIS PAGE INTENTIONALLY LEFT BLANK

The following command key processing modules are needed:

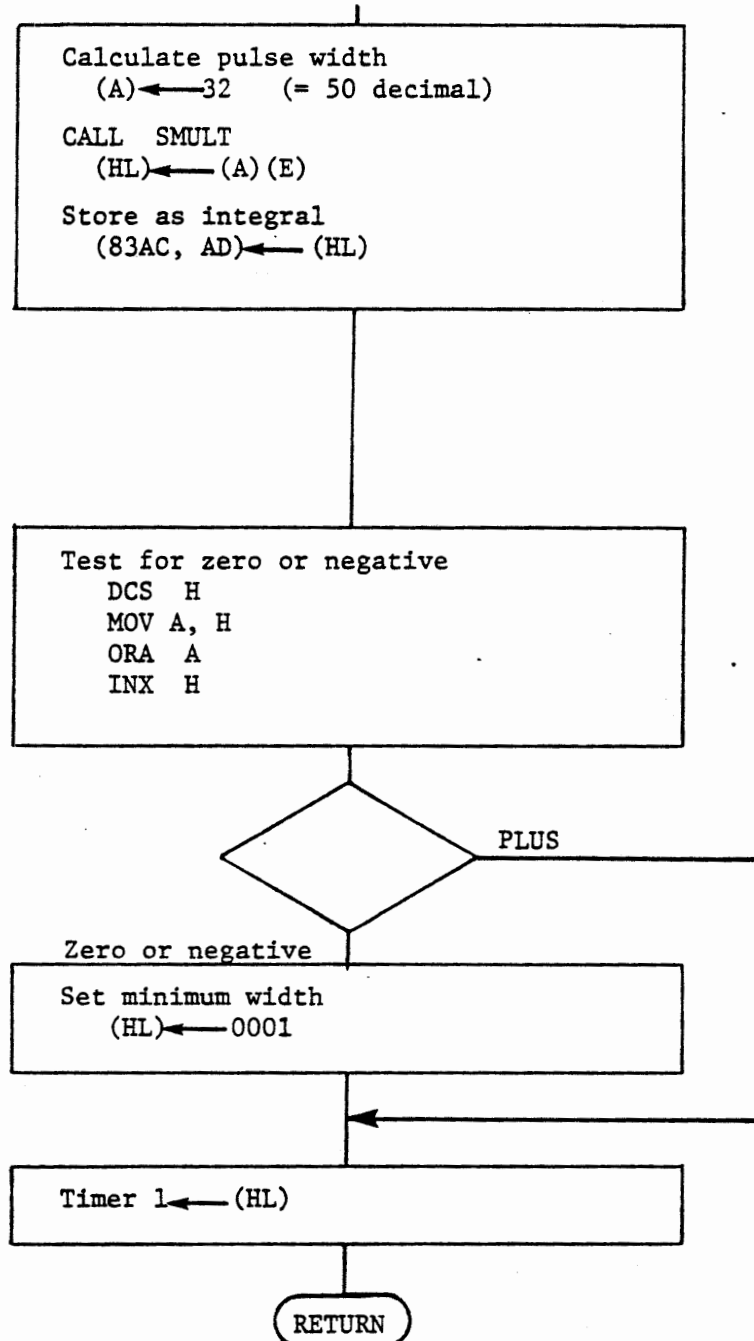
- CLR Clear the binary segment count and stop the motor.
 This permits measuring the stopping distance.
- STEP Stop the motor (store 02 at 83A1).
- RUN Start the motor (store 00 at 83A1).
 If a desired speed is entered, store its binary
 equivalent at 83A9.
- MEM Store integral and proportional gain. These are
 entered in binary, and are to be stored at 83AA and
 83AB. (83AA,AB) <--- (HL)
- NEXT Given a decimal duty cycle (converted to binary by
 DECBI) calculate pulse width. Store the results as
 a new value for the integral of error and load Timer 1.

The total period for PWM has been set to 5000 (decimal) clocks. We can calculate a pulse width by multiplying the duty cycle (treated as an integer) by 50 (decimal). For instance, if a duty cycle of 40% is requested, $40 \times 50 = 2000$, which is 40% of 5000.

NEXT

At entry:
 (E) = binary equivalent
 of duty cycle
 (E) = 00

LDTI



LOAD TIMER 1 MODULES

FIGURE 7-15

In fact the calculation will be done in binary, and the binary result is used. A multiplication subroutine SMULT (Section 7.2.7) returns $(HL) = (A)+(E)$. Since DECBI has given $(E) =$ binary equivalent of the input data, the processing for NEXT is:

```

MVI    A,32           Binary equivalent of 50
CALL   SMULT         (HL) <--- pulse width
SHLD   83AC          Store as integral
LOAD  TIMER 1

```

As in the voltage control program the Timer 1 loading module is also used in closed loop control. It tests for a zero or negative value in (HL) and replaces such a value with 0001 for minimum pulse width. Timer 1 is loaded with the contents of L and H.

7.2.6 Subroutine DECBI

A two digit decimal value can be converted to an equivalent binary value by:

$$\text{Binary value} = \text{Low digit} + 5/8 (\text{High digit})$$

A convenient algorithm for this calculation is expressed as:

$$\begin{aligned} \text{Binary Value} &= \text{Decimal Value} \\ &+ 1/8 \text{ High digit} \\ &- 1/2 \text{ High digit} \end{aligned}$$

For convenience in the motor control program subroutine DECBI accepts and returns data as follows:

ENTER

(DE) = Keyboard data

RETURN

(HL) = Keyboard data

(A) = (E) = Binary equivalent of low byte

(D) = 00

Zero set if low byte is zero

CY clear

(BC) is preserved

A program solution is given in Figure 7-

7.2.7 Subroutines SMULT, SCUML

SMULT multiplies two single byte values and returns the two byte product. SMULT clears the product at entry. SCUML is an alternate entry at which the product is not cleared, allowing cumulative multiplication. Data are entered and returned as follows:

ENTER

(A) = Multiplier

(E) = Multiplicand

(D) = 00 if multiplicand is positive

(D) = FF if multiplicand is the twos complement
of a negative value

(HL) = Previous product (SCUML only)

RETURN

(HL) = Product (A)*(E) for SMULT


```

(HL)    =    (HL) + (A)*(E) for SCUML
(DE)    destroyed
(BC)    preserved
(A)     =     00
Zero    set
CY      clear

```

The multiplication is carried out by repetitively shifting (A) right, and adding the multiplicand to the product if the bit shifted out is a one. After each shift of (A), and after the addition if it is performed, the multiplicand is shifted left. The subroutine returns when the content of A is zero.

A program solution is given in Figure 7-21

7.2.8 Open Loop Operation

With the functions described so far you can operate the system in open loop mode. (Use an unconditional JMP around the control functions instead of JNZ. This permits debugging of the main loop, KYTIM, and interrupt service.) Write all of these program modules. Check the program flow through interrupt service, using RST5 and RST6 instructions. check the program flow through KYTIM and see that the calculations and data storage are correct.

To run the motor enter a high duty cycle (60% or more) with NEXT and then press RUN. See that the motor can be speeded by higher duty cycles and slowed by lower duty cycles. Find the lowest duty cycle that will keep the motor running once it is started, and the lowest

that will cause it to start. Record and plot speed versus duty cycle, and compare your result with Figure 7-8.

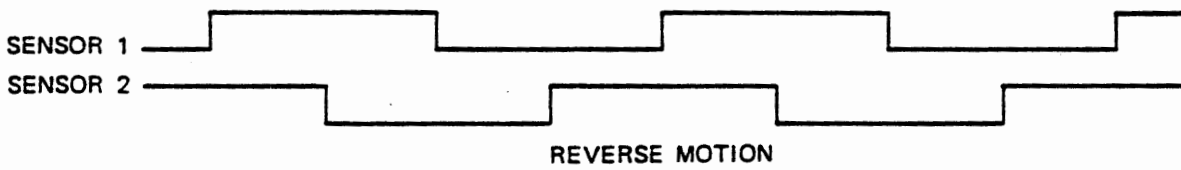
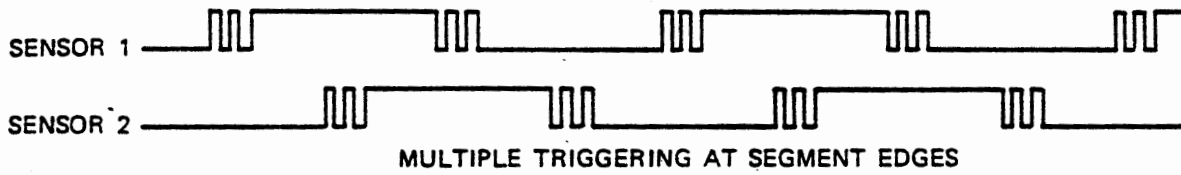
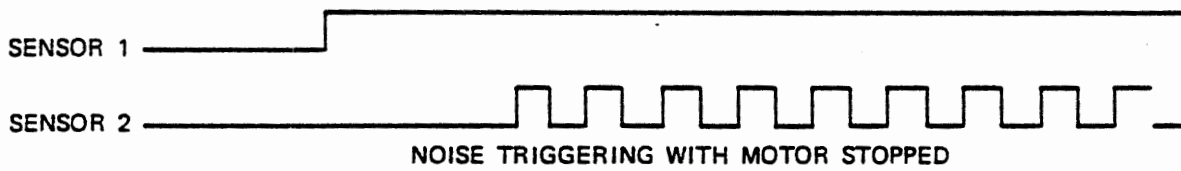
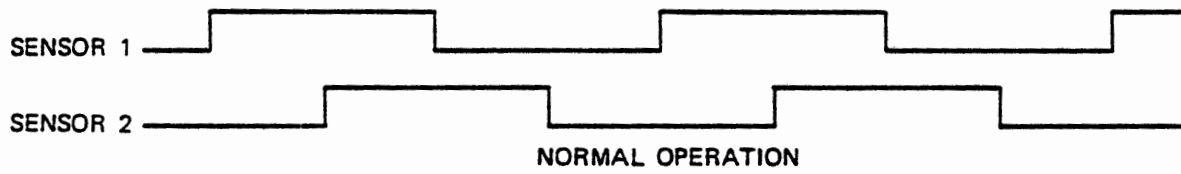
7.2.9 False Speed Indications

When the motor is stopped with an edge of a segment in the optical sensor light path, the phototransistor will be in an active state, neither fully on nor fully off. In this state the circuit is very susceptible to small signals, and noise pickup is likely to cause the EXT4 input to switch. The Timer 0 output, which is connected to Timer 1 gate in this experiment, is a source of such noise. If this source does cause switching, every Timer 0 cycle will result in an EXT4 interrupt. Then the binary and decimal segment counts will constantly be incremented. After 256 counts Timer 0 service will copy and clear the decimal count, so an average speed of 25.6 revolutions per second will be displayed. It is fairly difficult to find the exact sensor position necessary to obtain the continuous counting. If you turn the shaft slowly by hand it is easy to observe multiple counts for each segment, demonstrating that the false switching occurs.

Another false speed indication can occur when power is applied to the motor but the pulse width is insufficient to start it. At each pulse the motor shaft will turn slightly, but will be pulled back by the motor magnets when the pulse ends. If this occurs with a segment edge in the optical sensor light path the phototransistor will switch in time with the motor pulses.

There is no good solution to these problems with a single optical sensor. In any application where such false indications are

intolerable it is necessary to use two sensors to observe the disc. They must be so located that the two sensors cannot both see segment edges at the same time. Now the program can test for a sequence of switching in the two sensors to ensure that only valid segment transitions are observed. Figure 7-16 shows the sequences under various conditions. A program could be designed so that an interrupt from sensor 1 disables itself and enables the sensor 2 interrupt, and the sensor 2 interrupt disables itself and enables sensor 1. Now interrupts can occur only when both sensors switch in sequence. noise and multiple triggering are thereby excluded.



MOTION DETECTION WITH DUAL SENSORS

Figure 7-16

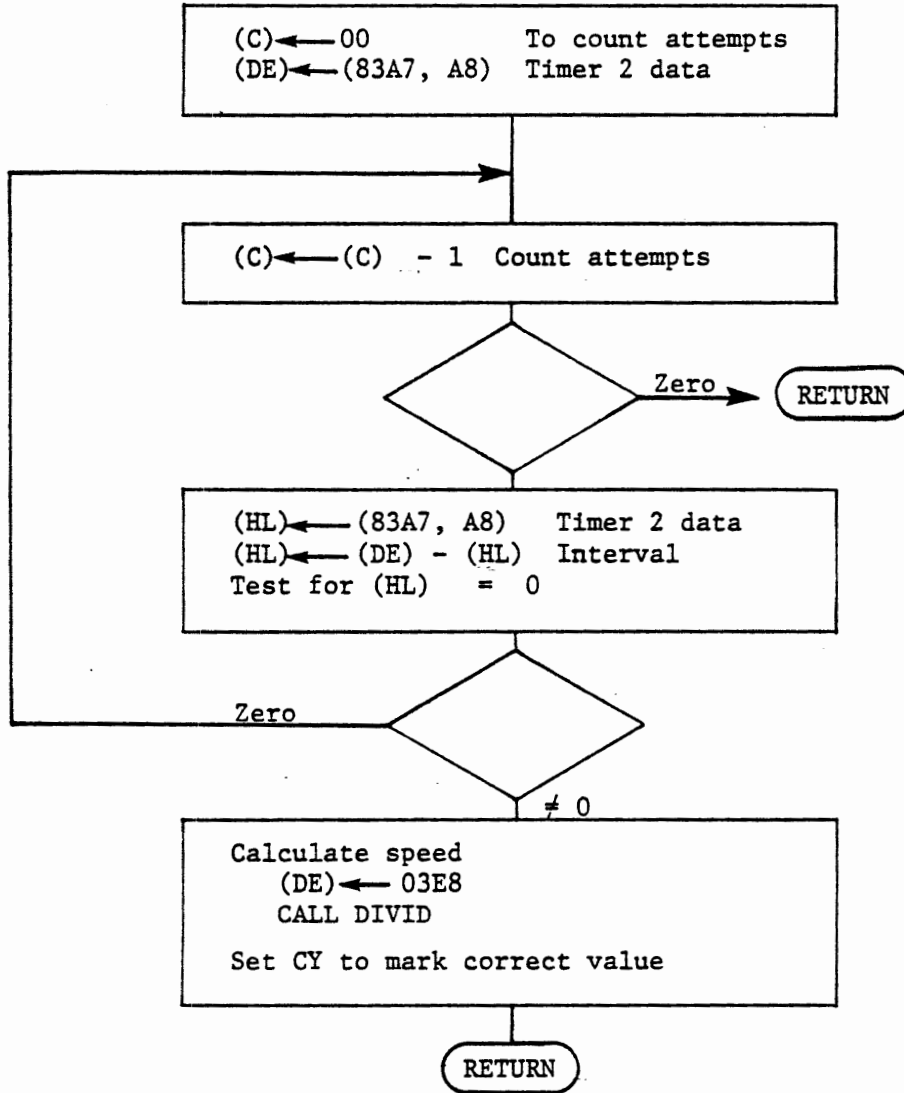
Dual sensors can also be used to detect direction of motion. The difference between the top and bottom diagrams in Figure 7-16 is readily analyzed to determine which way a shaft is turning. Since we have only a single sensor available, we cannot determine direction nor eliminate the false speed indication. This is a problem only when the motor is stopped or running very slowly. At speeds above 5 to 10 revolutions per second the single sensor is accurate. Although some schemes are available to reduce the probability of false speed indications with a single sensor, we will not attempt their implementation.

7.3 CLOSED LOOP MOTOR CONTROL

To close the loop we will calculate the instantaneous speed from the measurement of segment time, and apply the proportional plus integral control equation:

$$F = G_p E + \int G_i E$$

Five subroutines are used. SPEED obtains the interval time and calls DIVID to calculate the instantaneous speed. WIDTH subtracts the instantaneous speed from the desired speed to give the error signal, calls a cumulative multiplication subroutine SCUML to calculate a new error integral, and calls SCUML again to calculate a new pulse width. Finally LDT1 loads Timer 1 with the pulse width.



SUBROUTINE SPEED

FIGURE 7-17

7.3.1 Subroutine SPEED

EXT4 service records the content of Timer 2 when the optical disc generates an interrupt. SPEED (shown in Figure 7-17) repeatedly loads this value (from 83A7,A8) and subtracts it from the previous value. If the result is zero no EXT4 interrupt has occurred, so the reading and subtraction are repeated. After 256 attempts it is assumed that the motor is not moving and SPEED returns with (HL) = 0000.

If successive values of Timer 2 data are different the subtraction of the later value from the earlier gives the time interval, since the timer counts down.

We will obtain the instantaneous speed by division. The speed in revolutions per second is given by:

$$\frac{16 \text{ segments/revolution} \times 2048000 \text{ clocks/second}}{\text{clocks per segment}}$$

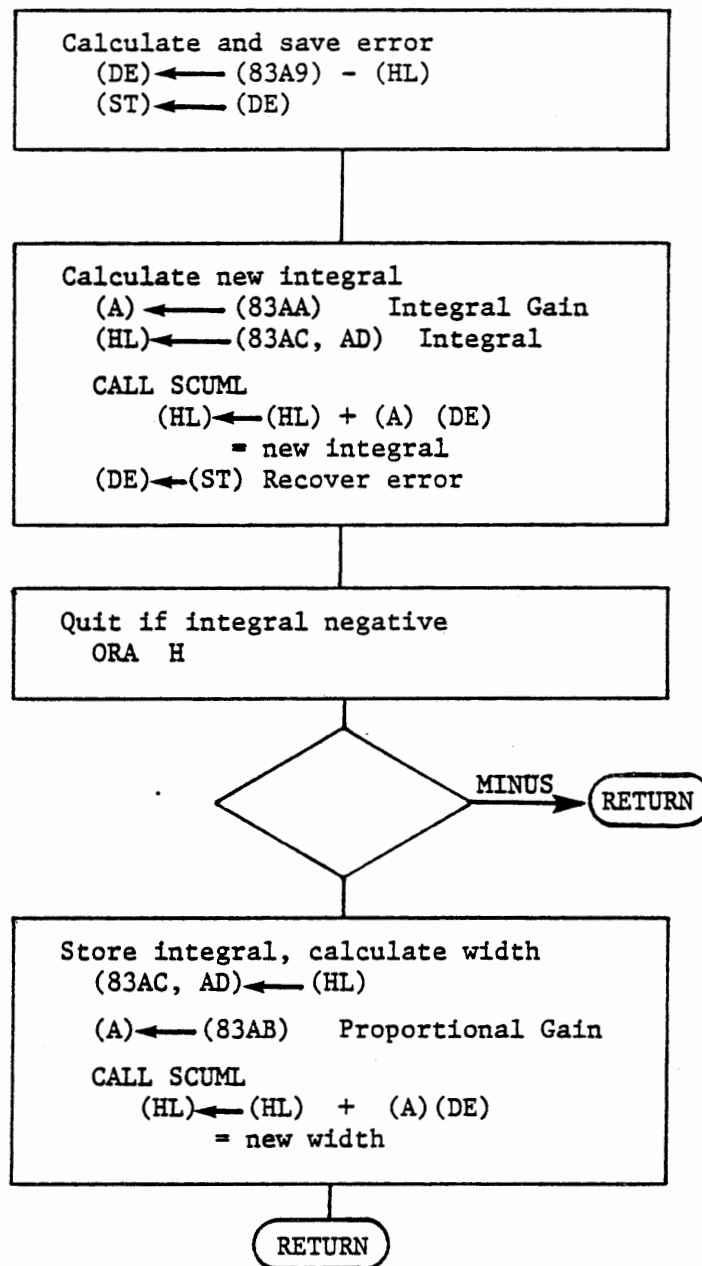
The division subroutine DIVID (Section 7.3.3) is designed for use with left justified floating point numbers, although it does not handle the exponents. It returns a 16 bit result in (HL) with the most significant bit representing the integer part and 15 bits representing a fraction. It can handle numbers that are not left justified provided that the dividend is not greater than twice the divisor.

It turns out that if we enter DIVID with a dividend of 03E8 (in DE) and the segment interval (in HL) as divisor, it returns a quotient (in HL) representing speed as XX.XX. That is, H contains the integer part and L contains the fractional part. The table below shows the relationship between speed and clocks per second. For all speeds that this motor can achieve the number of clocks per segment (the divisor) is more than half of 03E8, so DIVID will return a correct result.

Carry is set before return to indicate that a valid measurement has been made. This is in fact ignored in the present main program, but could be used to invoke full scale control.

Speed and Clocks per Segment

Motor Speed rpm	Segments per Second	Seconds per Segment	System Clocks per Segment	
			Decimal	Binary
2	32	.031250	64000	FA00
5	80	.012500	25600	6400
10	160	.006250	12800	3200
20	320	.003125	64000	1900
50	800	.001250	2560	0A00
100	1600	.000625	1280	0500
150	2400	.000417	853	0355



SUBROUTINE WIDTH

FIGURE 7-18

7.3.2 Subroutine WIDTH

At entry to WIDTH the instantaneous speed is in (HL) as XX.XX . The calculations are limited to single byte values with sign, so we calculate error from the high byte, rounding from the low byte of speed.

```

MOV    A,L      Set CY if low byte
RAL                    Greater than 1/2
LDA    83A9     Desired speed
SBB    H        Subtract speed
MOV    E,A      (E) <--- error
SBB    A        (D) <--- FF if negative
MOV    D,A      (D) <--- 00 if positive

```

The multiplication subroutine SCUML demands that register D contains 00 if (E) is positive, FF if negative. The error is saved in the stack because it is needed for both the integral and proportional calculations, and SCUML destroys the contents of DE. SCUML returns $(HL) = (HL) + (A)*(E)$. To calculate a new integral we load the integral gain to A and the old integral to HL and call SCUML. At return HL contains the new integral and A contains zero. The error is recovered by POP D, and the integral is tested by ORA H, which sets the SIGN BIT if the integral is negative. RM (Return if Minus) after the test avoids storing a negative integral.

Provided that the integral is positive, a new pulse width is calculated by loading A with the proportional gain and calling SCUML

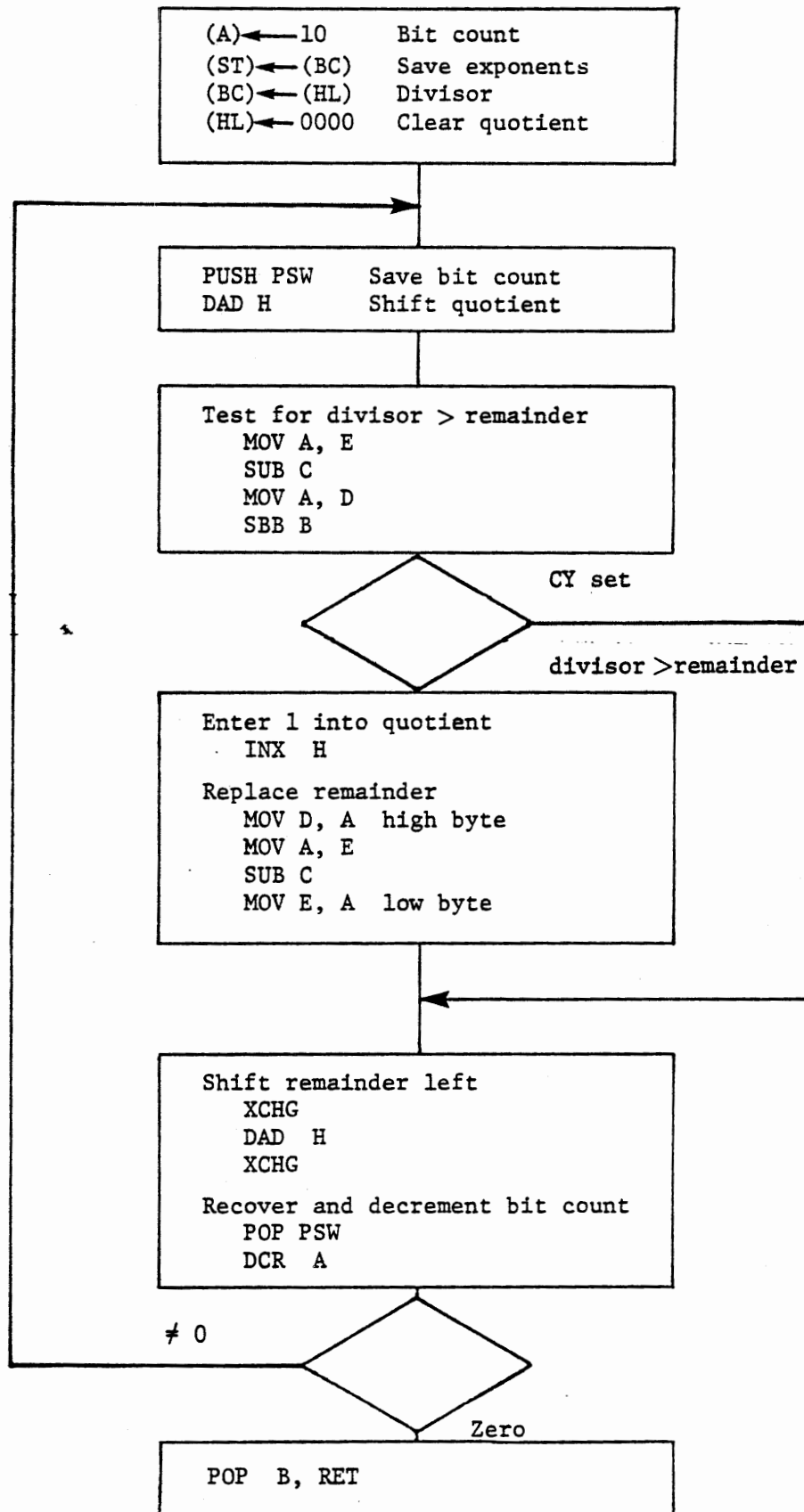
again. At this call we have:

(A) = Proportional gain
(E) = Error
(HL) = Integral

SCUML returns $(HL) = (HL) + (A)*(E)$ which is the new pulse width:

$$F = G_p E + \int G_i E$$

The main loop will call LDT1 to load Timer 1 with the new pulse width.



SUBROUTINE DIVID

Figure 7-19

7.3.3 Subroutine DIVID

Binary division is performed by repeatedly comparing the divisor to the dividend or remainder. If the divisor is less than the dividend or remainder, it is subtracted to form a new remainder and a one is entered into the quotient. Otherwise the old remainder is retained and a zero is entered into the quotient. Now both remainder and quotient are shifted left if more bits are to be processed, and the process is repeated.

Figure 7-19 shows subroutine DIVID. Registers B and C are saved in the stack; in a floating point division program which uses DIVID these registers hold the exponents. Here we do not really need to save them.

Register A is loaded with a bit count; the divisor is copied to (BC) and the quotient (HL) is cleared. The subroutine could be shortened here, because 16 left shifts will clear the old data from (HL). In some fixed point applications, however, a different bit count might be used.

Since all registers are used the bit count is saved in the stack during each loop. The quotient is shifted left at the beginning of the loop rather than at the end because it should not be shifted after the final bit has been processed. The left shift enters a zero into the quotient.

A two byte subtraction sets carry if the divisor is greater than the remainder, in which case the old remainder and the zero bit shifted

THIS PAGE INTENTIONALLY LEFT BLANK

into the quotient are retained. If the subtraction does not generate carry, the low bit of the quotient is made one by INX H, and the remainder is replaced. The high byte of the new remainder is available in A, but the low byte must be generated.

Now the remainder is shifted left and the bit count is recovered and decremented. When the bit count reaches zero the quotient is complete in (HL).

Because this subroutine was developed for floating point arithmetic it expects left justified values--the highest bit of the dividend and of the divisor should be one. This implies that the dividend is less than twice the divisor. The 16 bit quotient represents a single bit to the left of the binary point and 15 bits to the right. The algorithm is valid for numbers which are not left justified only if the dividend is less than twice the divisor. As we have seen, this relationship does hold in the motor control program.

	CY	ZERO	A	B	C	D	E	H	L
KYTIM - Entry Return	x x	x x	x x	x x	Command x	x x	x x	x Keyboard Data except if NEXT	x x
SPEED - Entry Return	x See Notes	x Set	x 00	x Saved	x 00	x x	x x	x Instantaneous Speed	x x
WIDTH - Entry Return	x Clear	x Set	x 00	x Saved	x x	x x	x x	Instant speed Pulse Width	
LDTI - Entry Return	x Clear	x x	x =(H)	x Saved	x x	x x	x x	Pulse Width See Notes	
DECBI - Entry Return	x Clear	x See Notes	x Binary of low byte	x Saved	x x	Keyboard Data 00 Binary of low byte		x Keyboard Data	x x
SMULT - Entry Return	x Clear	x Set	M'plier 00	x Saved	x x	00 or FF M'cand x (See Notes)		x (A)*(E)	x x
SCUML - Entry Return	x Clear	x Set	M'plier 00	x Saved	x x	00 or FF M'cand x (See Notes)		Previous Value (HL) + (A)*(E)	
DIVID - Entry Return	x Saved	x Set	x 00	x Saved	x x	Dividend x (See Notes)		Divisor Quotient	

SUBROUTINE REGISTER USAGE

FIGURE 7-20

7.3.4 Summary of Subroutines

The subroutines used in the motor control program are briefly summarized below. Figure 7-20 shows the register usage for each subroutine. In the figure the data required at entry and returned at exit are listed. An X for entry indicates that the register content does not matter; an X for return indicates that the register content is destroyed.

KYTIM accepts (via ENTWD) optional decimal or hexadecimal data and a command. Data entered are stored in assigned memory locations as described in Section 7.2.5. Valid commands are:

NEXT	-	Set duty cycle (enter in decimal)
STEP	-	Stop motor (optional - set speed)
RUN	-	Start motor (optional - set speed)
CLR	-	Stop motor, clear segment count
MEM	-	Store proportional and integral gains

SPEED calculates instantaneous speed. Obtains interval time from Timer 2 data stored in memory by EXT4 interrupt. Returns carry set except if no interval time is observed after 256 attempts. In that case speed is reported as 0000.

WIDTH calculates and stores integral of error, and returns new pulse width.

LDT1 loads pulse width to Timer 1. If entry value is zero or negative LDT1 replaces entry value with 0001 before loading timer, and returns

SIGN BIT set.

DECBI moves keyboard data from (DE) to (HL) and returns the binary equivalent of the low byte. The zero flag is set if the low byte was zero.

SMULT calculates $(A)*(E)$ and returns the two byte product in HL. At entry register D must contain 00 if (E) is positive or FF if (E) is the twos complement of a negative value.

SCUML calculates $(HL) + (A)*(E)$. It is an alternate entry to SMULT, and omits clearing the product before multiplying.

DIVID calculates $(DE)/(HL)$. The dividend must be no greater than twice the divisor. The quotient is represented as one bit left of the binary point and a 15 bit fraction.

7.3.5 Program Implementation

Since the motor control program essentially fills 512 bytes of memory it is suggested that the given solution be adopted unless your own subroutines can fit in the memory space available to you. There are two memory segments of 16 bytes each left free, at 82B0 and 8390. You may choose to extend the main loop or interrupt service to display the interrupt status byte, which allows observation of the EXT4 input. If you have 1024 bytes of memory you may want to log the instantaneous speed. To do this you will want a subroutine that records only once in ten or sixteen cycles through the main loop, because of the fairly long reaction times of the motor. Section 7.3.6, following the

program solution, discusses some of the results you will see with the given program.

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - INITIALIZE 7- 69

	A	D	D	R	CODE					
CODING SHEET	8	20	0	3E	MVI	A,	80			Program Ports
			1	80						Port 1B Out
			2	D3	OUT	CNT1				
			3	07						
			4	3E	MVI	A,	92			
			5	92						
			6	D3	OUT	CNT2				
			7	0F						
			8	3E	MVI	A,	25			Program Timer 0
			9	25						High byte
MICROCOMPUTER TRAINING SYSTEM		A	D3	OUT	TIMCT					Mode 2
		B	17							Decimal
		C	3E	MVI	A,	50				Load Timer 0
		D	50							for 2.4414 ms
		E	D3	OUT	TIMO					(256 interrupts
		F	14							= 10/16 second)
		8	21	0	3E	MVI	A,	7A		Program Timer 1
				1	7A					Both bytes
				2	D3	OUT	TIMCT			Mode 5
				3	17					Binary
INTEGRATED COMPUTER SYSTEMS			4	3E	MVI	A,	B0			Program Timer 2
			5	B0						Both bytes
			6	D3	OUT	TIMCT				Mode 0
			7	17						Binary
			8	AF	XRA	A				Start Timer 2
			9	D3	OUT	TIM2				
			A	16						
			B	D3	OUT	TIM2				
			C	16						
			D	CD	CALL	CLR				Clear segment
		E	20						count and	
		F	83						stop motor	
	8	22	0	3E	MVI	A,	13		Enable Interrupts	
			1	13					Timer 0, Timer 1.	
			2	D3	OUT	PORT2C			EXT4	
			3	0E						
			4	C3	JMP	8280				
			5	80						
			6	82						
			7							
			8							

Figure 7-21a

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - INTERRUPT SERVICE

	A	D	D	R	CODE															
CODING SHEET	8																			
MICROCOMPUTER TRAINING SYSTEM	8 22	8	F5		PUSH	PSW				RSTS -										
		9	E5		PUSH	H				Timer 0 interrupt										
		A	CD		CALL	STIM0														
		B	4U																	
		C	82																	
		D	C3		JMP	EXIT														
		E	39																	
		F	82																	
MICROCOMPUTER TRAINING SYSTEM	8 23	0	F5		PUSH	PSW				RST6										
		1	E5		PUSH	H				Timer 1 or EXT4										
		2	DB		IN	PORT2B				Read interrupt										
		3	0D							status byte										
		4	E6		ANI	02				Mask for										
		5	02							Timer 1										
		6	CD		CALL	STIM1				Call service routine										
		7	58																	
INTEGRATED COMPUTER SYSTEMS	8 23	9	D3		OUT	CNT2				EXIT - Renewable										
		A	0F							and clear interrupt										
		B	E1		POP	H														
		C	F1		POP	PSW														
		D	FB		EI															
		E	C9		RET															
		F	00																	
		8	0																	

Figure 7-21b

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - SERVICE TIMERS

		A	D	D	R	CODE										
CODING SHEET	8	2	4	0	2	1	L	X	I	H, 83A1	STIM0					
		1			A	1					Address, load					
		2			8	3					and output					
		3			7	E	M	O	V	A, M	motor control byte					
		4			D	3	O	U	T	P	O	R	T	I	C	to set PIC1 low
		5			0	6						if RUN, high if STOP				
		6			3	E	M	V	I	A, 01	To reenable					
		7			0	1					Timer 0 at exit					
		8			2	B	D	C	X	H	Address and					
		9			3	5	D	C	R	M	decrement					
MICROCOMPUTER TRAINING SYSTEM	A				C	0	R	N	Z	time counter						
	B				2	A	L	H	L	D	83A3	At 00 = 5/8 second				
	C				A	3				copy decimal						
	D				8	3				segment count						
	E				2	2	S	H	L	D	83A5	to display area				
	F				A	5										
	8	2	5	0	8	3										
		1			2	1	L	X	I	H, 0000	Clear decimal					
		2			0	0					segment counter					
		3			0	0										
	4			2	2	S	H	L	D	83A3						
	5			A	3											
	6			8	3											
	7			C	9	R	E	T								
INTEGRATED COMPUTER SYSTEMS	8	2	5	8	C	A	J	Z	S	E	X	T	4	STIM1		
		9			6	0					Jump if EXT4					
		A			8	2					Else (A) = 02					
		B			D	3	O	U	T	P	O	R	T	I	C	Turn motor off
		C			0	6										
		D			3	E	M	V	I	A, 03	To reenable Timer 0					
		E			0	3										
		F			C	9	R	E	T							
		8			0											
		1														
	2															
	3															
	4															
	5															
	6															
	7															
	8															

Figure 7-21c

THIS PAGE INTENTIONALLY LEFT BLANK

EXT4 INTERRUPT SERVICE

	A	D	D	R	CODE							
CODING SHEET	8	2	6	0	3E	MVI	A,	80				Latch, read,
					80							and store
					D3	OUT		TIMCT				Timer 2 data
					17							
					DB	IN		TIM2				
					16							
					6F	MOV	L,	A				
					DB	IN		TIM2				
					16							
					67	MOV	H,	A				
MICROCOMPUTER TRAINING SYSTEM	A				22	SHLD		83A7				
	B				A7							
	C				83							v
	D				21	LXI	H,	83A2				Address and
	E				A2							increment binary
	F				83							segment count
	8	2	7	0	34	INR	M					
					23	INX	H					Address and
					7E	MOV	A,	M				increment decimal
					C6	ADI	01					segment count
INTEGRATED COMPUTER SYSTEMS					01							
					27	DAA						
					77	MOV	M,	A				
					23	INX	H					
					7E	MOV	A,	M				
					CE	ACI	00					
	A				00							
	B				27	DAA						
	C				77	MOV	M,	A				v
	D				3E	MVI	A,	09				To reenable and
				09							clear EXT4	
				C9	RET						To exit	
8				0								
				1								
				2								
				3								
				4								
				5								
				6								
				7								
				8								

Figure 7-21d

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - MAIN LOOP

	A	D	D	R	CODE																	
CODING SHEET	8	2	8	0	E5	PUSH	H													Save avg-speed		
				1	DB	IN		PORT	0A												Test keyboard	
				2	00																(FF if no key)	
				3	3C	INR	A															
				4	C4	CNZ		KY	TIM												If key pressed	
				5	00																accept and process	
				6	83																input	
				7	2A	LHLD		83	A1													
				8	A1																(L) ← motor control	
				9	83																(H) ← segment count	
MICROCOMPUTER TRAINING SYSTEM		A			7D	MOV	A	L												Test for motor		
		B			B7	ORA	A													stopped (02)		
		C			C2	JNZ		82	A0											Jump to display		
		D			A0															if stopped		
		E			82																	
		F			CD	CALL		SPEED													(HL) ← instantaneous	
		8	2	9	0	C0															speed	
				1		82																
				2		E5	PUSH	H													(ST) ← speed	
				3		CD	CALL		WIDTH												(HL) ← pulse width	
INTEGRATED COMPUTER SYSTEMS			4		E0																	
			5		82																	
			6		CD	CALL		LDT1													Load Timer 1	
			7		3C																	
			8		83																	
			9		D1	POP	D														(DE) ← Speed	
		A			7A	MOV	A	D													Output speed	
		B			D3	OUT		PORT1B													to DIA	
		C			05																	
		D			00	NOP																
	E			00	NOP																	
	F			00	NOP																	
	8			0																		
			1																			
			2																			
			3																			
			4																			
			5																			
			6																			
			7																			
			8																			

Figure 7-21e

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - MAIN LOOP - DISPLAY

A D D R		CODE					
CODING SHEET	8 2 A 0	7C	MOV	A, H			If running display
		1 CD	CALL	DBYTE			speed or width
		2 95					If stopped display
		3 02					segment count
		4 E1	POP	H			(HL) ← old avg speed
		5 7D	MOV	A, L			
		6 2A	LHLD	83A5			(HL) ← latest
		7 AS					average speed
		8 83					from interrupts
		9 BD	CMP	L			Test for equal
		A C4	CNZ	DWORD			Display average
		B D1					speed if ready
		C 02					
		D C3	JMP	8280			Back to start
		E 80					
	F 82						
MICROCOMPUTER TRAINING SYSTEM	8 2 B 0						82B0 - BF not used
		1					
		2					
		3					
		4					
		5					
		6					
		7					
		8					
		9					
		A					
		B					
		C					
		D					
		E					
	F						
INTEGRATED COMPUTER SYSTEMS	8 0						
		1					
		2					
		3					
		4					
		5					
		6					
		7					
	8						

Figure 7-21f

THIS PAGE INTENTIONALLY LEFT BLANK

SUBROUTINE SPEED

	A	D	D	R	CODE						
CODING SHEET	8	2	C	0	0E	MVI	C, 00			Loop counter	
				1	00						
				2	2A	LHLD	83A7			} (DE) ← Timer 2 Data	
				3	A7						
				4	83						
				5	EB	XCHG					
		8	2	C	6	0D	DCR	C			Count tries and
					7	C8	RZ				exit if not moving
					8	2A	LHLD	83A7			(HL) ← Timer 2 Data
					9	A7					
				A	83						
MICROCOMPUTER TRAINING SYSTEM				B	7B	MOV	A, E			} (HL) ← initial time	
				C	95	SUB	L				- new time
				D	6F	MOV	L, A				= segment interval
				E	7A	MOV	A, D				
				F	9C	SBB	H				
		8	2	D	0	67	MOV	H, A			
					1	B5	ORA	L			Test and loop if
					2	CA	JZ	82C6			time not changed
					3	C6					
					4	82					
				5	11	LXI	D, 03E8			03E8 / Time	
				6	E8					= speed as XX.XX	
				7	03						
				8	CD	CALL	DIVID			(HL) ← (DE) / (HL)	
				9	70					= speed	
INTEGRATED COMPUTER SYSTEMS				A	83						
				B	37	STC				Mark as correct	
				C	C9	RET				result	
				D							
				E							
				F							
		8		0							
				1							
				2							
				3							
			4								
			5								
			6								
			7								
			8								

Figure 7-21g

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - SUBR WIDTH

		A	D	D	R	CODE								
CODING SHEET	8	2	E	0	7	D	MOV	A	-	L	(HL) = Speed.			
					1	7	RAL				MSB of low byte			
					2	3	A	LDA	83	A9	To CY for rounding			
					3	A	9				(A) ← desired speed			
					4	8	3							
					5	9	C	SBB	H					
					6	5	F	MOV	E	→	A	} (DE) ← speed error as two bytes		
					7	9	F	SBB	A					
					8	5	7	MOV	D	→	A			
					9	D	5	PUSH	H	D		Save error		
	A				3	A	LDA	83	A	A	(A) ← G _i			
	B				A	A					(integral gain)			
	C				8	3								
MICROCOMPUTER TRAINING SYSTEM		D			2	A	LHLD	83	A	C	(HL) ← old integral			
		E			A	C								
		F			8	3								
	8	2	F	0	C	D	CALL	S	C	U	M	L	(HL) ← (HL) + (A)(DE)	
					1	6	3						= SG _i E	
					2	8	3							
					3	D	1	POP	D				(DE) ← Error	
					4	B	4	ORA	H				Quit if integral	
					5	F	8	R	M	E	N	U	S	negative
					6	2	2	SHLD	83	A	C		Store integral	
				7	A	C								
				8	8	3								
INTEGRATED COMPUTER SYSTEMS					9	3	A	LDA	83	A	B	(A) ← G _p		
		A			A	B						(proportional gain)		
		B			8	3								
		C			C	D	CALL	S	C	U	M	L	(HL) ← (HL) + (A)(DE)	
					D	6	3						= SG _i E + G _p E	
					E	8	3							
					F	C	9	RET						
		8			0									
					1									
					2									
				3										
				4										
				5										
				6										
				7										
				8										

Figure 7-21h

THIS PAGE INTENTIONALLY LEFT BLANK

MOTOR CONTROL - SUBR KYTIM

A	D	D	R	CODE						
8	30	0	00	NOP						For DI during debug
	1		CD	CALL	ENTWD					(A) ← command
	2		46							(HL) ← keyboard data
	3		03							
	4		EB	XCHG						(DE) ← keyboard data
	5		21	LXI	H, 8305					Dispatch table address - 10
	6		05							
	7		83							
	8		85	ADD	L					Add command
	9		6F	MOV	L, A					
A			6E	MOV	L, M					
B			E5	PUSH	H					(ST) ← dispatch address
C			CD	CALL	DECB I					(HL) ← keyboard data
D			50							(E) ← binary value
E			83							of low byte (D) ← 00
F			7A	MOV	A, D					(A) ← 00
8	31	0	FB	EI						Needed during debug
	1		32	STA	8000					Initialize Log
	2		00							
	3		80							
	4		C9	RET						
	5		30	MEM						Store Gains
	6		14	REG						Undefined
	7		14	ADDR						"
	8		23	STEP						Stop Motor
	9		27	RUN						Start Motor
A			34	NEXT						Set Duty Cycle
B			14	BRK						
C			20	CLR						Clear count, stop
D			00							
E			00							
F			00							
8			0							
	1			ALTERNATE	ENTRIES					
	2			8320	CLEAR	SEGMENT				
	3				COUNT	AND	STOP			
	4			833C	LOAD	TIMER	1			
	5				IF	(HL) > 0				
	6									
	7									
	8									

Figure 7-21i

THIS PAGE INTENTIONALLY LEFT BLANK

KYTIM - CLR, STEP, RUN, MEM, NEXT, LDTI

		A	D	D	R	CODE				
CODING SHEET	8	3	2	0	3	2	STA	83A2		CLR - Clear
				1	A	2				binary segment count
				2	8	3				and stop motor
		8	3	2	3	3	E	MVI	A, 02	STEP - Stop motor
				4	0	2				
				5	D	3	OUT	PORTIC		Turn transistor off
				6	0	6				
		8	3	2	7	3	2	STA	83A1	RUN - Store motor
				8	A	1				control byte
				9	8	3				00=RUN 02=STOP
MICROCOMPUTER TRAINING SYSTEM		A			C	8	RZ			Exit if no data
		B			7	B	MOV	A, E		Store binary
		C			3	2	STA	83A9		equivalent of
				D	A	9				desired speed
				E	8	3				
				F	C	9	RET			
		8	3	3	0	2	2	SHLD	83AA	MEM - Store Gains
				1	A	A				$G_p \leftarrow (H)$
				2	8	3				$G_L \leftarrow (L)$
				3	C	9	RET			
INTEGRATED COMPUTER SYSTEMS		8	3	3	4	3	E	MVI	A, 32	NEXT - Duty cycle
				5	3	2				Multiply binary
				6	C	D	CALL	SMULT		duty cycle (DE)
				7	6	0				x 32 for pulse
				8	8	3				width
				9	2	2	SHLD	83AC		Store as
				A	A	C				integral of error
				B	8	3				
		8	3	3	C	2	B	DCX	H	LDTI - test
				D	7	C	MOV	A, H		pulse width
			E	B	7	ORA	A		for zero or	
			F	2	3	INX	H		negative	
	8		0							
			1							
			2							
			3							
			4							
			5							
			6							
			7							
			8							

Figure 7-21j

THIS PAGE INTENTIONALLY LEFT BLANK

LDT1 CONTINUED

		A	D	D	R	CODE							
CODING SHEET	8	3	4	0	F2	JPLUS	8346	Jump if pulse					
				1	46			width ≥ 0					
				2	83								
				3	21	LXI	H, 0001	Else set					
				4	01			minimum width					
				5	00								
		8	3	4	6	7D	MOV	A, L	Load Timer 1				
					7	D3	OUT	TIM1	with pulse width				
					8	15							
					9	7C	MOV	A, H					
MICROCOMPUTER TRAINING SYSTEM				A	D3	OUT	TIM1						
				B	15								
				C	C9	RET							
				D									
				E									
				F									
		8			0								
					1								
					2								
					3								
					4								
					5								
					6								
					7								
					8								
	INTEGRATED COMPUTER SYSTEMS				A								
				B									
				C									
				D									
				E									
				F									
		8			0								
					1								

Figure 7-21 k

THIS PAGE INTENTIONALLY LEFT BLANK

DECBI

		A	D	D	R	CODE						
CODING SHEET	8	3	5	0		EB	XCHG				(HL) ← input data	
		1				7D	MOV	A, L				
		2				E6	ANI	F0			(A) ← high digit	
		3				F0						
		4				1F	RAR				(A) ← 1/2 high	
		5				5F	MOV	E, A			(E) ← 1/2 high	
		6				1F	RAR				(A) ← 1/4 high	
		7				1F	RAR				(A) ← 1/8 high	
		8				85	ADD	L			(A) ← dec + 1/8	
		9				93	SUB	E			(A) ← dec - 3/8	
MICROCOMPUTER TRAINING SYSTEM	A					5F	MOV	E, A			} = binary	
	B					16	MVI	D, 00				} (DE) ← binary value with (D) = 00
	C					00						
	D					C9	RET					
	E											
	F											
												ENTER WITH
	8	0										(DE) = INPUT DATA, ASSUME
		1										(E) = PACKED DECIMAL
		2										RETURN
	3										(A) = (E) = BINARY VALUE	
	4										(D) = 00	
	5										(HL) = INPUT DATA	
	6										(BC) PRESERVED	
	7										CY CLEAR	
	8										ZERO SET IF VALUE = 00	
	9											
INTEGRATED COMPUTER SYSTEMS	A											
	B											
	C											
	D											
	E											
	F											
	8	0										
		1										
	2											
	3											
	4											
	5											
	6											
	7											
	8											

Figure 7-21 1

THIS PAGE INTENTIONALLY LEFT BLANK

SUBR SMULT, SCUML

		A	D	D	R	CODE					
CODING SHEET	8	3	6	0	2	1	LXI	H, 0000	SMULT		
					1	00			Clear Product		
					2	00					
		8	3	6	3	B	7	ORA	A	SCUML	Clear CY
					4	C	8	RZ		Exit when m'plier=0	
					5	1	F	RAR		(CY) ← LSB of	
					6	D	2	JNC	836A	multiplier.	
					7	6	A			Skip add if LSB=0	
					8	8	3			If LSB=1 add	
					9	1	9	DAD	D	multiplier into	
MICROCOMPUTER TRAINING SYSTEM	8	3	6	A	E	B	XCHG		product.		
					B	2	9	DAD	H	Shift multiplier	
					C	E	B	XCHG			
					D	C	3	JMP	8363	Loop to test	
					E	6	3			multiplier	
					F	8	3				
		8			0						
					1						
					2			ENTER	SMULT OR SCUML		
					3			(A) = MULTIPLIER			
				4			(E) = MULTIPLICAND				
				5			(D) = 00 OR FF (NEGATIVE)				
				6			RETURN				
				7			SMULT	(HL) ← (A)(E)			
				8			SCUML	(HL) ← (HL) + (A)(E)			
INTEGRATED COMPUTER SYSTEMS				9							
				A			(A) ← 00				
				B			ZERO SET				
				C			CY CLEAR				
				D							
				E			(BC) PRESERVED				
				F							
		8			0						
					1						
					2						
				3							
				4							
				5							
				6							
				7							
				8							

Figure 7-21m

THIS PAGE INTENTIONALLY LEFT BLANK

DIVID

$$(HL) \leftarrow (DE) / (HL)$$

	A	D	D	R	CODE						
CODING SHEET	8	3	7	0	3E	MVI	A	10			(A) ← bit count
				1	10						
				2	C5	PUSH	B				Save exponent (B)
				3	44	MOV	B	H			
				4	4D	MOV	C	L			
				5	21	LXI	H	0000			Clear quotient for smaller bit count
				6	00						
MICROCOMPUTER TRAINING SYSTEM	8	3	7	8	F5	DVI	PUSH	PSW			Save bit count
				9	29	DAD	H				Shift quotient
		A			7B	MOV	A	E			Test for
			B		91	SUB	C				divisor > remainder
			C		7A	MOV	A	D			
			D		98	SBB	B				
			E		DA	JC	DV2				Jump if
MICROCOMPUTER TRAINING SYSTEM	8	3	8	0	83						
				1	23	INX	H				Enter 1 into quotient
				2	57	MOV	D	A			(DE) ← new remainder
				3	7B	MOV	A	E			
				4	91	SUB	C				
				5	5F	MOV	E	A			
		8	3	8	6	EB	DV2	XCHG			
INTEGRATED COMPUTER SYSTEMS				7	29	DAD	H				
				8	EB	XCHG					
				9	F1	POP	PSW				(A) ← bit count
		A			3D	DCR	A				Decrement bit
			B		C2	JNZ	DV1				count and loop
			C		78						
			D		83						
INTEGRATED COMPUTER SYSTEMS			E		C1	POP	B				Recover exponent (B)
			F		C9	RET					
	8			0							
				1		ENTER	(DE)	=	DIVIDEND		
				2			(HL)	=	DEVISOR		
				3		DIVIDEND	<		DEVISOR		
				4		RETURN	(HL)	=	QUOTIENT		
				5		AS	.XXXX	-	-		
			6					(DE)	=	REMAINDER	
			7					(BC)		PRESERVED	
			8					(A)	=	0	
								(CY)		PRESERVED	

Figure 7-21n

THIS PAGE INTENTIONALLY LEFT BLANK

7.3.6 Motor Control Program Operation

To operate closed loop control you must enter three data bytes; proportional gain, integral gain, and desired speed. Gains are entered as two consecutive bytes with MEM:

```
0801    MEM    Set proportional gain = 8
          Set integral gain = 1
50     RUN    Set desired speed 50 rps
```

The system allows speed requests from one to 99 rps, but will not operate successfully at speeds much less than 10 rps. Experiment to see the average speed respond to various speed requests. Connect the voltmeter from the D/A output to ground to see an analog indication of instantaneous speed. This together with an oscilloscope display of the pulse width is especially interesting.

Observe the response of average speed as you place a load on the motor. This is most easily done by pushing the end of the motor shaft with a finger. Do not attempt it by dragging on the optical disc, because if you start it slipping it will wear the hole that mounts it on the shaft.

When you apply the load, the motor will slow down but the closed loop control will apply more power to restore the speed. The range of loads over which speed can be maintained is fairly impressive, although unfortunately we have no means of measuring load.

When you release the load the motor will speed up rapidly, and the

control system will hunt for the desired speed just as the voltage control system hunted for a desired voltage. The hunting is easily observed from the sound of the motor. Try different gain values and observe their effect.

If you stall the motor by holding the shaft it will not restart. This is because the integral will increase to a large positive value while the motor is stalled. In following cycles of the main loop a value greater than 7FFF will be calculated for the integral; this is taken to be negative and will not be stored. LDT1 will substitute a minimum pulse width which will not start the motor. You can restart the motor by pressing NEXT, which clears the error integral, allowing closed loop control to function again. SPEED returns to the main loop with carry clear if the motor is stalled. Develop a program modification to enter a 50% duty cycle if the motor is stalled.

The CLR key was defined to stop the motor and clear the binary segment count. This allows observation of the coasting distance after power is removed. Measure the relationship between speed and coasting distance.

7.4 Motor Control by Variable Voltage

We can control the motor by varying the voltage amplitude instead of using pulse width modulation. The output voltage from the power transistor can be controlled by varying the drive to its optical coupler. Remove the connection from MOT CTL+ to +5 volts, and connect ANALOG OUT to MOT CTL+. Now the program can vary the voltage. Only three trivial changes to the program are required:

a) Timer 1 has no function, since power is to be turned on whenever the motor is running. Disable the Timer 1 interrupt by changing the initialization step that originally enabled it.

b) Remove CALL LDT1 from the main loop.

c) Output the high byte of pulse width to the D/A converter instead of the high byte of speed.

These three changes are shown in Figure 7-22.

Now run the program as before.

```

0801    MEM    Set gain
      50    RUN    Request speed

```

Before the motor will start the integral must build up to a much higher value than for pulse width modulation. This is principally because the optical coupler that drives the power transistor must receive a voltage input above about 1.5 volts before it will start to turn the power transistor on. The control will not be as smooth, because the optical coupler makes a much larger change in the motor

voltage. Try different speed requests and see how small a change in the control voltage is required. connect your voltmeter to the motor (MOT DRV) and observe that the very small change in control voltage displayed by the computer makes a much larger change in the motor voltage.

PATCHES FOR VARIABLE VOLTAGE CONTROL

	A	D	D	R	CODE																		
CODING SHEET	8	2	2	0	3E		MVI	A	0	11										Enable EXT4			
			1		11	*															and Timer 0		
			2		D3		OUT		PORT	2	C										interrupts		
			3		0E																	(not Timer 1)	
MICROCOMPUTER TRAINING SYSTEM	8296				00	*	NOP														Delete CALL LDT1		
					00	*	NOP															from main loop	
					00	*	NOP																
					D1		POP	D														(DE) ← instant speed	
					7C	*	MOV	A	0	H												Voltage control	
					D3		OUT		PORT	1	B											("pulse width")	
					05																	to DIA output	
																							(instead of speed)
							*	CHANGES	FROM	PWM													
							PROGRAM	(FIGURE	7-21)														
							FOR	VARIABLE	VOLTAGE														
							MOTOR	CONTROL															
							CONNECT	MOT	CTL+														
							TO	ANALOG	OUT														
							INSTEAD	OF	+5V														

INTEGRATED COMPUTER SYSTEMS

Figure 7-22

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A

This Appendix contains the following Figures for reference:

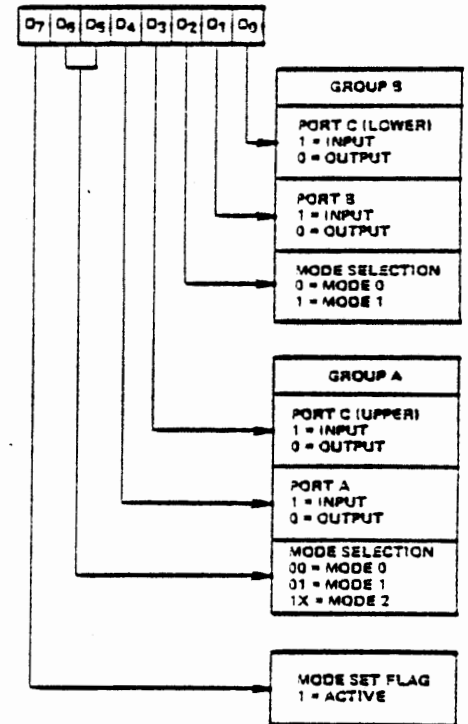
- Figure A-1 (same as 2-2)
- A-2 (same as 2-13)
- A-3 (same as 2-15)
- A-4 (same as 3-3)
- A-5 (same as 3-4)
- A-6 (same as 3-24)
- A-7 (8253 Table)

PORT ADDRESSES AND ASSIGNMENTS

ADDRESS	PORT NAME	FUNCTION	SPECIAL ASSIGNMENTS FOR 0C AND 1C		
00	PORT 0A	MTS Keyboard Input	0C7	Display Control	(1 = On)
01	PORT 0B	Unassigned except 0B0	0C6	Enable Command Keys	(0 = On)
02	PORT 0C	See column at right	0C5	Enable Keys 8-F	(0 = On)
03	CNT 0	Control Port for MTS 8255	0C4	Enable Keys 0-7	(0 = On)
04	PORT 1A	LED and Driver Outputs	0C3	Unassigned	
05	PORT 1B	D/A Output or A/D Input	0C2	Unassigned	
06	PORT 1C	See column at right	0C1	Unassigned	
07	CNT 1	Control Port for 8255 # 1	0C0	Cassette Modem Out	
0C	PORT 2A	Unassigned	0B0	Cassette Modem In	
0D	PORT 2B	Interrupt Status Input	1C7-4	Unassigned	
0E	PORT 2C	Interrupt Enable Output	1C3	Interrupt (If Enabled by 2C6)	
0F	CNT 2	Control Port for 8255 # 2	1C2	Unassigned	
14	TIM 0	Timer 0	1C1	Motor Drive Buffer	(1 = On)
15	TIM 1	Timer 1	1C0	D/A Control (1 = Automatic A/D)	
16	TIM 2	Timer 2			
17	TIM CT	Control Port for 8253			

8255 PROGRAMMING CONTROL BYTES (WRITE TO 8255 CONTROL PORT)

CONTROL BYTE	PORT A	PORT B	PORT C0-C3	PORT C4-C7	USE WITH 8255 #		
					0	1	2
80	Out	Out	Out	Out		D/A	
81	Out	Out	In	Out	●		
82	Out	In	Out	Out		A/D	*
83	Out	In	In	Out		A/D	
88	Out	Out	Out	In		D/A	
89	Out	Out	In	In	●		
8A	Out	In	Out	In		A/D	
8B	Out	In	In	In		A/D	
90	In	Out	Out	Out	*	D/A	
91	In	Out	In	Out	*	●	
92	In	In	Out	Out	*	A/D	*
93	In	In	In	Out	*	A/D	
98	In	Out	Out	In		D/A	
99	In	Out	In	In	●		
9A	In	In	Out	In		A/D	
9B	In	In	In	In		A/D	



* Generally only these control bytes should be used for normal operation.
 ● Forbidden configurations

Mode Definition Format

Figure A-1

STATUS AND COMMAND BYTES

INTERRUPT SOURCE	STATUS BYTE OBTAINED BY IN PORT 2B (see Note 2)								COMMAND BYTE WRITTEN BY OUT CNT 2 (see Note 1)		
	BINARY								HEX	DISABLE	ENABLE
Timer 0	0	X	X	X	X	X	X	1	01	00	01
Timer 1	0	X	X	X	X	X	1	X	02	02	03
Timer 2	0	X	X	X	X	1	X	X	04	04	05
A/D Comparator	0	X	X	X	1	X	X	X	08	06	07
EXT 4	0	X	X	1	X	X	X	X	10	08	09
EXT 5	0	X	1	X	X	X	X	X	20	0A	0B
Port 1C3	(see Note 3)									0C	0D

Note 1: Disable or enable command byte must be output to CNT 2 to clear the interrupt flip flop for Timer 0, Timer 1, EXT 4, or EXT 5. Disable or enable for A/D Comparator clears the interrupt in automatic A/D mode only.

Note 2: The hex values shown assume all other bits are 0. ANI (hex value) will give zero if the interrupt is not present.

Note 3: Port 1C3 does not appear in the status byte. It is read as XXXX1XXX by IN PORT1C. It is cleared by reading PORT1A in strobed input mode (mode 1 or mode 2) or by writing to PORT1A in strobed output mode (mode 1 or mode 2). Otherwise it can be cleared or set by writing 06 or 07 to CNT1. The interrupt enable for Port 1C3 is cleared or set by writing 0C or 0D to CNT2, but this does not change the data at Port 1C3.

STATUS AND COMMAND BYTES

Figure A-2

Program 8255's - 1B out

3E	MVI	A,80
80		
D3	OUT	CNT1
07		
3E	MVI	A,92
92		
D3	OUT	CNT2
0F		

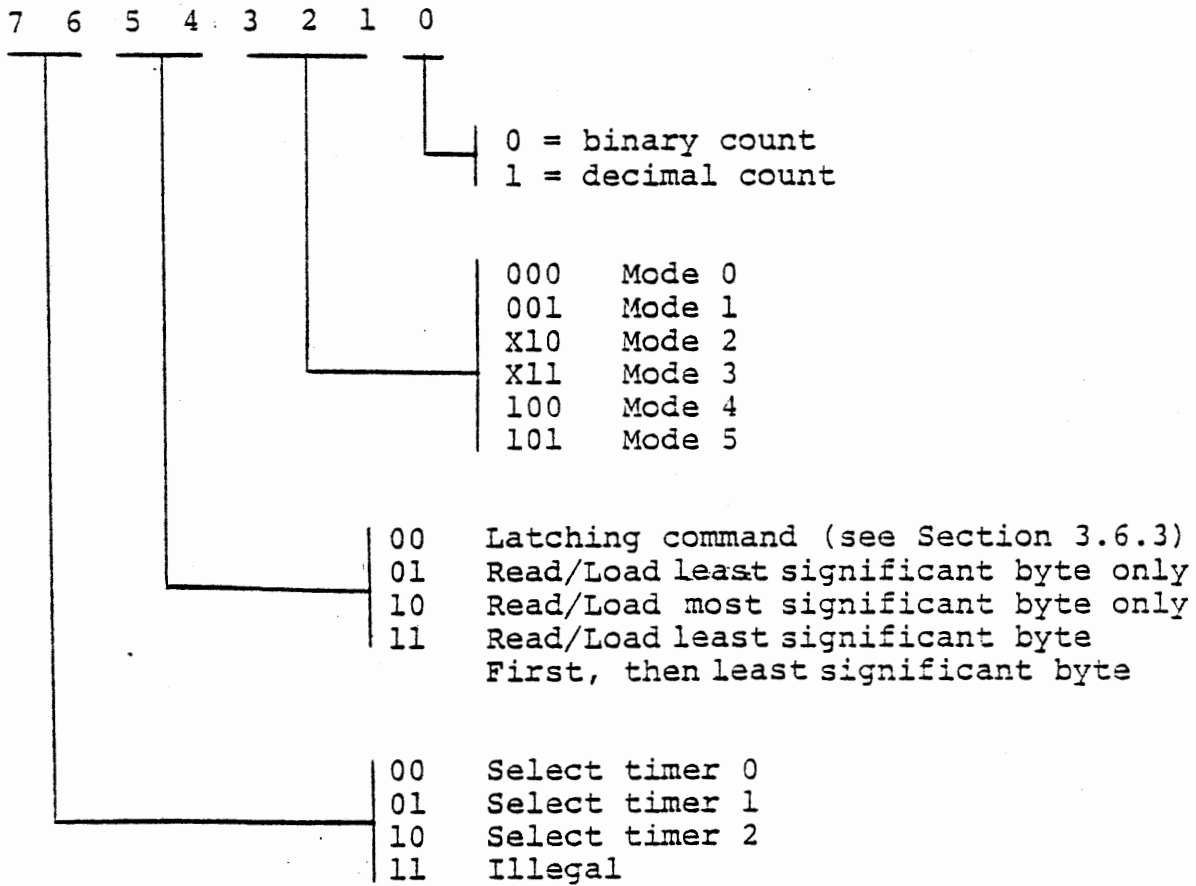
Program 8255's - 1B in

3E	MVI	A,82
82		
D3	OUT	CNT1
07		
3E	MVI	A,92
92		
D3	OUT	CNT2
0F		

STANDARD PROGRAMMING FOR 8255'S

Figure A-3

Timer Control Byte Structure



TIMER CONTROL BYTE STRUCTURE

Figure A-4

Timer 0	Mode					
	0	1	2	3	4	5
Latch	00	00	00	00	00	00
Read/Load LSB	10	12	14	16	18	1A
Read/Load MSB	20	22	24	26	28	2A
Read/Load Both (LSB first)	30	32	34	36	38	3A

Timer 1	Mode					
	0	1	2	3	4	5
Latch	40	40	40	40	40	40
Read/Load LSB	50	52	54	56	58	5A
Read/Load MSB	60	62	64	66	68	6A
Read/Load Both (LSB first)	70	72	74	76	78	7A

Timer 2	Mode					
	0	1	2	3	4	5
Latch	80	80	80	80	80	80
Read/Load LSB	90	92	94	96	98	9A
Read/Load MSB	A0	A2	A4	A6	A8	AA
Read/Load Both (LSB first)	B0	B2	B4	B6	B8	BA

Control Bytes shown set binary counting

Add 1 for decimal counting

Write control byte to TIMCT, Port 17

Latching control byte does not affect mode

TIMER CONTROL BYTES

FIGURE A-5

Mode	Output after mode set	Starts counting	Output goes low	Output goes high	Count restarted	Comments
0 Interrupt	Low	When final byte loaded	At mode set	At zero	By reloading	Output is set low by setting mode or by reloading.
1 One Shot	High	After gate rising edge	After gate rising edge	At zero	By gate rising edge	Can be preloaded during counting. Present period not affected.
2 Rate Generator	High	When final byte loaded	At count=1	At zero	At zero or by gate rising edge	New value effective for next period.
3 Square Wave	High	When final byte loaded	At $n/2$ or $(n + 1)/2$	At zero	At zero or by gate rising edge	If loaded while counting new period is effective for next half of total period.
4 Software Strobe	High	When final byte loaded	At zero	At next clock after zero	By reloading	
5 Hardware Strobe	High	After gate rising edge	At zero	At next clock after zero	By gate rising edge	If loaded while counting new period is effective after next gate rising edge.

8253 TIMER MODES

Figure A-6.

8253 Table

Binary Count (Hex representation)	Decimal Count	Time (milliseconds)
0100	256	0.125
0200	512	0.250
0400	1024	0.500
0800	2048	1.000
1000	4096	2
1800	6144	3
2000	8192	4
2800	(10240)	5
3000	(12288)	6
3800	(14336)	7
4000	(16384)	8
4800	(18432)	9
5000	(20480)	10
A000	(40960)	20
F000	(61440)	30
0000	(65536)	32
		Time (seconds)
1F40	8000	1/256
0FA0	4000	1/512
07D0	2000	1/1024

FIGURE A-7

APPENDIX B

MTS - INTERFACE BOARD MODIFICATIONS

MTS - Interface Board Modifications

In order to connect the Experiment Board (ITS) to the unmodified MTS board it is necessary to bring some additional signals from the MTS to the 100 Pin edge connector, P1. If you have the unmodified MTS board (part number 1052501-1) you may make these changes yourself. If you have the modified MTS (part number 1052501-1D) these changes are unnecessary.

a second modification is necessary to fix a potential problem. If your board is designated REV.E, this second modification is NOT necessary.

To modify the MTS, the following tools are required:

- 1) Low wattage soldering iron (20-40 watts)
- 2) Multi-core solder (Sn 60)
- 3) Wire strippers
- 4) Wire cutters
- 5) X-acto knife or razor blade

The following parts are supplied with your modification kit:

- 1) 1 - 2n2222 transistor
- 2) 1 - 1K 1/4 watt resistor (Brown, Black, Red)
- 3) 1 - 10K 1/4 watt resistor (Brown, Black, Orange)
- 4) 1 - 1n4148 Diode
- 5) 3 - 4 feet of jumper wire
- 6) Heat shrink tubing
- 7) 3 - 8.2 1/4 watt resistors (Grey, Red, Red)

THIS PAGE INTENTIONALLY LEFT BLANK.

Flip the MTS board so that the solder side is face up and the keyboard mounting assembly is to your left. In Figure B-1, P1 is the 100 Pin edge connector. The "A" side corresponds to the pins on the component side of the board and is the set of plated through holes closest to the edge connector. The "B" side corresponds to the pins on the solder side of the board and is the second set of through holes from the edge connector pins.

The signals to be tapped (in "Board Designation" column of table B-1) are marked on the component side of the MTS board. Follow the step-by-step procedure as outlined in table B-1 and depicted in Figure B-2a.

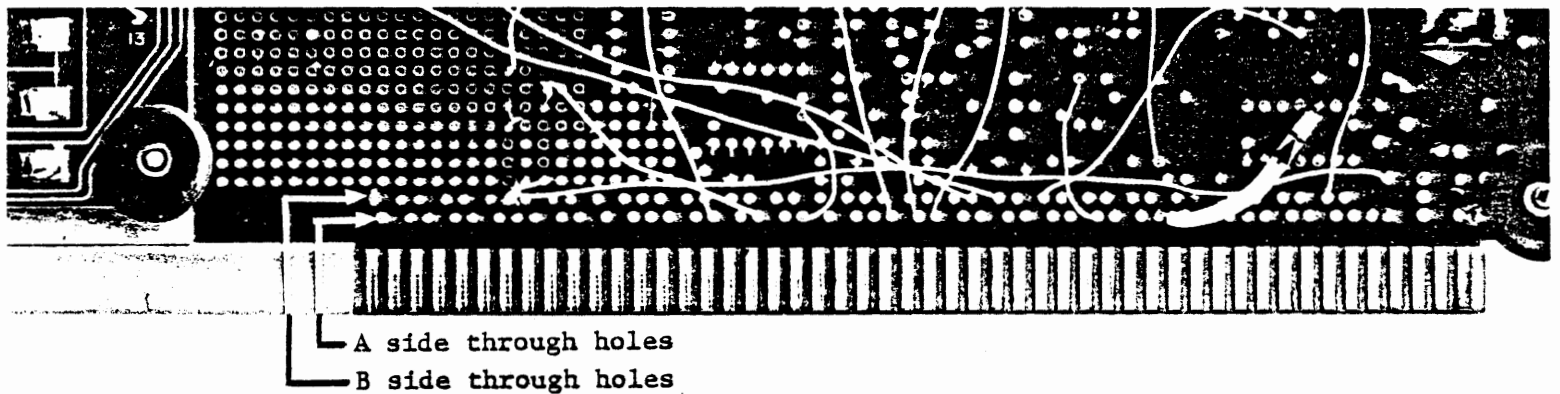
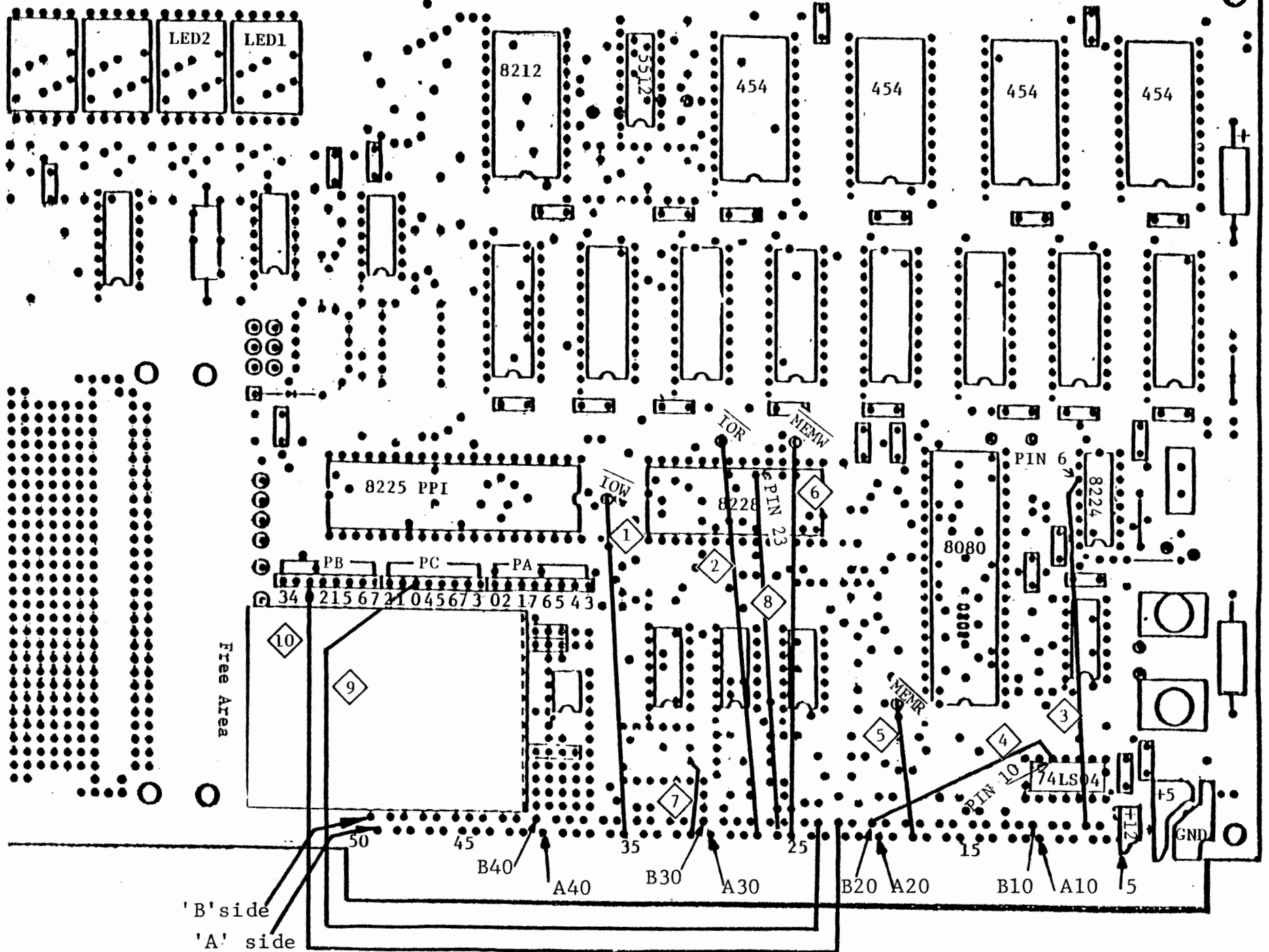


Figure B-1

P1 100 Pin Edge Connector

Figure B-2a

Steps 1-10 of MTS Modification



c/7/78

STEP	CONNECTION		LENGTH OF JUMPER WIRE (INCHES)
	From Board Designation	To Edge Connector Designation Side/Pin	
1	\overline{IOW}	A35	3
2	\overline{IOR}	A27	$3\frac{1}{4}$
3	\emptyset_2 (TTL)	B7	3
4	\overline{RESET}^1	B20	2
5	\overline{MEMR}	A18	$1\frac{1}{4}$
6	\overline{MEMW}	A25	$3\frac{1}{2}$
7	\overline{INTR}	A31	$\frac{3}{4}$
8	\overline{INTA}	A26	3
9	$PC\emptyset^2$	B23	4
10	$PB\emptyset^3$	B22	5

- NOTES: 1 \overline{RESET} Signal can be tapped from U1 (74LS04)
 2 $PC\emptyset$ is 8255 port C bit \emptyset
 3 $PB\emptyset$ is 8255 port B bit \emptyset

Table B-1
Steps 1-10 of MTS Modification

Steps 11 through 15 of MTS Board Modifications

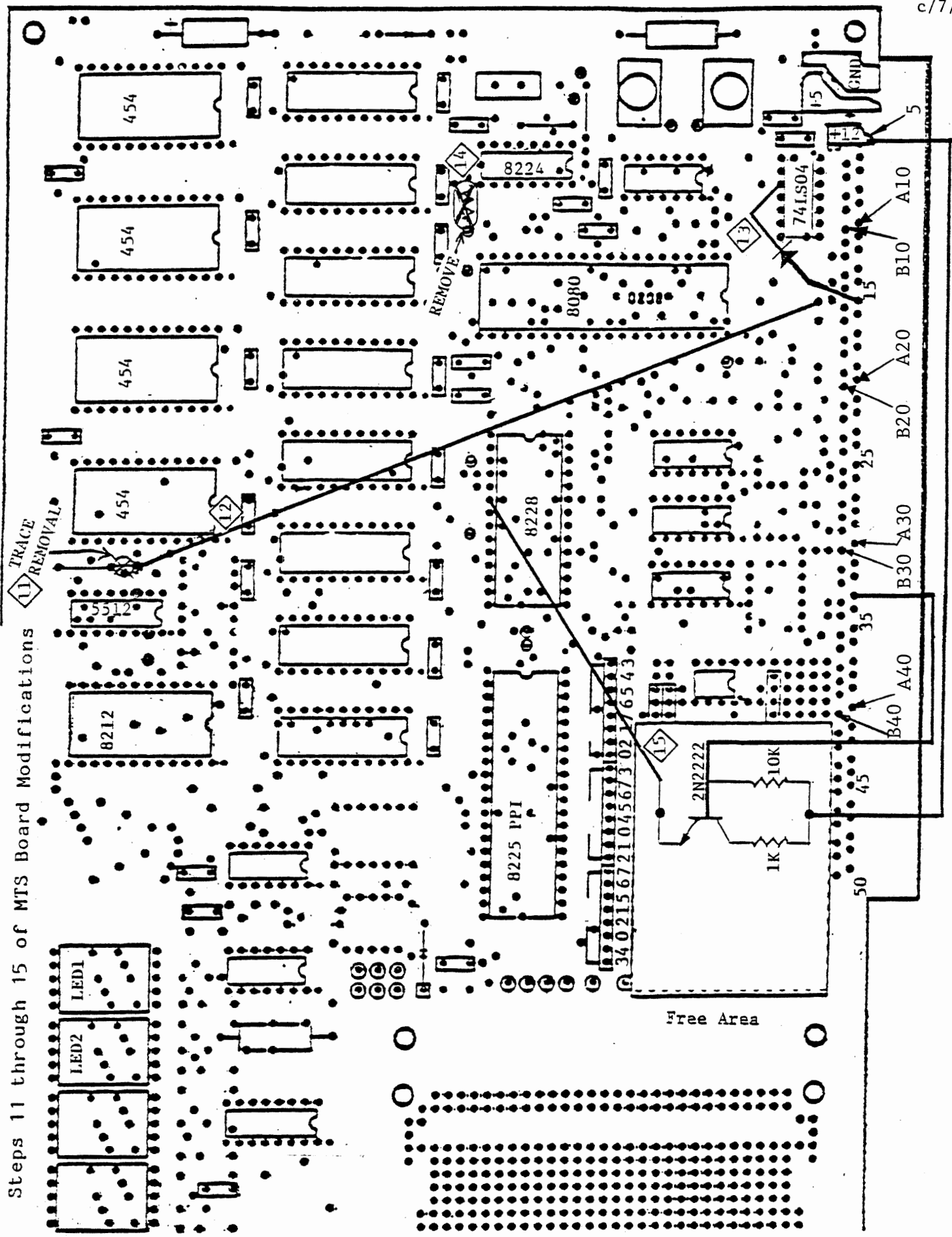


Figure B-2b

Refer to Figure B-2b for the following steps.

11 Trace Removal

To allow for decoding of more memory, remove the trace between U8 (74155 chip) pin 14 and ground. A section of the printed circuit must be cut and removed between two solder points, as marked in Figure B-3. Using the X-acto knife or razor blade cut the trace as shown. Insert the knife edge below the trace and pry up the edge from the fiber board backing. Now peel the line of plating away from the board up to the solder point and cut the trace from the board.

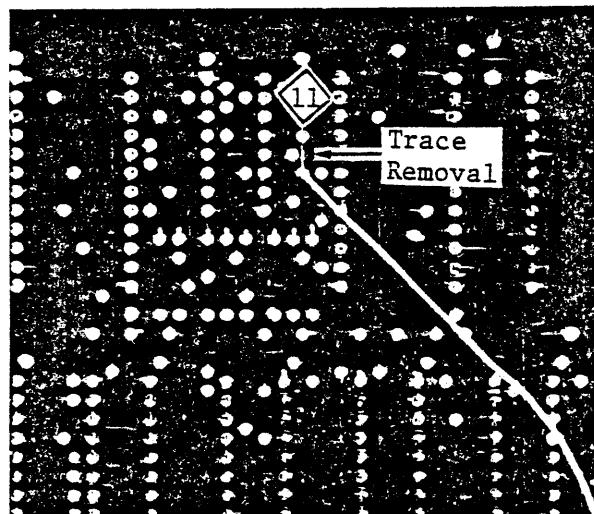


FIGURE B-3
Trace Removal

12 U8 Pin 14 to Address Bus 10

Now connect U8 pin 14 to Address Bus 10 located on the 100 pin edge connector as side "A" hole 15 (P1-A15).

13 Diode Installation

Install the 1n4148 diode from U1 (74LS04) pin 12 to P1-A15. Make sure

THIS PAGE INTENTIONALLY LEFT BLANK

that the cathode side is towards U1 pin 12. The cathode side of the diode is marked with the dark band. Refer to Figures B-4a and B-4b for proper diode orientation. Insulate both ends of the diode with the heat shrink tubing. CAUTION: the tubing will shrink anytime heat is applied.

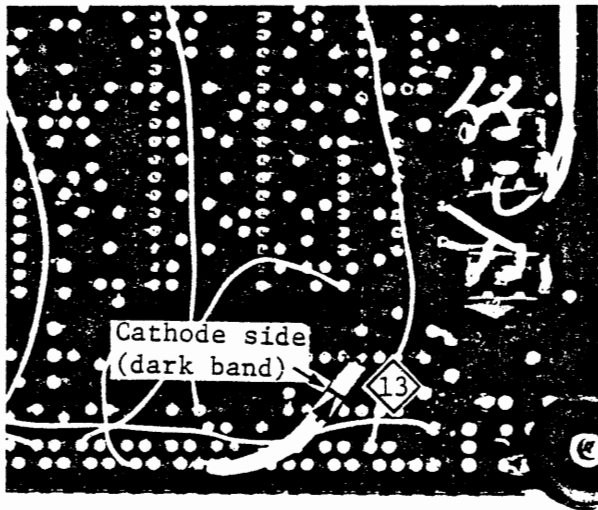


FIGURE B-4a

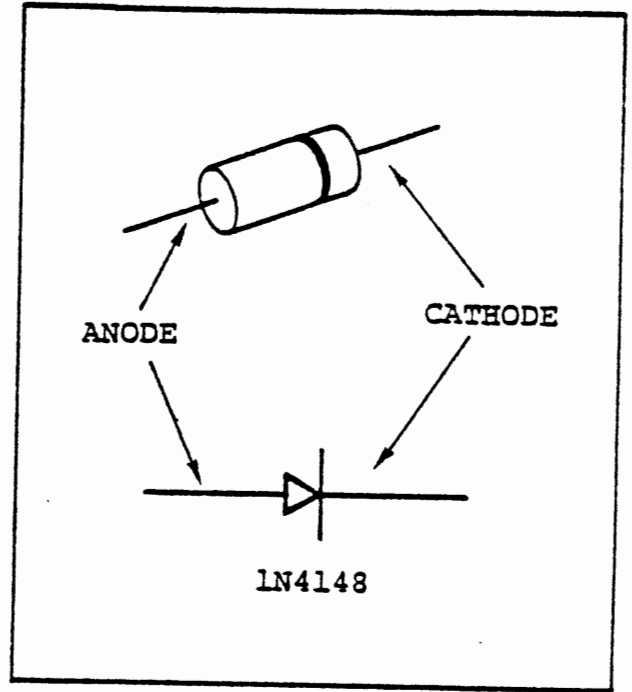


FIGURE B-4b

Diode Orientation

14 Remove Resistor R62

Remove resistor R62 from the component side of the board as shown in Figure B-5. To remove R62, grasp the resistor from the component side of the board and heat each of the solder points until that end of the resistor is easily extracted. If the resistor binds, check the solder side of the board to see if the resistor leads are crimped. Straighten the leads if necessary for extraction.

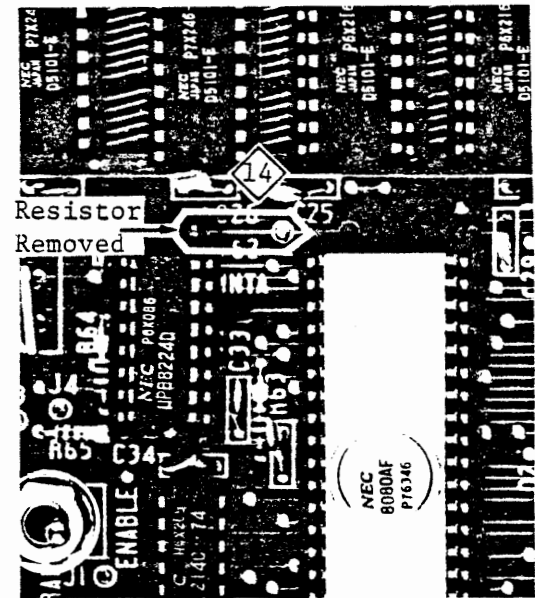
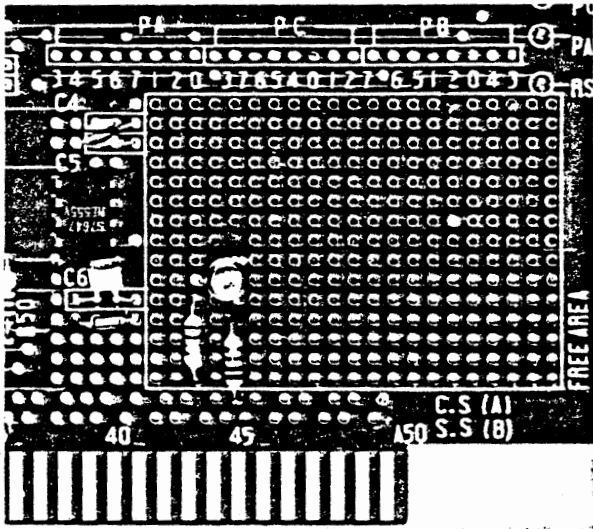


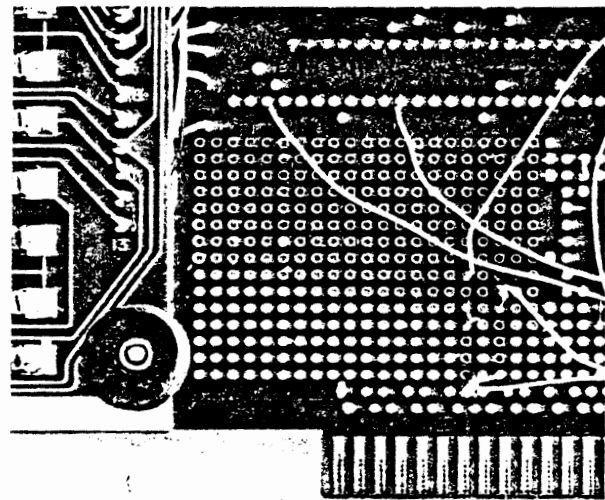
Figure B-5
R62 Resistor Removal

15 Transistor Circuit Installation

Install the transistor circuit shown in Figure B-2b in the Free Area of the MTS board. A suggested parts layout is shown in Figures B-6 and B-7. NOTE: The transistor and resistors are mounted on the component side of the board and the solder connections are made on the solder side of the board.



Suggested Parts Layout
Component Side



Suggested Parts Layout
Solder Side

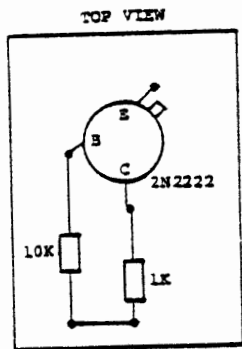


Figure B-6
Transistor Circuit
Component Side

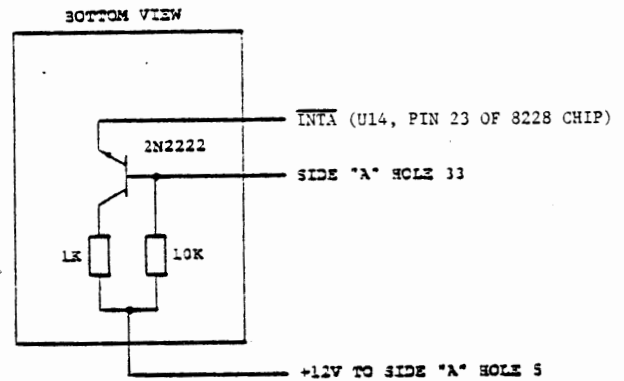


Figure B-7
Transistor Circuit
Solder Side

The second modification, to update from Revision D to Revision E, merely necessitates the replacement of the three resistors marked R46, R47, and R48 on the component side of the MTS board. These resistors are replaced by the three 8.2K $\frac{1}{4}$ watt resistors (Gray, Red, Red) included in the MTS/ITS Modification Kit. Refer to Figure B-8 for the following steps.

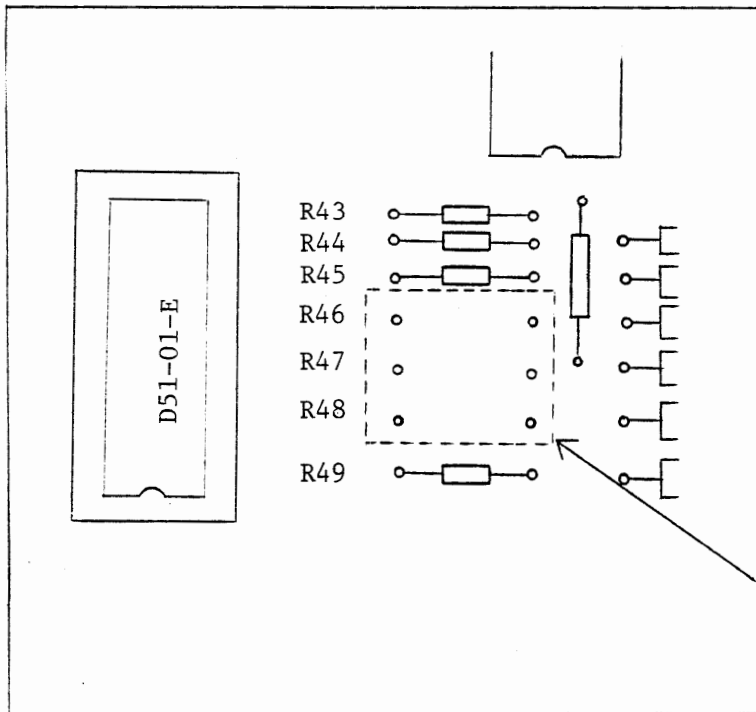


FIGURE B-8

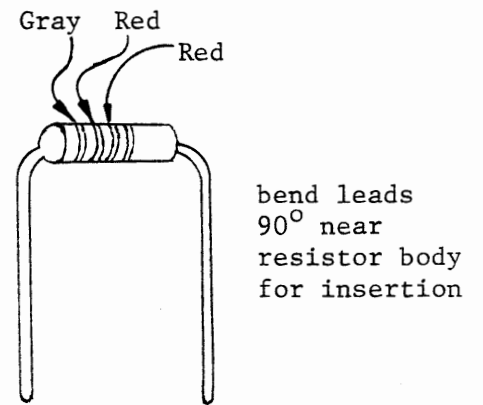


FIGURE B-9

a/7/78

- 16 As in 14 remove resistor R46
- 17 Remove resistor R47
- 18 Remove resistor R48
- 19 For each of the three $\frac{1}{4}$ watt 8.2 K OHM resistors, bend both of the resistors' leads, near the body of the resistor, 90° (see Figure B-9).
- 20 Insert an 8.2K resistor through the pair of R46 through holes, and solder both leads. Insert the resistor such that the gray band is towards the R46 designation (for cosmetic reasons).
- 21 Insert a second 8.2K resistor through R47's through holes, and solder both leads.
- 22 Insert the third 8.2K resistor through R48's through holes, and solder both leads.
- 23 Clip resistor leads on solder side of board.
- 24 Test board via test procedures given in Instructions.

APPENDIX C

CABLE AND EDGE CONNECTOR DESIGNATIONS

A=Component side
B=Solder Side

MTS

1053602-2

11/14/77
MEA

c/7/78

ITS 50 PIN CONNECTOR	SIGNAL	MTS 100 PIN EDGE CONNECTOR
1	GND	A1
2	GND	B1
3	GND	A2
4	GND	B2
5	+5	A3
6	+5	B3
7	GND	not used
8	+12	A5
9	+12	B5
10	GND	not used
11	GND	B6
12	Ø2 (TTL)	B7
13	GND	not used
14	AB15	A10
15	AB7	B10
16	AB6	B11
17	AB5	B12
18	AB4	B13
19	AB3	B14
20	AB10	A15
21	AB2	B15
22	AB9	A16
23	AB1	B16
24	AB8	A17
25	AB0	B17
26	<u>MEMR</u>	A18
27	RESET	B20
28	PBO	B22
29	PCO	B23
30	<u>MEMW</u>	A25
31	INTA	A26
32	<u>DB7</u>	B26
33	IOR	A27
34	DB6	B27
35	DB5	B28
36	DB4	B29
37	<u>DB3</u>	B30
38	INTR	A31
39	DB2	B31
40	DB1	B32
41	INTC	A33
42	<u>DB0</u>	B33
43	IOW	A35
44-50	not used	A4, A6, A7, A8, A9, A11, A12, A13, A14, A19, A20, A21, A22, A23, A24, A28, A29, A30, A32, A34, A36-A50, B4, B8, B9, B18 B19, B21, B24, B25

APPENDIX D
CASSETTE INTERFACE INSTRUCTIONS
AND
PROGRAM CASSETTE LIBRARY

I M P O R T A N T

THE TAPE PROVIDED IS A MASTER TAPE. IT SHOULD BE COPIED TO A WORKING TAPE WHICH SHOULD BE USED FOR LOADING OF PROGRAMS. KEEP THE MASTER TAPE IN A SAFE PLACE SO THAT IF YOUR WORKING TAPE BECOMES DAMAGED OR WORN OUT, YOU CAN MAKE ANOTHER COPY. THE WORKING TAPE SHOULD BE A HIGH-QUALITY C-60 OR C-30 CASSETTE.

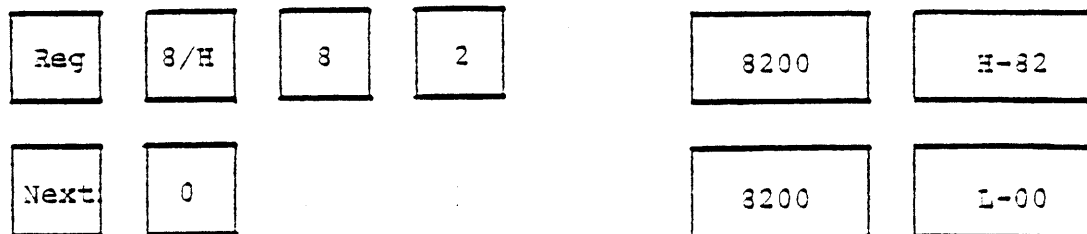
CASSETTE INTERFACE INSTRUCTIONS

The MTS monitor program contains the software routines for loading and storing binary data using a tape cassette unit. The input routine, SERIN, resides at location 03B1 and the output routine, SEROT, resides at 0375.

A cable has been provided with your ITS board for connection between the tape unit and the interface circuitry on the board. One end of the cable has a plug for connection to either the MIC or EAR socket on the tape unit. The other end of the cable has two alligator clips. The black (or ground shield) clip should be connected to CASSETTE GND, screw terminal #6. The red (or center wire) clip should be connected to CASSETTE EAR, screw terminal #7, when reading from tape or to CASSETTE AUX, screw terminal #5, when writing to tape.

D.1.1 INSTRUCTIONS FOR READING FROM TAPE

- 1) Find the program (or data) on the tape and listen for the solid tone which PRECEDES the program.
- 2) Connect the cable to EAR on the tape unit and CASSETTE EAR (screw terminal #7) on the ITS board.
- 3) Enter the beginning load address into Reg Pair H.
E.g., if the beginning address is 8200:



- 4) Enter the start address of SERIN into the Program

Counter:

Addr	0	3	B	1
------	---	---	---	---

- 5) With about 80% of full volume, press the PLAY button on the tape unit. Wait until the OUT LED emits a constant glow.
- 6) Depress **Run**
The MTS displays will blank out and the OUT LED will flicker.
- 7) A display of **03EF** in the left MTS display indicates a successful load.
- 8) A display of **Err** indicates an unsuccessful load, so repeat the procedure.
- 9) Inspect RAM by comparing it to the program listing to verify a proper load: Press **RST** **NEXT** **NEXT**...

D.1.2

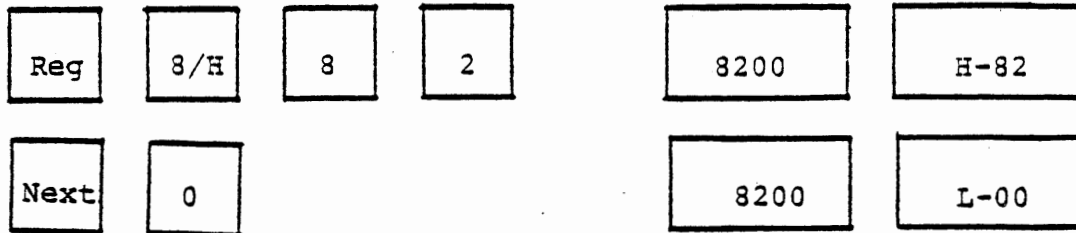
INSTRUCTIONS FOR WRITING TO TAPE

- 1) Advance the tape to the point where you want the program (data) stored. If using the beginning of the tape, make sure you have advanced the tape beyond the leader.
- 2) Connect the cable to MIC or AUX on the tape unit and connect the red clip to CASSETTE AUX (screw terminal #5) on the ITS.
- 3) Enter the number of bytes (in hex) to be stored in Reg Pair D. E.g., if 1A5 (hex) bytes are to be stored:

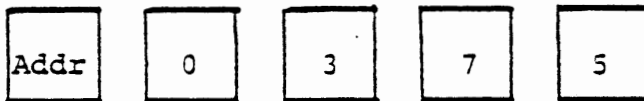
Reg	D	1	3200	D-01
Next	A	5	3200	E-A5

c/7/78

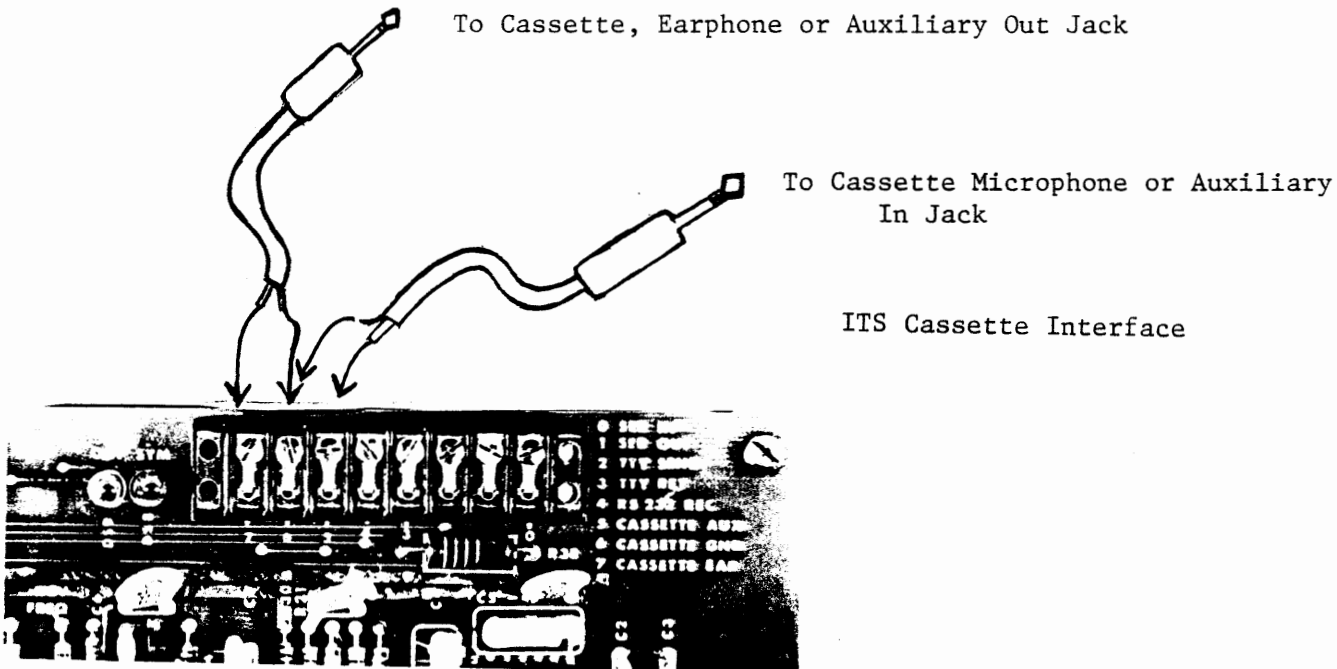
- 4) Enter the beginning address of the program (data) in memory into Reg Pair H. E.g., if the program starts at 8200:



- 5) Enter the start address of SEROT into the program counter:



- 6) Press the RECORD and PLAY buttons on the tape unit simultaneously. Wait 5-10 seconds.
- 7) Depress **Run**. The MTS displays will blank out.
- 8) A display of **03EF** indicates a successful transfer.
- 9) A display of **Err** indicates an unsuccessful transfer, so repeat the procedure.



COURSE 536
PROGRAM CASSETTE LIBRARY
TAPE DIRECTORY

D-4
c/7/78

Program Number	Count*		Program Name	Reference Page
	Relative	Actual		
			MTS/ITS Test Routine	xiii
1			Control Demonstration	D.2.1
2			Pong	D.2.2
3			RS-232 Bit-Banging Routine	E
4			Programming the 8255's	2-9
5			Clearing Interrupt Flip-Flops	2-28
6			EXT 4&5 Service Prog. & Subr.	2-42
7			Short EXT4&5 Service Subr.	2-45
8			Compare Timing Loop	3-18
9			Getky Using Interval Timer	3-25
10			Pulse Width Measurement	3-33
11			Time-of-day	3-48
12			Revised Time Clock	3-59
13			Pulse Width Modulation	4-20
14			Tune(w/"Home on the Range")	4-44
15			Data Table for "The Drunken Sailor"	4-53
16			Patch to Display Tone	4-60

*The count shown is relative and may not correspond to the counter on your tape unit. You should find each program on the tape and fill in the actual count for each from your unit.

FIGURE D-1

D.2 PROGRAM CASSETTE LIBRARY

Your program cassette library is a cassette tape that contains some diagnostic/test, demo and game programs as well as solutions for the longer 536 course exercises. The cassette is included only to free the student from the time-consuming key-in procedure. It is recommended that you use the cassette only after you have attempted your own solutions to the exercises. Furthermore, you should verify a correct load by comparing memory to the accompanying listings.

The listings of the programs are available in the appropriate course section. Those programs that are not exercise solutions have source code listings and instructions on the following pages.

A list of library programs is shown in Figure D-1.

CONTROL DEMONSTRATION PROGRAM

Description:

D.2.1 This program demonstrates the ability of the microcomputer to control the following types of functions:

- Record the running time of a motor and set an audible and visual alarm when the running time reaches a preset value.
- Monitor a temperature and set an alarm if it exceeds a preset limit.
- Monitor two switches and set an alarm if the switches are actuated.
- Display time-of-day when no other display has been requested and no alarm has been set.

Figure 1 describes the functions associated with each of the command keys as used in this program.

KEY	FUNCTION	OPTION
NEXT	Display time-of-day	
STEP	Stop the motor	
RUN	Start the motor	Set speed
ADDR	Display motor running time	Set alarm limit
REG	Display temperature	Set alarm limit
MEM	Display time-of-day	Set time-of-day
BRK	Display time-of-day	
CLR	Clear alarm	

FIGURE D-2

Each of the three motor control functions displays the motor running time. The alarm limit is set by keying in hours and minutes before pressing ADDR. The alarm will be set when the cumulative motor running time reaches the alarm limit.

The temperature is displayed only in response to REG. An alarm limit may be entered in degrees centigrade before pressing REG. The alarm is set when the temperature increases above the alarm limit.

All other commands display time of day, unless the alarm is on. The time may be set by keying in hours and minutes followed by MEM.

An alarm is set if any of the following occurs:

EXT 4 input is switched to ground. The display will show E4 and LED 4 (fifth from the right) will be turned on - CLR turns off the alarm.

EXT 5 input is switched to ground. The display will show E5, and LED 5 (sixth from the right) will be turned on. CLR turns off the alarm.

Temperature increases above an alarm limit that has been entered. The initial limit exceeds full scale so no alarm will occur unless a value has been entered using the REG key. The display will show the alarm limit at the left and the temperature at the right. LED 3 (fourth from the right) will be on. Pressing CLEAR or REG will clear the alarm, but if the temperature increases further the alarm will be set again unless a higher alarm limit is entered.

Cumulative motor running time exceeds the present alarm limit. This limit is initially set to zero, so if the motor is started without an alarm limit being keyed in first, the alarm will occur. The alarm can be cleared by stopping the motor (with STEP) or by entering a new alarm limit (with ADDR).

OPERATION

1. Connect the system as shown in FIGURE D-3.
2. Load the program from tape:

starting address = 8000 (H,L pair)
3. Begin program execution at 8200 by depressing RST, RUN. The program has a self-check feature which will display an address in the left-hand hex digits if the program was loaded incorrectly. This address will be the first of a block of 16 memory locations in which an error was detected. If this occurs, reload the program.

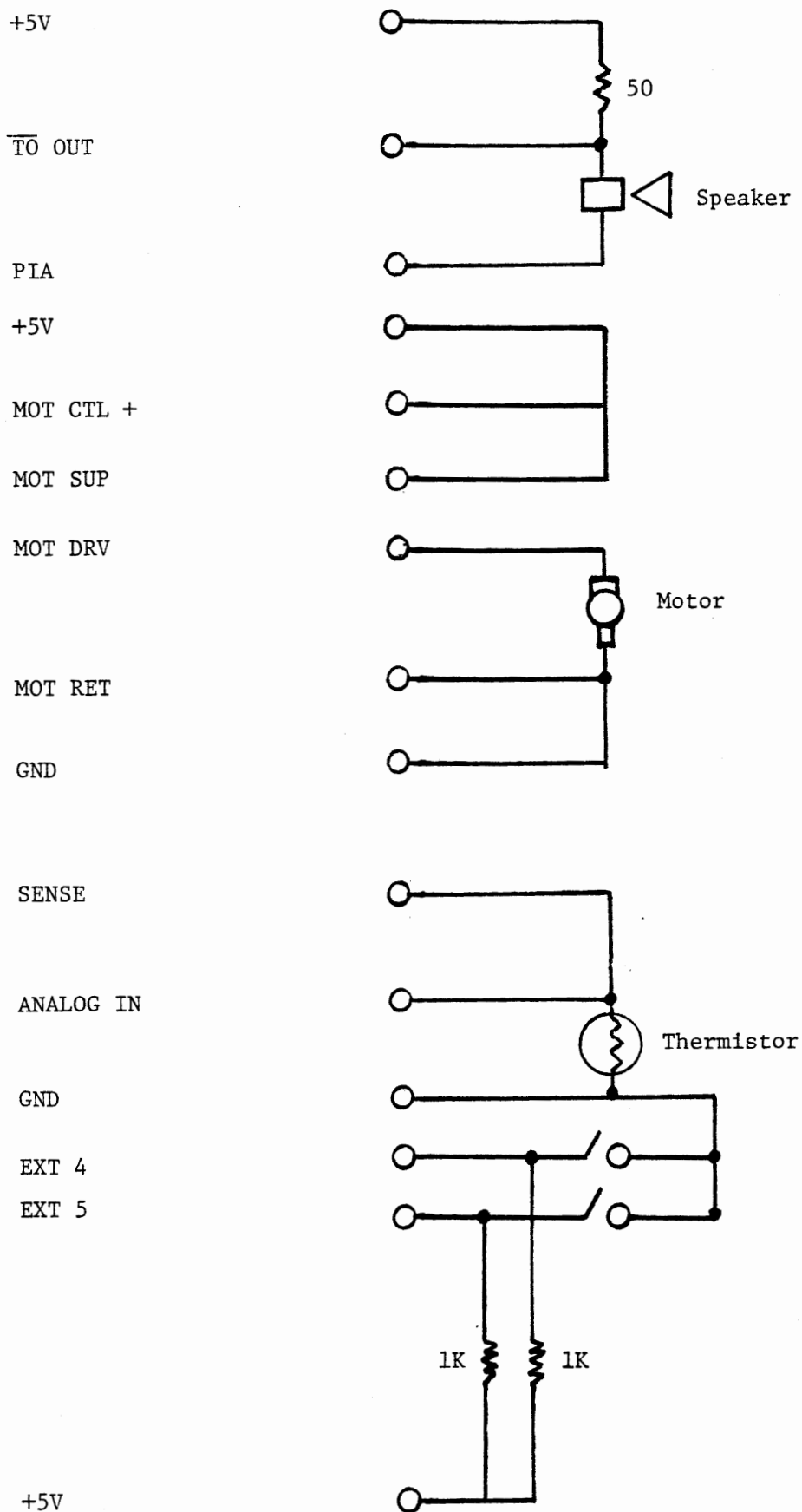
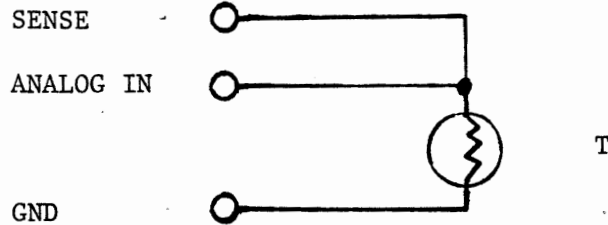


FIGURE D-3

- If no error is indicated, depress NEXT. The time-of-day display will appear with an initial value of zero. It will immediately begin counting seconds. Set the display to the correct time by entering hours and minutes on the keyboard and then pressing MEM.

- Calibrate the thermistor using the following procedure:



Turn SENSE pot and ANALOG IN fully to the right.

With the program loaded, press:

RESET

RUN

REG

Display will show: 0000 132.6

After about one second the display will show some other value, perhaps: 0000 052.4

Now adjust ANALOG IN pot until the display shows the room temperature. Note that the smallest adjustments available near room temperature are 0.3 to 0.4 degrees, and this program measures temperature only once per second.

- The following sequence of operations is suggested to demonstrate the system:

- Set motor running time limit to 2 minutes.

2 ADDR

- Start motor at nominal speed.

5 0 RUN

Display will show total running time.

- Decrease motor speed.

4 0 RUN

- Stop the motor.

STEP

- Set temperature limit to 40° celsius.

4 0 REG

- Display time-of-day.

NEXT

- Momentarily close EXT 4 switch. The alarm will sound and E4 will be displayed.

- 8) Clear the alarm.

CLR

- 9) Display temperature.

REG

- 10) Place the thermistor in a hot liquid (40° C). When the temperature exceeds 40° C, the alarm will sound.

- 11) Clear the alarm.

CLR

The display will show time-of-day. Remove the thermistor from the liquid.

- 12) Start the motor at low speed.

4

0

RUN

- 13) Momentarily close the EXT 5 switch. The alarm will sound and E5 will be displayed.

- 14) Clear the alarm.

CLR

- 15) Display motor running time. When it reaches 2 minutes the alarm will sound.

- 16) Clear the alarm.

CLR

The alarm restarts because the motor is still running.

- 17) Stop the motor.

STEP

The alarm clears automatically when the motor is stopped.

END OF DEMONSTRATION.

PONG

D.2.2

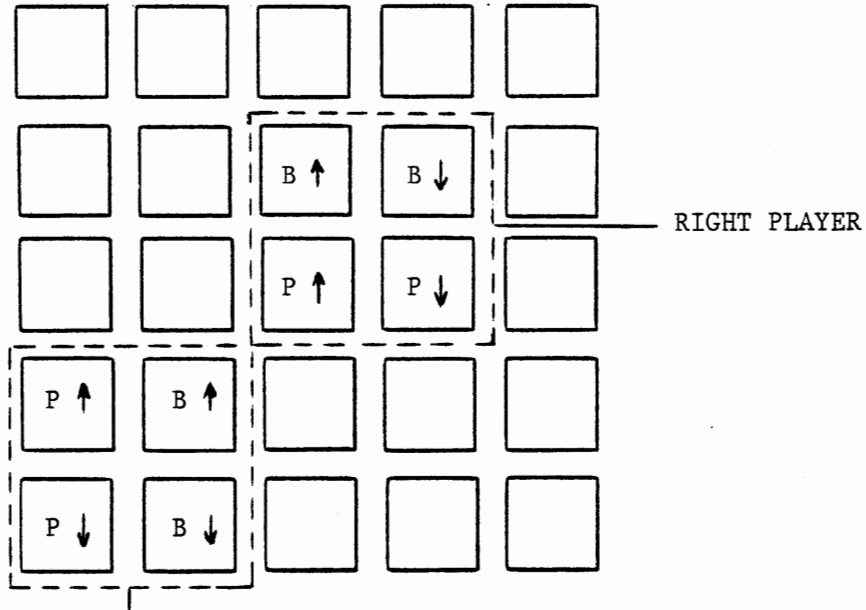
STORE INTO MEMORY STARTING AT 8200.

DEPRESS.

RST

RUN

CONTROL KEYS ARE AS FOLLOWS:



LEFT PLAYER

P MOVE PADDLE UP

P MOVE PADDLE DOWN

B MOVE BALL UP

B MOVE BALL DOWN

LOCATION 830C CONTROLS BALL SPEED.

0C=NORMAL SPEED, 08=FAST.

FIRST PLAYER TO SCORE 15 POINTS WINS.

TO START NEW GAME, DEPRESS ANY KEY.

ICS PONG GAME using MTS board^{D-111}
 only. Hex keypad controls ball & paddles:
 Left player: 4-paddle up, 0-down
 5-ball English up, 1-down
 Right player: A-paddle up, B-down
 E-ball English up, F-down
 Author: Ted Lappin

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8200	21	STRT	LXI	H,	LEFT	Clear score - Load
1	FE					HL with score location
2	82					
3	AF		XRA	A		
4	4F		MOV	C,	A	Clear ball control
5	77		MOV	M,	A	
6	23		INX	H		
7	77		MOV	M,	A	
8208	3E	INIT	MVI	A,	40H	Set Paddle Positions
9	40					to Middle
A	32		STA	D1		(1=TOP;40=MIDDLE;
B	F8					8= BOTTOM)
C	82					
D	32		STA	DB		
E	FF					
F	83					
8210	3E		MVI	A,	8FH	Set all SCAN lines on
1	8F					Keyboard to 0
2	D3		OUT	PORT0C		
3	FA					
4	06		MVI	B,	40H	Set initial ball position
5	40					to middle
6	21		LXI	H,	D4	Set ball position in HL
7	FB					
8	83					
8219	78	LOOP	MOV	A,	B	Load ball into display
A	B6		ORA	M		
B	77		MOV	M,	A	
C	CD		CALL	DELAY		Delay for 1 frame
D	0B					
E	83					
F	DB		IN	PORT0A		Get left player

FIGURE D-4

A D D R		CODE																			
CODING SHEET	8	2	2	0	F	8													paddle info	c/7/78	
				1	E	6		A	N	I		1	1	H					Bit 4 = Up, Bit 0 = Down		
				2	I	I															
				3	E	A		J	P	E		N	C	H	L					If both the same.	
				4	2	F														ignore (No Change	
				5	8	2														Left)	
				6	E	6		A	N	I		0	1							See if Down	
				7	0	1															
				8	I	I		L	X	I		D	,	D	1					Load paddle location	
				9	F	8															
MICROCOMPUTER TRAINING SYSTEM			A	8	3																
			B	C	D		C	A	L	L		C	H	P					Change Paddle Position		
			C	0	0																
			D	8	3																
			E	I	2		S	T	A	X		D								Save	
		8	2	2	F	D	B	N	C	H	L	I	N		P	O	R	T	0	A	Get right player paddle
		8	2	3	0	F	8													info	
				1	E	6		A	N	I		0	C	H						Bit 2 = Up, Bit 3 = Down	
				2	0	C															
				3	E	A		J	P	E		N	C	H	R					If both the same,	
INTEGRATED COMPUTER SYSTEMS			4	3	F														ignore (No Change		
			5	8	2															Right)	
			6	E	6		A	N	I		0	8	H						See if Down		
			7	0	8																
			8	I	I		L	X	I		D	,	D	8						Load paddle position	
			A	F	F																
			B	C	D		C	A	L	L		C	H	P					Change Paddle Position		
			C	0	0																
			D	8	3																
			E	I	2		S	T	A	X		D								Save	
	8	2	3	F	7	D	N	C	H	R	M	O	V	A	,	L			See if end of field		
	8		0																		
			1																		
			2																		
			3																		
			4																		
			5																		
			6																		
			7																		
			8																		

FIGURE D-5

		A	D	D	R	CODE										
CODING SHEET	8	2	4	0	F	E		C	P	I	F	8	H	If left of Net		
				1	F	B										
				2	C	A		J	Z		L	B	C	Then → LBC ;		
				3	7	C										
				4	8	2										
				5	F	E		C	P	I	F	F	H	Else IF right side		
				6	F	F								of net		
				7	C	A		J	Z		R	B	C	Then → RBC ;		
				8	8	6										
				9	8	2										
MICROCOMPUTER TRAINING SYSTEM		A			A	F		X	R	A	A			Else Clear Old Position		
		B			7	7		M	O	V	M	,	A			
		C			B	9		C	M	P	C			Who has the ball?		
		D			C	A		J	Z		N	B	C	No-one → No Ball Change		
		E			7	8										
		F			8	2										
		8	2	5	0	0	0		D	C	R	C			Count 1 less frame	
				1	7	9			M	O	V	A	,	C		
				2	3	F			C	M	C					
				3	1	7			R	A	L					
INTEGRATED COMPUTER SYSTEMS				4	D	A		J	C		L	F	T	C	IF C ₇ =1	
				5	6	3									Then → LFTC	
				6	B	2									Else → RTC	
				7	D	B	RTC	I	N		P	O	R	T	0	A
				8	F	5										
				9	E	6			A	N	I	C	0	H		Bit 7 = Down, Bit 6 = Up
				A	C	8										
				B	E	A			J	P	E	N	C	H		IF A ₇ = A ₆
				C	7	1										Then → No Change
				D	B	2										
			E	E	6			A	N	I	B	0	H		If Down	
	8	2	5	F	8	0									Then	
	8		0													
			1													
			2													
			3													
			4													
			5													
			6													
			7													
			8													

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE								
8	2	6	0	C3		JMP	SAVB					→ Go Change
			1	6C								
			2	82								
8	2	6	3	DB	LFTC	IN	PORT	0A				Check for left Player
			4	F8								Change
			5	E6		ANI	22H					Bit 5=Up, Bit 1=Down
			6	22								
			7	EA		JPE	NCH					If A ₅ =A ₁
			8	71								Then → No Change
			9	82								
			A	E6		ANI	02H					If Down
			B	02								Then
			C	78	SAVB	MOV	A, B					Load Old Ball Position
			D	CD		CALL	CHB					Change Ball Position
			E	01								
			F	83								
8	2	7	0	47		MOV	B, A					Save in Ball's Reg.
8	2	7	1	79	NCH	MOV	A, C					Get Change Reg.
			2	E6		ANI	0FH					If Count = 0
			3	0F								Then → No Ball Change
			4	C2		JNE	NBC					
			5	78								
			6	82								
			7	4F		MOV	C, A					Else Set Count = 0
8	2	7	8	2B	NBC	DCX	H					Change Ball Position -
			9	C3		JMP	LOOP					Either Up or Down
			A	19								
			B	82								
			C	7E	LBC	MOV	A, M					If Paddle Position ≠
			D	B8		CMP	B					Ball Position
			E	C2		JNE	LLOSE					Then → Left Lose
			F	98								
8	2	8	0	82								
			1									
			2									
			3									
			4									
			5									
			6									
			7									
			8									

FIGURE D-7

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE							
8		0									
8	2	8	1	0E		M	V	I	C,	03H	Else New Ball Counter
			2	03							
			3	C3		J	M	P	R	E	V
			4	0D							
			5	02							
8	2	8	6	7E	RBC	M	O	V	A,	M	If Paddle Position ≠
			7	B8		C	M	P	B		Ball Position
			8	C2		J	N	Z	R	L	O
			9	9E							Then → Right Lose
			A	02							
			B	0E		M	V	I	C,	03H	Set New Ball Counter
			C	03							
8	2	8	D	3A	REV	L	D	A	N	B	C
			E	78							Reverse Ball Direction
			F	02							by Complementing
			8	2	9	0	E	F	X	R	I
			1	08							via XOR. (Classical
			2	32		S	T	A	N	B	C
			3	78							
			4	02							
			5	C3		J	M	P	N	B	C
			6	78							Go Change Ball Position
			7	02							
8	2	9	8	21	LLOSE	L	X	I	H,	L	E
			9	FE							FT Add 1 to Right's Score
			A	02							
			B	C3		J	M	P	S	C	O
			C	A1							
			D	02							
			E	21	RLOSE	L	X	I	H,	R	I
			F	7F							GH T Add 1 to Left's Score
8	F	A	0	02							
			1								
			2								
			3								
			4								
			5								
			6								
			7								
			8								

FIGURE D-8

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE										
8	0													
82A	1	7E	SCOR	M	O	V	A	,	M		Load Score And			
	2	C6		A	D	I	0	1	H		Add 1			
	3	01												
	4	27		D	A	A					with Decimal Adjust			
	5	77		M	O	V	M	,	A		Save			
	6	FE		C	P	I	1	5			IF (A) = 15			
	7	15												
	8	C2		J	N	Z	N	W	I	N	THEN → No Winner Yet			
	9	D5												
A		82												
B		2C		I	N	R	L				Else			
C		21		L	X	I	H	,	D	8	IF L+1 = 0			
D		FF												
E		83												
F		CA		J	Z		R	T	L		Then Right Lost			
82B	0	B4												
	1	82												
	2	2E		M	V	I	L	,	F	B	H	Else Left Lost		
	3	FB												
82B	4	E5	RTL	P	U	S	H		H		Save Position			
	5	CD		C	A	L	L		C	L	E	A	R	Clear Display
	6	87												
	7	02												
	8	E1		P	O	P			H					
	9	36		M	V	I	M	,	"	E	"			
A		79												
B		2B		D	C	X	H							
C		36		M	V	I	M	,	"	S	"			
D		6D												
E		2B		D	C	X	H							
F		36		M	V	I	M	,	"	O	"			
82C	0	3F												
	1													
	2													
	3													
	4													
	5													
	6													
	7													
	8													

FIGURE D-9

		A	D	D	R	CODE														
CODING SHEET	8	0																		
		82C	1	2	8		D	C	X	H										
			2	3	6		M	V	I	M	,	"	L	"						
			3	3	8															
			4	3	E		M	V	I	A	,	F	F	H						Hold Display
			5	F	F															
			6	C	D		C	A	L	L		D	I	S						
			7	0	D															
			8	8	3															
	MICROCOMPUTER TRAINING SYSTEM			9	D	B	NKY	I	N	P	O	R	T	0	A					
		A	F	8																begin new game
		B	8	7		O	R	A	A											
		C	E	A		J	P	E	N	K	Y									
		D	C	9																
		E	8	2																
		F	C	D		C	A	L	L		C	L	E	A	R					clear Display
		8 2 D	0	8	7															
			1	0	2															
			2	C	3		J	M	P		S	T	R	T						Start New Game
INTEGRATED COMPUTER SYSTEMS			3	0	0															
			4	8	7															
			5	C	D	NWIN	C	A	L	L		C	L	E	A	R				
			6	8	7															
			7	0	2															
			8	R	1		L	X	I	H	,	L	E	F	T					LOAD left losing Score
			9	F	E															
		A	8	2																
		B	1	1		L	X	I	D	,	D	6								Display on Right
		C	F	D																
	D	8	3																	
	E	7	E		M	O	V	A	,	M										
	F	C	D		C	A	L	L		D	B	Y	2						via Monitor	
	8 2 E	0	9	8																
	8 2 E	1	0	2																
		2																		
		3																		
		4																		
		5																		
		6																		
		7																		
		8																		

FIGURE D-10

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

A	D	D	R	CODE										
8	0													
	1													
82E	2	2	3		INX	H								Get Right Losing Score
	3	7	E		MOV	A,	M							
	4	C	D		CALL	L	DB	Y	2					Display on Left
	5	9	8											
	6	0	2											
	7	3	E		MVI		A,	4	0	H				Hold for a while
	8	4	0											
	9	C	D		CALL		DIS							
	A	0	D											
	B	8	3											
	C	C	D		CALL		CLEAR							
	D	8	7											
	E	0	2											
	F	A	F		XRA		A							
82F	0	4	F		MOV	C,	A							Clear Ball Counter
	1	C	3		JMP		INIT							Go Back
	2	0	8											
	3	8	2											
82F	4	F	E	DOWN	CPI		0	8	H					If Down & on Bottom
	5	0	8											
	6	C	8		RZ									Then Do Not Change
	7	B	7		ORA		A							Else Clear Carry
	8	1	F		RAR									and Change Position
	9	1	F		RAR									(A ₁ → A _{N-2})
	A	1	F		RAR									
	B	C	9		RET									
	C													
	D													
82F	E			LEFT										Left's Misses (Right's Score)
82F	F			RIGHT										Right's Misses (Left's Score)
8	0													
	1													
	2													
	3													
	4													
	5													
	6													
	7													
	8													

FIGURE D-11

		A	D	D	R	CODE									
CODING SHEET	8 3 0 0	1	A	CHP	L	D	A	X	D		Change Paddle c/7/78				
		1	C	A	CHB	J	Z		D	O	W	N	Assume we have position for		
		2	F	4									Ball		
		3	8	2											
		4	F	E		C	P	I	0	1	H		IF UP & ON TOP		
		5	0	1											
		6	C	8		R	Z						Then Do Not Change		
		7	1	7		R	A	L							
		8	1	7		R	A	L							
		9	1	7		B	A	L							
MICROCOMPUTER TRAINING SYSTEM	A	C	9		R	E	T								
	B	3	E	DELAY	M	V	I	A	,	0	C	H	Delay for 1 frame		
	830	C	0	C	**								** BALL SPEED **		
	830	D	1	1	DIS	L	X	I	D	,	0	7	B	4	Main Delay
		E	B	4											
		F	0	7											
	8 3 1 0	1	D	DEL	D	C	R	E						Delay by decrement	
		1	C	2		J	N	Z	D	E	L				
		2	1	0											
		3	0	3											
	4	1	5		D	C	R	D							
	5	C	2		J	N	Z	D	E	C					
	6	1	0												
	7	0	3												
	8	3	D		D	C	R	A							
	9	C	2		J	N	Z	D	I	S					
INTEGRATED COMPUTER SYSTEMS	A	0	D												
	B	0	3												
	C	C	9		R	E	T								
	D														
	E														
	F			**	(Change Ball Speed by Changing Location 830C from 0C to 08)										
	8	0													
		1													
		2													
		3													
	4														
	5														
	6														
	7														
	8														

FIGURE D-12

APPENDIX E

RS 232c Interface System

The RS 232c Interface System consists of the following I/O driver subroutines and Interface Training System board connections.

Software is used to convert parallel data, transmitted by the MTS, into a serial stream of bits consisting of one low start bit, eight bits of data, and one high stop bit. A Software delay subroutine is used to produce a baud rate of 300 bits/second or 30 characters/second. This delay program may be modified to allow transmission rates of up to 4800 baud (480 characters/second). The ITS hardware option and connections of section E.1 are required to convert the digital data up to the standard +/- 12 volt EIA-RS232 connector in order to communicate with a terminal.

E.1 ICS RS232 SOFTWARE

The RS232 Software includes the following subroutines:

- * IMSG - Input a line of text from the terminal, terminated by a Carraige Return, via ICHR.
- Entry Point 8297
- * ICHR - Input a character from the terminal into register E via Port 1A0.
- Entry Point 82B0
- * OCHR - Output a character from register E to the terminal via Port 1C1.
- Entry Point 82D0.
- * OMSG - Output a block of characters, terminated with a CNTL-C (03H), to the terminal via OCHR.
- Entry Point 82F6.
- * Specifications, flowcharts and source code for the above programs are given in section E.3.

E.2 Required Connections for RS 232 Interface

E.2.1 Parts List

QTY	Description
1	16-pin DIP plug
7	12" lengths of ~22 gauge wire jumpers
1	25 pin CANNON DB-25s-5 connector
1	1 K ohm resistor - 1/4 watt
5	spade clips (optional)
*	-12 volt power supply (most terminals)

E.2.2 Fabrication Procedure

DIP plug connector

- 1 Solder a jumper wire to pin 2 of DIP plug. (Fasten spade clip to other end of wire.)
- 2 Solder a jumper wire to pin 16 of DIP plug. (Fasten spade clip to other end of wire.)

CANNON connector

- 3 Connect pins 5, 6 and 8 together.
- 4 Solder one end of a jumper wire to this connection (pin 5, 6 or 8).
- 5 Solder the other end of this jumper to the 1K resistor (insulate solder joint with heat shrink tubing or electrical tape).
- 6 Connect pins 1 and 7 together.
- 7 Solder a jumper wire to pin 1 (or 7). (Fasten spade clip to other end of jumper.)

- 8 Solder a jumper wire to pin 2. (Fasten spade clip to other end of jumper.)
- 9 Solder a jumper wire to pin 3. (Fasten a spade clip to other end of jumper.)

E.2.3 ITS Board Connections

- 1 CANNON pin 1 (or 7) to ITS ground (TIE Block GND or CASSETTE GND).
- 2 CANNON pin 2 to ITS terminal strip pin 4 (RS 232 REC).
- 3 CANNON pin 3 to terminal strip pins 2 and 3 (TTY SEND and TTY RET).
- 4 CANNON pin 5 (or 6 or 8) jumper with 1K resistor to TIE Block +12v.

E.2.4 -12 Volt Power Supply

- * 5 Connect the -12 volt power supply lead and its ground to the ITS TIE Block -12 and GND tie points, respectively.
- * NOTE - While the RS 232 standards specify a high (or 1) signal level of +12 volts and a low (or 0) signal of -12 volts, some terminals (such as the Lear-Seigler ADM-3) will recognize a signal level of 0 volts or less as a low. For these devices the -12v supply is not necessary.

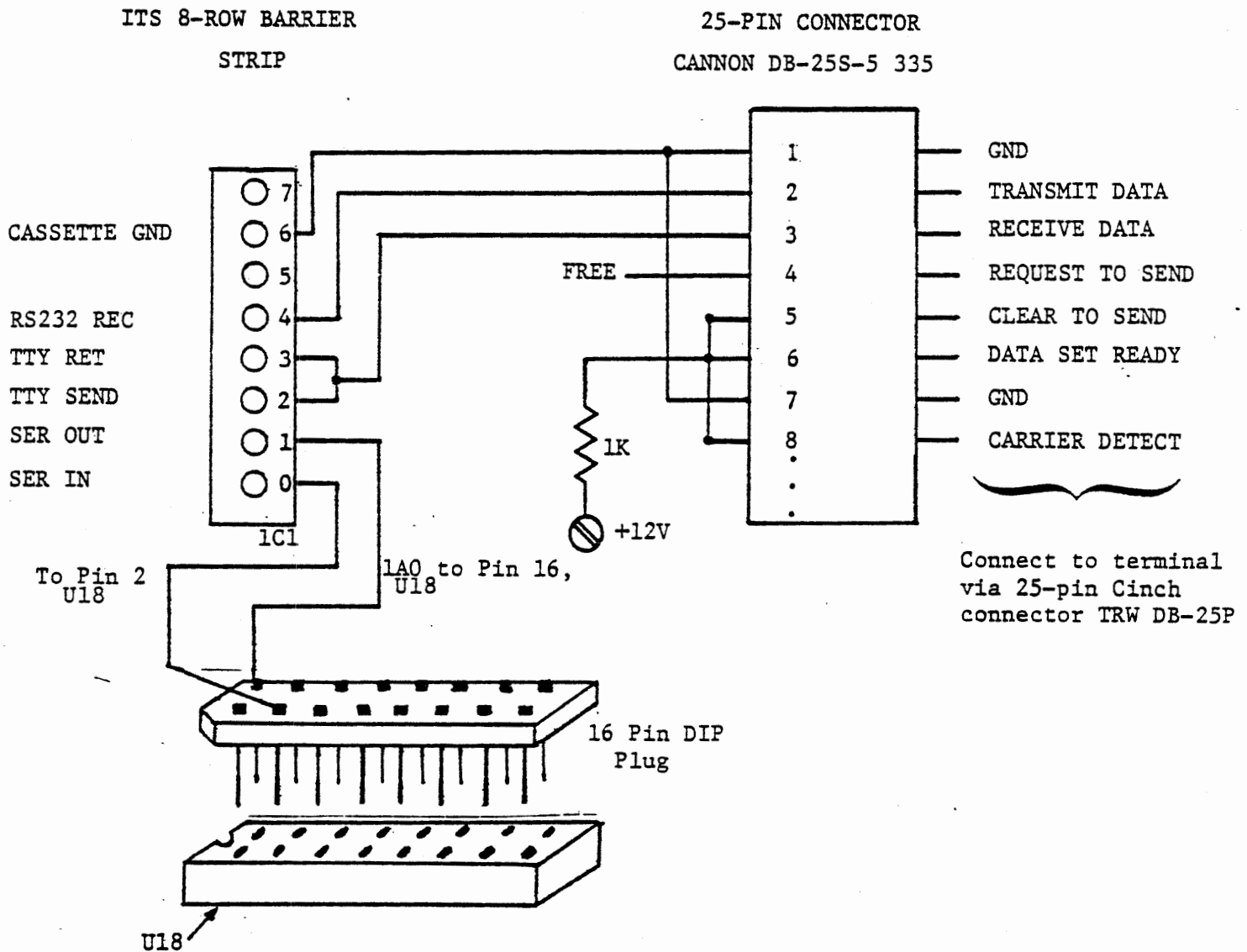


FIGURE E-1

<u>TERMINAL SIGNALS</u>	<u>25-PIN CANNON CONNECTOR</u>	<u>ITS CONNECTIONS</u>	<u>NOTES</u>
GND	1	GND	Screw terminal #7
TRANSMIT DATA	2	RS232 REC	Screw Terminal #4
RECEIVE DATA	3	"TTY SEND"/ "TTY RET"	Screw Terminal #2&3
REQUEST TO SEND	4	unconnected	
CLEAR TO SEND	5	Pulled up to +12V through 1K resistor	
DATA SET READY	6	Pulled up to +12V through 1K resistor	
GND	7	GND	
CARRIER DETECT	8	Pulled up to +12V through 1K resistor	

SUMMARY OF SIGNAL CONNECTIONS

FIGURE E-2a

BIT 8 = 0
Parity = INH
STOP = 1
Data = 8
Parity = Don't Care
RS232
FDX
BAUD RATE = 300

ADM3A SWITCH SETTINGS

FIGURE E-2b

E.3 ICS RS232 Software Specifications

The following pages contain the formal specifications, flowcharts, and source code listings for the RS232 Software. The student is advised to note the format used in this documentation for his/her own efforts.

E.3.1.1 Subroutine IMMSG

This subroutine inputs a string of characters from the terminal and echos the characters back to the terminal until it encounters a Carriage Return character (0DH). The routine stores the input string beginning at the address specified in the HL register pair. The parity bit (high order bit 7) is masked out before the ASCII character is stored. The last character in the buffer is always ETX (03H or CNTL-C). Register C contains the maximum input string length acceptable (input buffer size).

The PMPT entry point outputs a question mark ('?') as a prompt character prior to invoking IMMSG. All arguments are the same.

* Entry Point 8297

* Arguments

- Upon entry

- H,L register pair contains the input buffer start address

- C contains the maximum input buffer length minus 1

* Upon return

- C contains the number of input buffer bytes remaining

- A contains 03H

- B and D contain 00H

- E contains last input character
- H,L register pair contain the address of the last input buffer entry (the CNTL-C, 03H, or ETX)
- The Input Buffer contains the inputted character string where the last character in the buffer is an ETX (03H or CNTL-C) character. The Carraige Return character is not stored.
- The Stack has been used and restored.

* Subroutine used

- OCHR at 82D0H
- ICHR at 82B0H

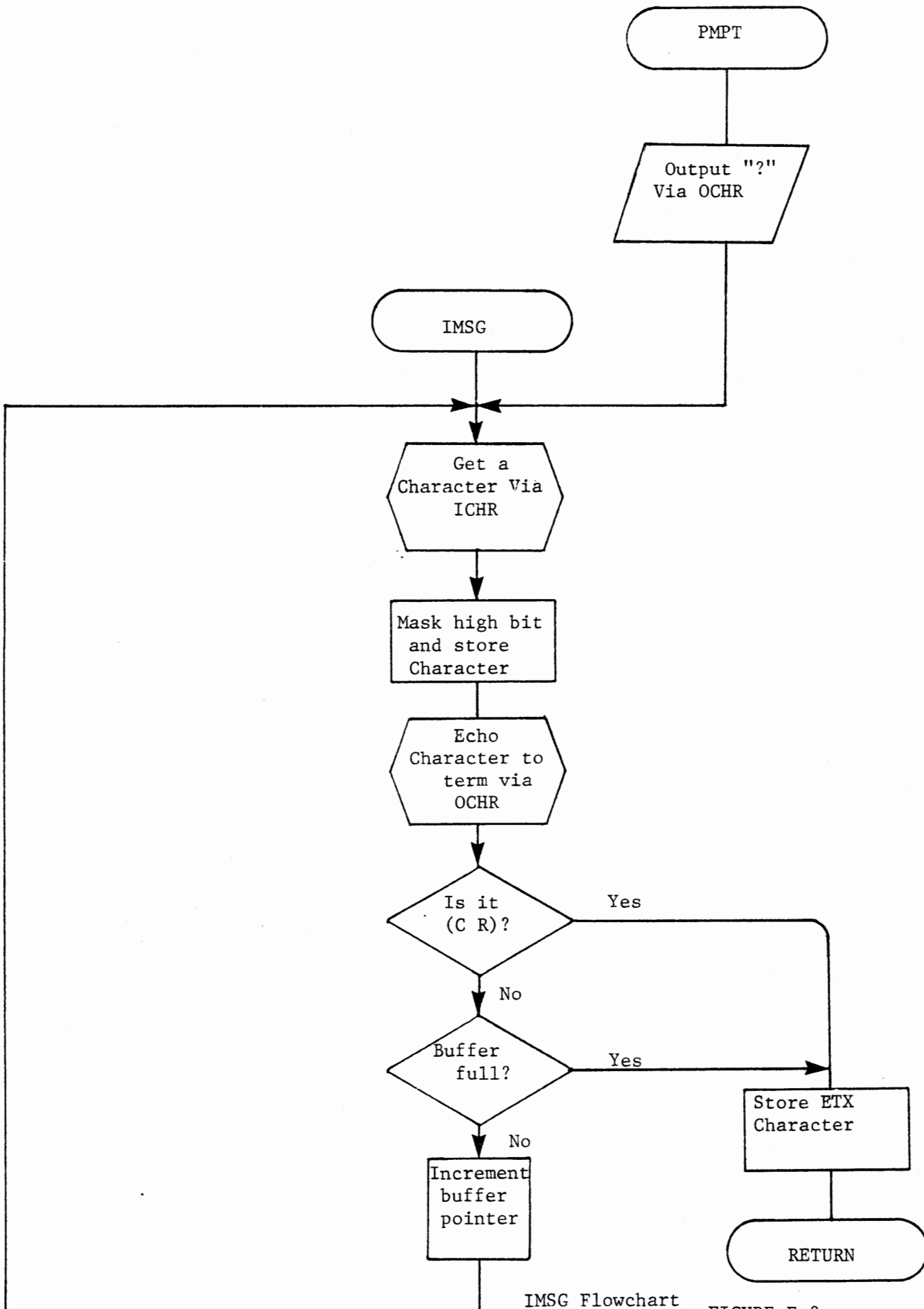
* Other entry points

- PMPT - Output a prompt character ('?') before invoking IMSG.
Entry point 8290H

* Sample usage

21 00 83	LXI	H,INBF	Load input buffer address
0E 1F	MVI	C,BFLEN	Load input buffer length
CD 90 82	CALL	PMPT	Prompt with '?' and input string

E.3.1.2



MSG Flowchart

FIGURE E-3

A D D R		CODE									
CODING SHEET	8 2 9 0	C 5	P M P T	P U S H	B					Prompt:	c/7/78
	1	1 E		M V I	E	,	" ? "			Output '?' as a	
	2	3 F								prompt character	
	3	C D		C A L L	O C H R						
	4	D 0									
	5	8 2									
	6	C 1		P O P	B						
	8 2 9 7	C 5	I M S G	P U S H	B					I M S G Entry	
	8	C D		C A L L	I C H R					Get a character from the	
	9	B 0								keyboard	
MICROCOMPUTER TRAINING SYSTEM	A	8 2									
	B	E 6		A N I	7 F					Mask-Out the Parity Bit,	
	C	7 F								bit 7 (MSB)	
	D	7 7		M O V	M A						
	E	C D		C A L L	O C H R					Echo the inputted	
	F	D 0								character back to CRT	
	8 2 A 0	8 2									
	1	C 1		P O P	B						
	2	7 B		M O V	A	E					
	3	F E		C P I	C R					Is it a Carriage Return?	
4	0 D										
5	C A		J Z	D O N E					Yes, finish up		
6	A D										
7	8 2										
8	0 D		D C R	C					No,		
9	2 3		I N X	H					Update buffer pointer		
A	C 2		J N Z	I M S G					and character counter.		
B	9 7								Is buffer full?		
C	8 2								No, → Get next character		
8 2 A D	3 6	D O N E	M V I	M	,	0 3			Yes, Put ETX character		
E	0 3								into buffer and		
F	C 9		R E T						return.		
INTEGRATED COMPUTER SYSTEMS	8 0										
	1										
	2										
	3										
	4										
	5										
	6										
	7										
8											

FIGURE E-4

E.3.2.1 Subroutine ICHR

This subroutine inputs a character from the terminal keyboard through Port 1A0, with no translation, into the A and E registers.

* Entry point 82B0

* Arguments

- No entry arguments
- Upon return
 - A contains the last character inputted
 - E contains a copy of A (input character)
 - B and C are 00H
 - D, H and L are unaffected

* Subroutines used

- DLY of OCHR (at 82E9)
- DLY1 of OCHR (at 82EB)

* Sample usage

```
CD B0 82      CALL  ICHR      Input a character
```

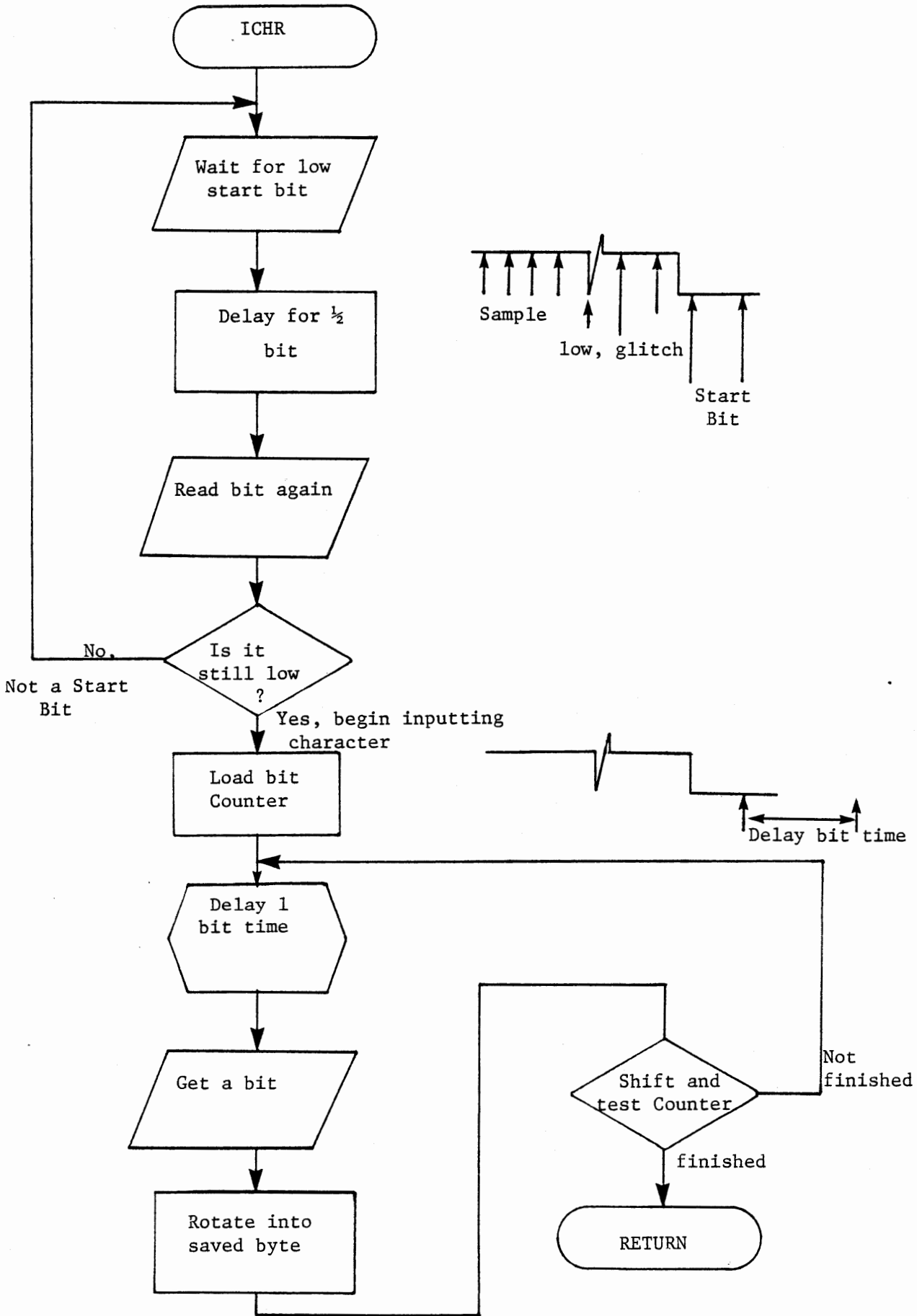


FIGURE E-5

A D D R		CODE					CH	
			;	GET	K/B		CHAR INTO REG A	
82B0	DB	ICHR	IN	PORT	1	A	Get a bit	
1	04							
2	1F		RAR					
3	DA		JC		ICHR		Wait for low	
4	B0						start bit	
5	82							
6	0E		MVI	C	2		Delay for 1/2 bit	
7	02						time	
8	CD		CALL	DLY	1			
9	EB							
A	82							
B	DB		IN	PORT	1	A	Read bit again.	
C	04							
D	1F		RAR				Is it still low?	
E	DA		JC		ICHR		False alarm	
F	B0							
82C0	82							
1	1E		MVI	E	80		Bit counter	
2	80							
82C3	CD	ILP	CALL	DLY			Delay full bit	
4	E9						time	
5	82							
6	DB		IN	PORT	1	A	Get a bit	
7	04							
8	1F		RAR				CY ← (Ae)	
9	7B		MOV	A	E		Get previous bits	
A	1F		RAR				A _n ← (CY)	
B	5F		MOV	E	A		SAVE New Byte	
C	D2		JNC	ILP			Get next bit,	
D	C3						unless CY shows	
E	82						we've got all 8	
F	C9		RET					
82D0							Returns with last	
							Keyboard character	
							in Regs A and E	

FIGURE E-6

c/7/78

5.3.3.1 Subroutine OCHR

This subroutine outputs the character in the E register to the terminal via Port 1C1 at 300 baud (30 chars/sec). The baud rate may be changed by changing the programmed delay in the DLY subroutine at address 82ECH.

Other entry points are DLY and DLY1 for the appropriate delays for the 300 baud (or higher) data rates.

* Entry point(s)

- 82D0 for OCHR
- 82E9 for DLY
- 82ED for DLY1

* Arguments

- Upon entry
 - E contains the character to be outputted
- Upon return
 - E contains the outputted character
 - A contains 03H (or ETX)
 - B, C and D contain 00H
 - H and L are unaffected

* Sample usage

```

06 21      MVI    E,"A"      Output an 'A' to the CRT
CD D0 82   CALL   OCHR      terminal.

```

* Other entry points

- DLY at 82E9
 - Causes the appropriate delay for a 300 baud data rate. The baud

rate may be changed up to 4800 baud by recoding address 82EC as follows:

82EC	baud rate	chars/sec
70H	300	30
38H	600	60
0BH	1200	120
04H	4800	480

- DLY1 at 82EB

- Alternate delay entry allows 1/2 bit time delay (used by ICHR). Also allows 110 baud rate delay. User must load register C with appropriate counter (see source code for OCHR).

E.3.3.2

Subroutine OCHR

Subroutine OUTB

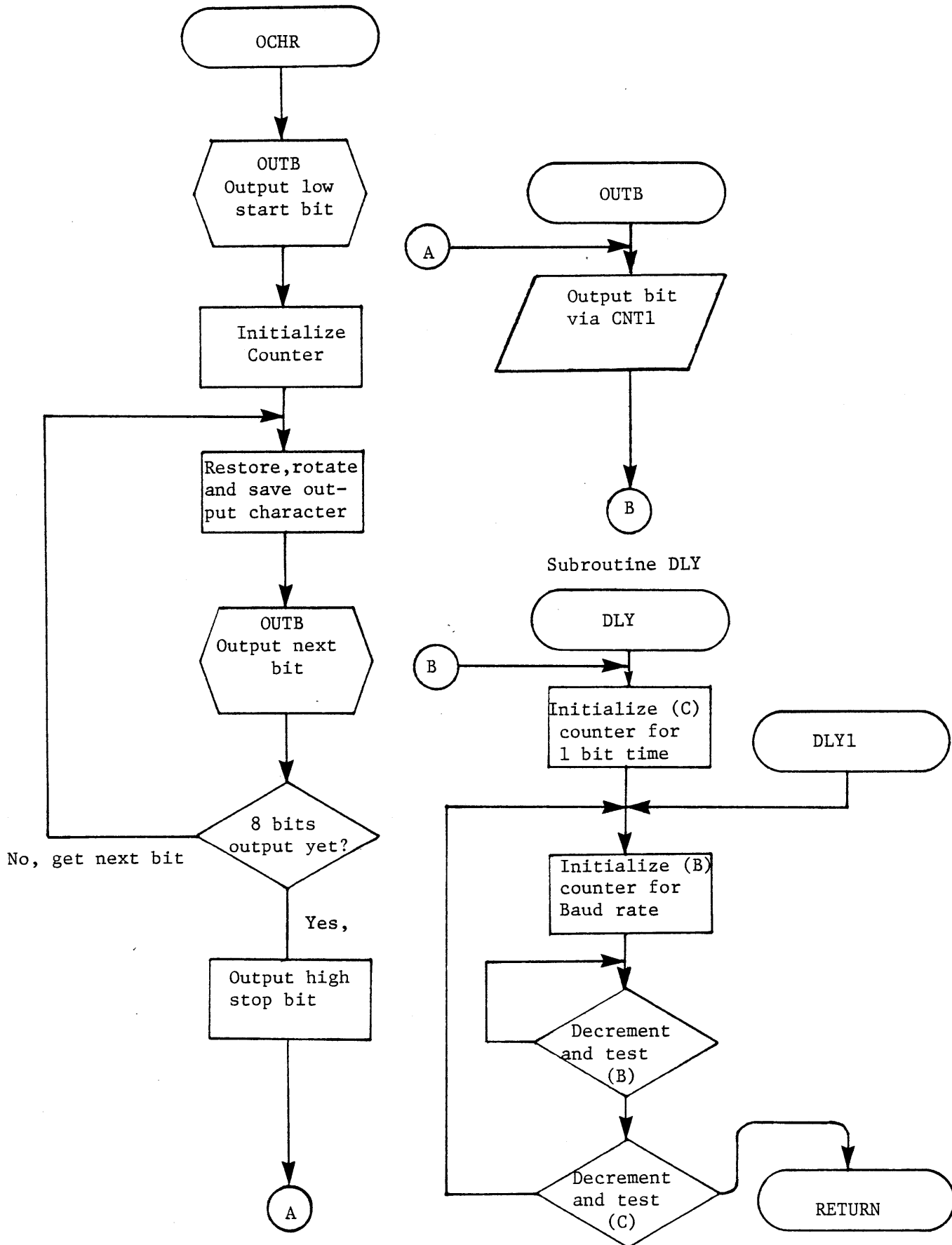


FIGURE E-7

		A	D	D	R	CODE						
		Output contents of E to RS232 device										
		at 300 baud										
CODING SHEET	82D0	3E	OCHR	MVI	A,	02	Output a low start bit					
	1	02										
	2	CD		CALL		OUTB						
	3	E7										
	4	82										
MICROCOMPUTER TRAINING SYSTEM	5	AF		XRA		A	Clear Carry					
	6	16		MVI		D,	8	Initialize bit counter				
	7	08										
	82D8	7B	OLP	MOV		A,	E	Get Output Character				
	9	0F		RRC								
	A	5F		MOV		E,	A	Save Rotated Character				
	B	3E		MVI		A,	01	If CY then CNT1 ← 03				
	C	01					else CNT1 ← 02					
	D	8F		ADC		A						
	E	CD		CALL		OUTB	Output bit					
	F	E7										
	82E0	82										
	1	15		DCR		D	Count bits until					
	2	C2		JNZ		OLP	all 8 are done					
	3	D8										
4	82											
5	3E		MVI		A,	03	Finished, Output high					
6	03					stop bit						
82E7	D3	OUTB	OUT		CNT1	Output the bit						
8	07											
82E9	0E	DLY	MVI		C,	03	Load counter for 1					
A	03					bit time						
82EB	06	DLY1	MVI		B,	70	Load timer for baud					
C	70					rate: 70 = 300 baud						
82ED	05	DLY2	DCR		E	3E = 600 baud						
E	C2		JNZ		DLY2	1B = 1200 baud						
F	ED					0B = 2400 baud						
82FD	82					04 = 4800 baud						
1	0D		DCR		C							
2	C2		JNZ		DLY1	Loop 'till done						
3	EB											
4	82											
5	C9		RET									

FIGURE E-8

3.3.4.1 Subroutine OMSG

This subroutine will output a block of characters, whose starting address is specified in the HL register pair, via OCHR until an ETX (03H or CNTL-C) character is encountered. (02H, 01H and 00H also terminate transmission.)

* Entry point 82F6

* Arguments

- Upon entry

- HL register pair contain the output buffer start address

- Upon return

- A contains 03H (ETX character)

- B, C and D contain 00H

- E contains the last character output

- HL contain the address of the last character output +1 (The address of the ETX byte.)

* Subroutines used

- OCHR at 82D0

* Sample usage

21 00 83	LXI	H,OBUF	Load output buffer address
CD F6 82	CALL	OMSG	and output it

E.3.4.2

Subroutine OMSG Flowchart

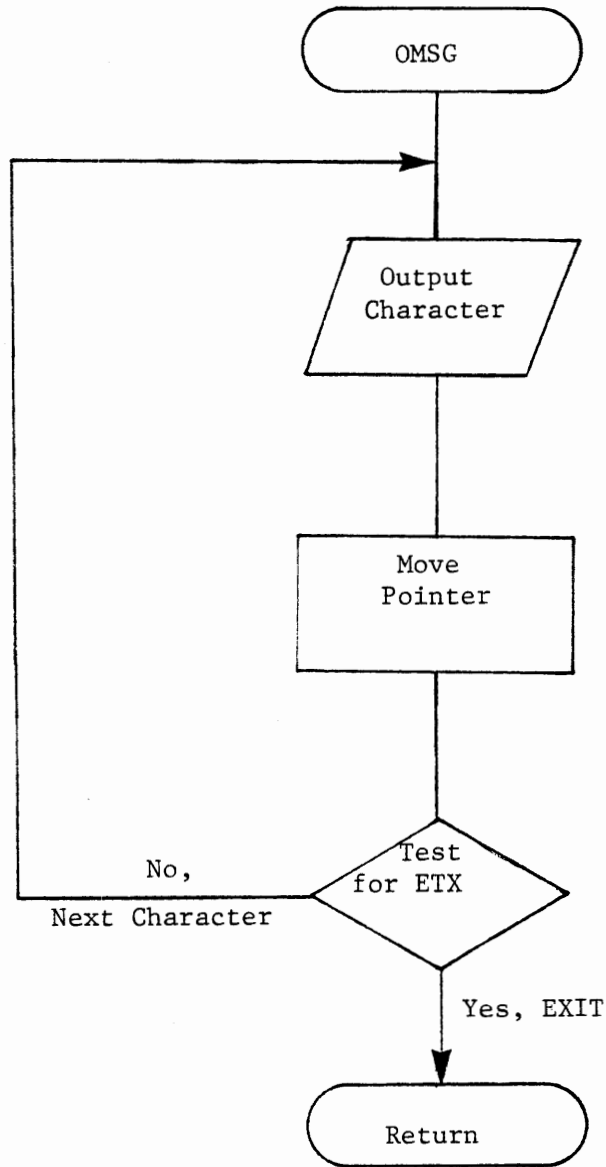


FIGURE E-9

Output the buffer whose address is in H,L pair
to terminal until an STX character is encountered

OMSG

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

82F6	5E	OMSG	M	O	V	E	M			Get next character	
F7	CD		C	A	L	L	O	C	H	R	Output it
F8	DO										
F9	82										
FA	23		I	N	X	H					Move pointer
FB	93		S	U	B	E					Compare to 03
FC	F0		R	P							Exit if 00, 01, 02, or 03
FD	C3		J	M	P	O	M	S	G		else go for next character
FE	F6										
82FF	82										

8

FIGURE E-10

E.3.5.1 Main Calling Program

This routine exercises the RS 232 system by calling the appropriate subroutine modules to output a message to the terminal, accept an input message (with echo), and output the inputted string. The program then repeats the procedure.

* Entry point 8200H

* Arguments *none*

* Subroutines used

- IMSG at 8290 to prompt and then input a message
- OMSG at 82F6 to output a character string message

* Sample usage procedure

- 1 Load the RS 232 System from the Cassett Library
- 2 Verify memory using the enclosed listings
- 3 Press RST , RUN

4 The display should go blank, and the terminal should display:

Hi There. I am your friendly computer running an RS232 Interface
program.

What is your name?

5 The system is now awaiting keyboard input. Type in your response
followed by a Carraige Return character (Press the RETURN key).

6 The system will respond with

Hello <your response>

Hi There. I am ... etc.

7 The system will repeat the sequence ad nauseam.

* Data Tables

- An output buffer is provided at addresses 8300H-8362H with the "Hi There ..." message
- An output buffer is provided at addresses 83A0H - 83A8H with the "Hello " character string.
- An input buffer is provided at addresses 8370H-838FH for the response character string.

NOTE You may wish to experiment with your own messages and input and output sequences. There is ample space in the 512 bytes of RAM to code your own MAIN Calling program with your own messages. However, note that OMSG expects an ETX (03H) character as the terminating character in the output buffer.

If you have the 1K RAM option, you may wish to use the alternate IMSG routine provided at address 8000H with basic editing capabilities (Underscore for delete character, CTRL-X for delete line). The alternate IMSG is provided in section E.4.

E.3.5.2

Main Flowchart

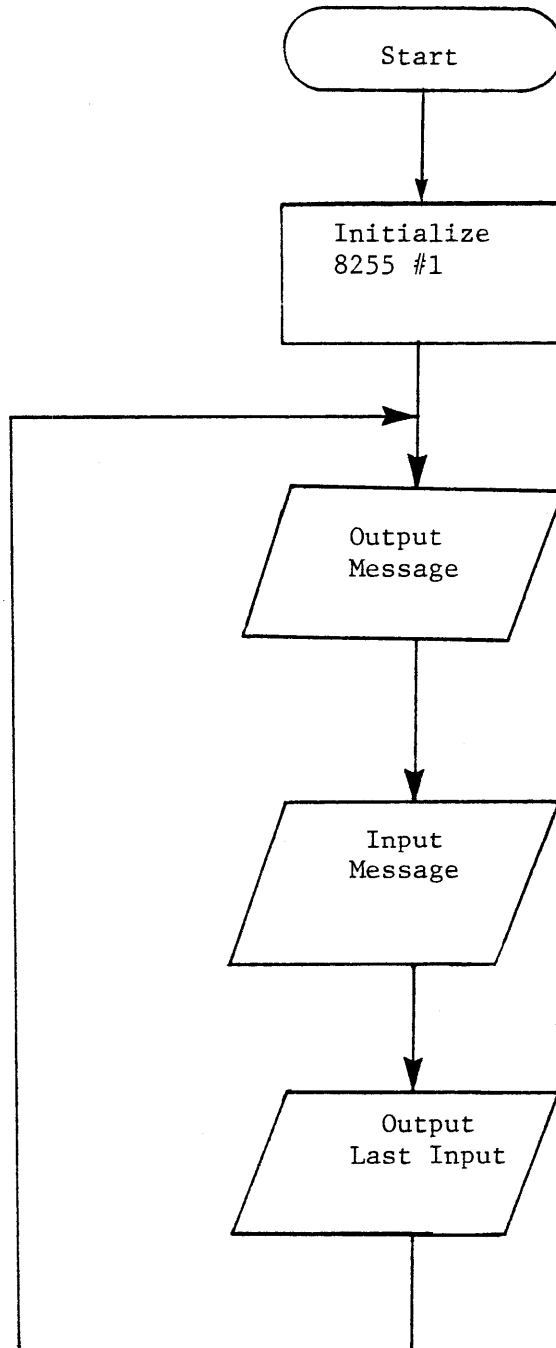


FIGURE E-11

RS 232 Interface experiment program

c/7/78

MAIN CALLING PROGRAM

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8200	3E	MAIN	MVI	A,	92H	Initialize 8255 #1
1	92					
2	D3		OUT	CNT1		
3	07					
8204	21	LOOP	LXI	H,	8300	Output buffer address
5	00					
6	83					
7	CD		CALL	OMSG		
8	F6					
9	82					
A	2E		MVI	L,	70H	Input buffer address
B	70					
C	0E		MVI	C,	1F	Input buffer maximum length - 1
D	1F					
E	CD		CALL	PMPRT		Prompt with "?" and input a message
F	9D					
8210	82					
1	2E		MVI	L,	A0H	"Hello" address
2	A0					
3	CD		CALL	OMSG		
4	F6					
5	82					
6	2E		MVI	L,	70H	Name buffer address
7	70					
8	CD		CALL	OMSG		
9	F6					
A	82					
B	C3		JMP	LOOP		
C	04					
D	82					
E						
F						

FIGURE E-12

Output Buffer

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8 3 0 0	0 A	OBUF	D B		L F				Line Feed
1	0 D		D B		C R				Carraige Return
2	4 8		D B		H				
3	6 9		D B		i				
4	2 0		:	:					
5	5 4		:	:	T				
6	6 8		v		h				
7	6 5			v	e				
8	7 2				r				
9	6 5				e				
A	2 E				.				
B	2 0								
C	7 0								
D	4 9				I				
E	2 0								
F	6 1				a				
8 3 1 0	6 D				m				
1	2 0								
2	7 9				y				
3	6 F				o				
4	7 5				u				
5	7 2				r				
6	2 0								
7	6 6				f				
8	7 2				r				
9	6 9				i				
A	6 5				c				
B	6 E				n				
C	6 4				d				
D	6 C				l				
E	7 9		D B		y				
8 3 1 F	2 0		D B						
8	0								
1									
2									

FIGURE E-13

RS 232 Exercise Data Tables cont

Output Buffer, cont

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8	32	0	6	3		D	B		c										
		1	6	F		D	B		o										
		2	6	D		.	.		m										
		3	7	0		.	.		p										
		4	7	5		v	.		u										
		5	7	4			v		t										
		6	6	5					e										
		7	7	2					r										
		8	2	0															
		9	7	2					r										
		A	7	5					u										
		B	6	E					n										
		C	6	E					n										
		D	6	9					l										
		E	6	E					n										
		F	6	7					g										
8	33	0	2	0															
		1	6	1					a										
		2	6	E					n										
		3	2	0															
		4	2	0															
		5	5	2					R										
		6	5	3					S										
		7	3	2					2										
		8	3	3					3										
		9	3	2					2										
		A	4	3					C										
		B	2	0															
		C	4	9					I										
		D	6	E					n										
		E	7	4					t										
	833	F	6	5		D	B		e										
8		0																	
		1																	
		2																	

FIGURE E-13b

RS 232 Exerciser Data Tables

Output Buffer, cont

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

B	34	0	7 2	DB	r				
		1	6 6	DB	f				
		2	6 1		a				
		3	6 3		c				
		4	6 5		e				
		5	2 0						
		6	5 0		P				
		7	7 2		r				
		8	6 F		o				
		9	6 7		g				
		A	7 2		r				
		B	6 1		a				
		C	6 D		m				
		D	2 E		.				
		E	0 A		L F				Line Feed
		F	0 D		C R				Carriage Return
B	35	0	5 7		W				
		1	6 8		h				
		2	6 1		a				
		3	7 4		t				
		4	2 0						
		5	6 9		l				
		6	7 3		s				
		7	2 0						
		8	7 9		Y				
		9	6 F		o				
		A	7 5		u				
		B	7 2		r				
		C	2 0						
		D	6 E		n				
		E	6 1		a				
		F	6 D		m				
B	36	0	6 5		e				
		1	2 0						
		2	0 3	DB	E T X				End of Buffer Character

FIGURE E-13c

RS 232 Exerciser Data Tables

c/7/78

Hello Output Buffer, Input buffer

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

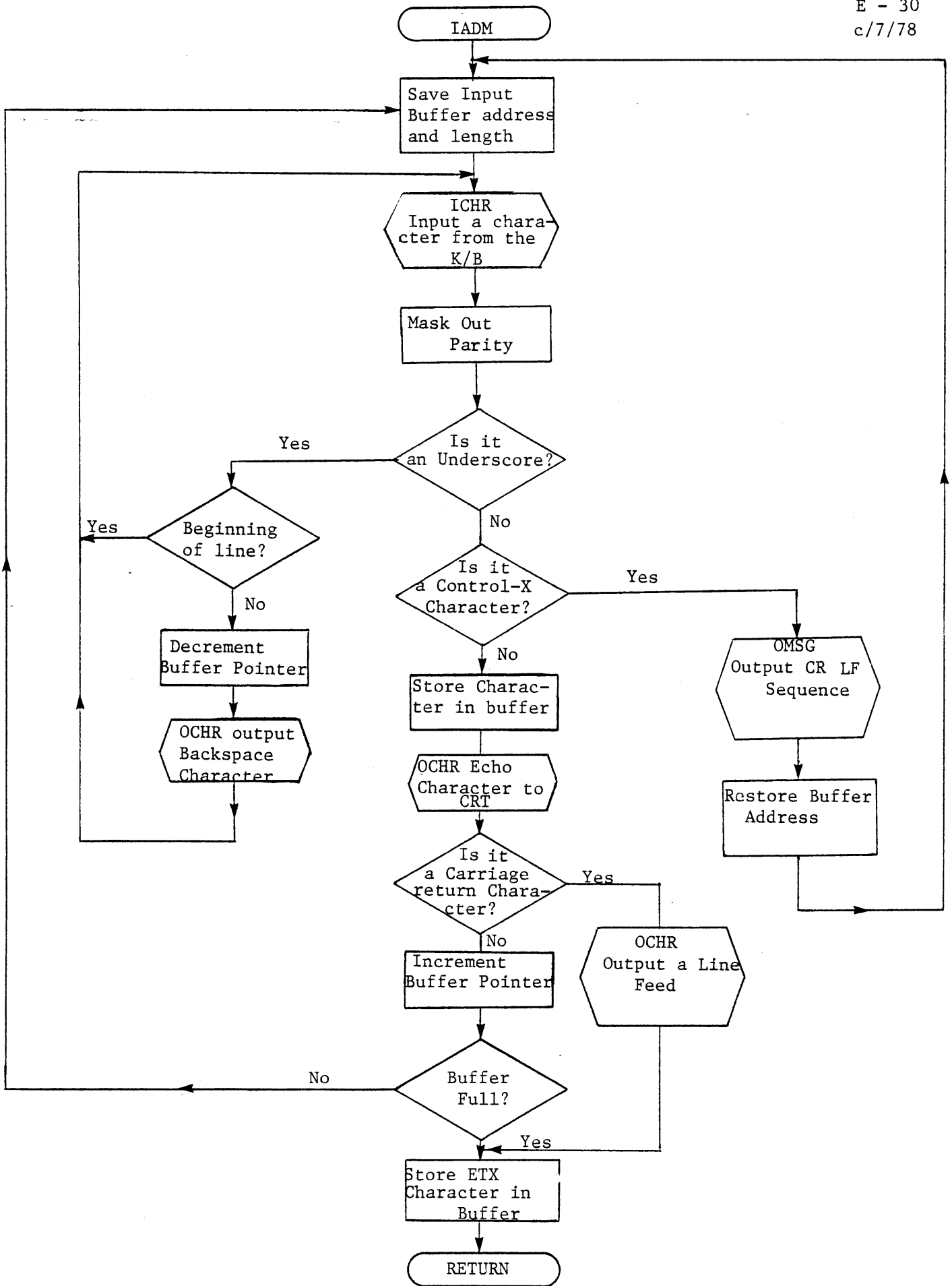
INTEGRATED COMPUTER SYSTEMS

8	3 A 0	0 A	HELLO	D B		LF			HELLO message
	1	0 D		D B		CR			Line feed carriage return
	2	4 8		D B		" H "			
	3	6 5		D B		" E "			
	4	6 C		D B		" L "			
	5	6 C		D B		" L "			
	6	6 F		D B		" O "			
	7	2 0		D B		SPACE			
	8	0 3		D B		ETX			End of buffer
	9								
	A								
	B								
	C								
	D								
	E								INPUT BUFFER
	F								
8	3 7 0		IBUF	D S		3 2 H			Reserve 32 bytes
	8 3 9 0								of RAM for 31 input
	2								characters + ETX → 32 bytes
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	A								
	B								
	C								
	D								
	E								
	F								
8	0								
	1								
	2								

FIGURE E-13d

E.4 Alternate IMSG program

This subroutine is identical to the IMSG routine with the following exceptions; a) Entry point is 8000H; 2) There is no PMPT entry; 3) An underscore character at the keyboard will delete the last character entered; 4) A CTRL-X (18H) character will delete the entire input line. The flowchart is shown in Figure E-14 and the code is in Figure E-15.



Alternate IMSG, Flowchart
FIGURE E-14

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

Input a string from CRT terminal
 handling underscore as a backspace character
 with ASCII as a carriage line character
 Input buffer address in High pair and memory
 location in C.

E-31

11/7/78

8000	E5	IADM	PUSH	H	H				Save buffer address
1	41		MOV	B	C				Save buffer length
8002	C5	LP2	PUSH	B					
8003	C7	LP2	CALL	ICHR					Get a character
4	B0								
5	B2								
6	B6		ANI	7F					Mask out parity bit
7	77								
8	F5		CPI	"_"					Test for underscore
9	5F								
A	C7		JZ	BKSP					Go backspace
B	2C								
C	D0								
D	F8		CPI	CAN					^X: Test for CNTL-X
E	18								
F	CA		JZ	DELN					Yes, Go Delete Line
8010	44								
1	B0								
2	77		MOV	M	A				Store the character in the input buffer
3	5F		MOV	E	A				Echo character to CRT
4	CD		CALL	ICHR					
5	D0								
6	B2								
7	C1		POP	B					
8	7B		MOV	A	E				
9	FE		CPI	CR					Is character a Carriage Return?
A	0D								
B	06		MVI	E	LF				
C	03								
D	CC		CN	ICHR					Yes, add Line Feed character and
E	D0								
F	B2								

FIGURE E-15 a

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8020	CA	JZ	EXIT						then Exit.
1	28								
2	80								
3	23	INX	H						Move buffer pointer
4	0D	DCR	C						Decrement character counter,
5	C2	JNE	LP1						and go get next
6	02								character, iff buffer not
7	80								full.
8028	36	EXIT	MVI	M,	03				Store ETX (03H)
9	03								character at end of MSG,
A	D1	POP	D						and return.
B	C9	RET							***
802C	C1	BKSP	POP	B					Back space 1 character:
D	D1	POP	D						Retrieve buffer address.
E	D5	PUSH	H	D					
F	7B	MOV	A,	E					
8030	BD	CMP	L						Already front of line,
1	CA	JZ	LP1						ignore backspace.
2	02								
3	80								
4	2B	DCX	H						Move buffer pointer back
5	0C	INR	C						one character.
6	C5	PUSH	H	B					
7	1E	MVI	E,	"^H"					Output a "move cursor
8	08								left" ADM-3 character
9	CD	CALL	OCHE	R					
A	D0								
B	82								
C	00								These NOPs leave room
D	00								to include code to output
E	00								a backslash/delete char-
F	00								

CODING SHEET

MICROCOMPUTER TRAINING SYSTEM

INTEGRATED COMPUTER SYSTEMS

8040	00		NOP							acter \ sequence (like D.E.C)
1	C3		JMP			LPZ				
2	03									Go get next character
3	80									
8044	21	DELN	LXI			H, CR LF				Delete entire line:
5	50									
6	30									
7	0D		CALH			CM SG				Output Carraige Return - Line Feed sequence.
8	1									
9	7									
A	0E		POP			B				Restore buffer address and character counter.
B	4B		MOV			C, F				
C	3E		POP			H				
D	01		JMP			IADM				Go back to beginning
E	0									
F	X0									
8050	0D	CR LF	DS			CR				CR, LF buffer
1	0A		LS			LF				
2	03		DS			FTX				
3										
4										
5										
6										
7										
8										
9										
A										
B										
C										
D										
E										
F										